Procesamiento paralelo con arquitecturas multi-dsp.

Alumno:

Luis Alberto Acosta Burllaile

Director:

Ing. Armando de Giusti

9/9/97 (

Memorando

Sr. Prof.

Bria

Remito a Ud. copia del Informe Final del Trabajo de Grado

Procesamiento paralelo con arquitecturas Multi-DSP

Se le solicita tenga a bien entregar la respuesta
antes del : **9/10/97**

Lic. Patricia Bazán
Coordinacion de Trabajos de Grado

Agradecimientos:

A Gregory Randall

A Delio Dirazar

A Alejandra Cosentino y Marcelo De Andrea

INDICE

1-Introducción:

En los últimos tiempos, una de las áreas que despierta mayor interés y que ha tenido más auge dentro del amplio campo de la informática es la del procesamiento paralelo. El paralelismo invade no sólo las super-computadoras, sino también las workstations, las computadoras personales y las redes. Los programas necesitan entonces explotar las capacidades de los múltiples procesadores localizados en cada computadora más los procesadores adicionales disponibles a través de una red.

En líneas generales, podemos definir una arquitectura paralela como un conjunto de procesadores que son capaces de trabajar en forma conjunta. La ventaja principal que ofrece este tipo de procesamiento es que permite concentrar los recursos computacionales (procesadores, memoria, entrada/salida, etc.) para resolver cualquier problema informático específico. Dado que la mayoría de los algoritmos existentes son secuenciales, surge la necesidad de desarrollar nuevos algoritmos y nuevas estructuras de programa que permitan aprovechar las ventajas mencionadas antes.

En nuestro caso particular, nos interesa investigar la aplicación de las arquitecturas paralelas para el procesamiento de imágenes digitales, específicamente para el reconocimiento de patrones.

Teniendo en cuenta esto último, hemos orientado nuestro trabajo hacia arquitecturas paralelas tipo CISC específicas para el procesamiento de señales: DSP y multi-DSP; si bien también se utilizaron TRANSPUTERS (arquitecturas tipo RISC) como punto de referencia para evaluar los resultados.

Se planteó como objetivo el desarrollo y la implementación de un algoritmo paralelo que realizara el reconocimiento de formas geométricas en el plano (

cuadrados, círculos y triángulos), y la evaluación de las arquitecturas paralelas citadas anteriormente mediante la ejecución de dicho algoritmo.

Nuestro trabajo se puede dividir entonces en dos áreas principales: el procesamiento de imágenes digitales, que culmina con la implementación del algoritmo reconocedor paralelizable; y las arquitecturas paralelas (dsp , multi-dsp y transputers) que culmina con la evaluación de dichas arquitecturas mediante los resultados obtenidos con la ejecución del algoritmo reconocedor.

En una primera etapa, se procedió a estudiar el procesamiento de imágenes digitales, poniendo especial interés en las técnicas existentes que se aplican para realizar el reconocimiento de patrones.

Una vez estudiadas y considerando que no se adaptaban totalmente para resolver nuestro problema, se procedió a implementar un algoritmo propio, en base a la extracción y clasificación de atributos de la imagen a procesar.

El algoritmo se implementó primero en forma secuencial en lenguaje C y se probó en una PC 486; asimismo se lo implementó en lenguaje OCCAM y se lo probó en un transputer. Seguidamente, se lo implementó en el lenguaje C del dsp TMS320C3x y se lo probó en el simulador del dsp.

Luego se realizó la paralelización del algoritmo. Una vez paralelizado, se lo implementó en una primera etapa en lenguaje OCCAM y se lo probó en una red de transputers; posteriormente se lo implementó en el lenguaje C del dsp TMS320C3x y se lo probó en el ambiente de simulación multi-dsp.

Considerando que no se cuenta con las arquitecturas dsp y multi-dsp, los algoritmos realizados se probaron utilizando el simulador del dsp TMS320C3x disponible. Se estudió la arquitectura del dsp TMS320C3x, así como también las herramientas de generación de código y el ambiente de simulación. Una vez puesto en marcha el ambiente de simulación, y probado en él el algoritmo

secuencial, se realizó la extensión del ambiente de simulación para que pudiera simular una arquitectura multi-dsp.

Una vez extendido el ambiente utilizando NETBIOS como soporte de comunicación entre los diversos procesadores, se procedió a implementar librerías (en el lenguaje C del dsp TMS320C3x) que proveyeran las funciones de comunicación y sincronización entre los distintos procesos, ajustándose al modelo de pasaje de mensajes sincrónico.

A partir los resultados obtenidos, se realiza una comparación de las arquitecturas transputers contra las arquitecturas dsp y multi-dsp, tanto desde el punto de vista de la performance, como así también del esfuerzo requerido y de las herramientas disponibles para el desarrollo y la implementación del software. Además se evalúa la extensión del simulador de dsp a una máquina multi-dsp.

2-Procesamiento de imágenes:

En general, el procesamiento de imágenes se refiere a la manipulación y al análisis de la información de una imagen visual de dos dimensiones. Cualquier operación que mejore, corrija, analice o cambie en algún aspecto la imagen original está incluída en este concepto. El objetivo del procesamiento de imágenes es transformar o analizar una imagen de tal forma que se pueda obtener más información acerca de ella. En nuestro caso particular, nos interesa el procesamiento de imágenes digitales.

En el campo digital, una imagen está representada por puntos discretos con brillo definido numéricamente. Manipulando dichos valores de brillo, la computadora puede realizar operaciones complejas con relativa facilidad.

Existen numerosas y variadas operaciones de procesamiento de imágenes; algunas se utilizan para mejorar la calidad de las imágenes mientras que otras sirven para extraer información de ellas automáticamente. Se pueden agrupar en cinco clases fundamentales: realce, restauración, análisis, compresión y síntesis.

Análisis de imágenes:

Las operaciones de análisis de imágenes son usadas en aplicaciones que requieren mediciones y clasificaciones de la información de la imagen. El objetivo de las operaciones de análisis de imágenes es "comprender" una imagen cuantificando sus elementos. La cuantificación incluye aspectos tales como mediciones de tamaño, indicadores de forma, descripciones, etc. Otros elementos de interés pueden incluir atributos tales como brillo, color y textura. Las operaciones de análisis de imágenes juegan un rol fundamental en la visión

automatizada por computadora y en las aplicaciones de interpretación de imágenes.

Una imagen existe como un gran arreglo de pixeles con sus brillos asociados. Esta forma de imagen nativa no contiene inteligencia inherente acerca del contenido. Esa interpretación es dejada completamente al observador. Las operaciones de análisis de imágenes dividen a estas en una lista concisa de objetos individuales que satisfacen ciertas medidas y/o descripciones específicas, convirtiéndola de una forma visual a una forma descriptiva. El resultado es una forma de reducción drástica de la imagen que provee reconocimiento e identificación de objetos. El análisis de imágenes se realiza a través de la siguiente secuencia de procesos:

- Segmentación: simplifica la imagen reduciéndola a sus elementos básicos. Es una operación que aísla objetos individuales en una imagen. Su objetivo es simplificar la imagen sin alterar ni descartar las características importantes. Se puede dividir en pre-procesamiento, discriminación de los objetos y afinado de los bordes.

- Extracción de características: una vez que la imagen ha sido segmentada en objetos discretos de interés, el siguiente paso es medir las características individuales de cada objeto (varias características pueden ser usadas para describir un objeto). En cualquier aplicación de análisis de imágenes el objetivo es usar la menor cantidad posible de medidas para caracterizar un objeto de tal forma que dicha caracterización no resulte ambigua.

- Clasificación de objetos: el proceso de clasificación de objetos implica comparar un conjunto de mediciones de las características del objeto (valores de los atributos) con otro conjunto de valores conocido, o con algún criterio establecido. A su vez, se deben establecer los márgenes de tolerancia que

permitan determinar si el objeto pertenece o no a una determinada clase. Este proceso de clasificación se puede dividir en tres etapas: determinar las características del objeto que queremos usar para la clasificación, establecer los márgenes de tolerancia, y crear grupos o categorías de clasificación, a los cuáles un objeto será asignado o no dependiendo de la comparación de las medidas con el criterio establecido.

En nuestro caso, hemos utilizado un caso específico del análisis de imágenes: el reconocimiento de patrones; concentrándonos en los procesos de extracción de características y clasificación de objetos.

Reconocimiento de patrones:

Se define un "patrón" como una descripción estructural o cuantitativa de un objeto o cualquier otra entidad de interés en una imagen. En general un patrón está formado por una disposición de uno o más descriptores (características). [GON92]

El reconocimiento de patrones en imágenes digitales consiste en tratar de hallar la presencia de los mismos dentro de una imagen digital. Los patrones que se analizan pueden ser formas, texturas, imágenes complejas con formas y colores, un conjunto de pares atributo-valor, etc.

Existen diferentes técnicas para realizar el reconocimiento de un patrón Px en una imagen, una de éstas consiste en hallar exactamente Px dentro de la imagen (por ejemplo, pattern matching); otra alternativa es hallar una subimagen Yx similar a Px sin necesidad de ser idéntica. Las diferencias que se toleran pueden referirse a tamaño, rotación, pequeñas deformaciones, etc.

A partir de lo enunciado arriba, los procesos de reconocimiento de patrones serán variados; podemos tener:

a Patrones que se reconocen por la coincidencia exacta con la imagen analizada. Rara vez se requiere un análisis de este tipo, ya que la forma en que se generan las imágenes digitales hacen imposible obtener imágenes idénticas.

b Búsqueda en la imagen de alguna porción que se parezca mucho al patrón, aunque no sea exactamente igual; el patrón buscado no podrá entonces estar rotado o escalado con respecto al patrón original. Es apropiada la técnica de correlación [JAI89].

c Buscar en la imagen la presencia del patrón como una porción de la imagen donde ciertos atributos clasificadores del patrón (valores de esos atributos) se dan con valores que se mantienen dentro de cierta tolerancia admisible. Para esto uno debe extraer atributos del patrón (o recibir el patrón de esta manera), analizar qué valores toman esos atributos para el patrón dado, definir un margen de tolerancia para cada atributo, y luego decir que una imagen satisface el patrón si ocurre que al extraerle esos mismos atributos, el valor de cada uno de ellos se halla dentro de los límites de tolerancia de ese atributo para el patrón dado.

En general se identifican dos etapas, en una primera fase se extraen los atributos que describen a la imagen que se analiza y en la segunda se clasifican dichos atributos con el fin de determinar si se corresponden o no con el patrón.

3-Planteo del Problema. Aplicaciones.

Como una primer etapa para problemas de reconocimiento de patrones aplicados a Robótica se fija como objetivo analizar la identificación de tres tipos de figuras simples (cuadrados, círculos y triángulos) a partir de imágenes digitalizadas de las mismas.

Esta clase de soluciones apuntan al funcionamiento de robots simples de manipulación, que pueden reconocer un objeto en el plano para tomarlo o realizar otra acción controlada por software.

No es importante en una primera etapa la complejidad de la imagen digitalizada (de hecho, en un problema real siempre iba a ser posible construir una base de conocimiento con un conjunto de atributos reconocibles, o bien descomponer el patrón complejo en patrones más simples), sino la posibilidad de tener un algoritmo seguro y paralelizable.

Se considera adecuado utilizar una técnica de reconocimiento basada en atributos, a fin de no depender de ubicación, orientación o tamaño de la figura específica.

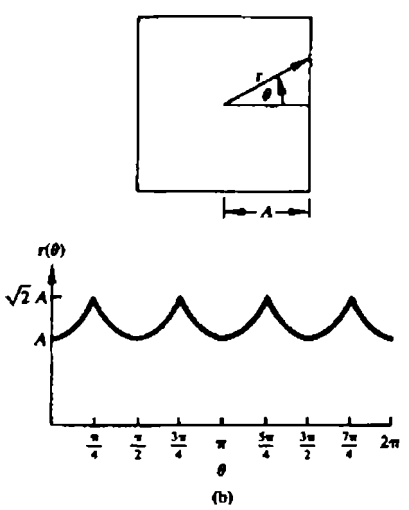
Interesa particularmente analizar dificultades posibles del algoritmo (discriminación equivocada de una figura, margen de error admisible, mínimo tamaño manejable, discriminación de figuras falsas, etc.).

Las imágenes utilizadas son bi-level con formato BMP, la figura es negra y el fondo es blanco. Además, se requiere que los bordes no tengan ramificaciones y que la figura no tenga huecos en su interior.

4-Algunas Técnicas Alternativas para la Identificación de Atributos:

Para realizar el desarrollo del sistema se analizaron diversas técnicas utilizables en el reconocimiento de patrones entre las que se pueden mencionar:

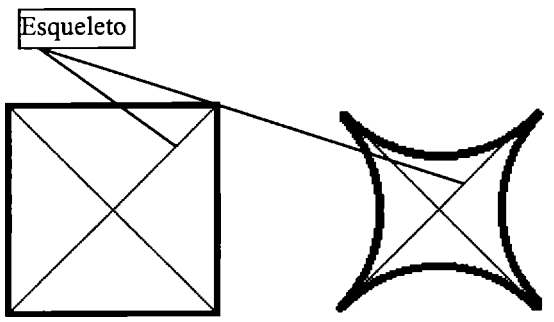
1 Signature: es una representación del borde por medio de una función de una dimensión. Una de las maneras más simples de obtenerla es tomando la distancia del centro de masa al borde como una función del ángulo.



Se podría realizar un reconocimiento de patrones analizando la forma de la función resultante. No se adoptó esta técnica ya que reconocer la forma de la signature es tan complejo como analizar la forma de la figura. [GON92]

2 Esqueletonización: Esta técnica extrae el eje medio de la figura que se

analiza; dejando simplemente un grafo. No se utilizó esta técnica pues figuras diferentes tienen el mismo esqueleto. [GON92]



3 Correlación: Se basa en hallar una porción de la imagen que sea muy parecida a la imagen del patrón haciendo superposición de las mismas. La técnica no es apropiada en este caso ya que requiere que el patrón tenga un tamaño y posición similar a la figura analizada.

4 Descripción explícita: Se obtiene creando una lista secuencial de las posiciones de los pixeles del borde. No es apropiada para usarse en operaciones de clasificación ya que la descripción es muy sensible a la forma en que el objeto aparece en la imagen. Cambios en la localización, orientación y tamaño del objeto alteran las descripciones, haciendo imposible la comparación entre ellas.

5 Códigos cadena: Esta técnica se basa en recorriendo los puntos del borde y almacenando la dirección que se sigue al ir de uno a otro. El resultado es una lista de códigos que muestra la dirección que fue seguida moviéndose de cada punto del borde al siguiente. Este código es llamado código de cadena absoluto, ya que representa la dirección absoluta con respecto a la

dirección 0. Una variante es el código de cadena relativo, que se utiliza para almacenar los movimientos relativos. El código absoluto es sensible a la forma (orientación) en que el objeto aparece en la imagen; mientras que el relativo no lo es. Sin embargo, ambos son sensibles al tamaño del objeto.

5-Algoritmo desarrollado:

Si bien las técnicas estudiadas podrían ayudar al reconocimiento de figuras particulares, para el objetivo planteado se definieron nuevas técnicas que se adaptaran mejor al reconocimiento de dichas figuras, es decir, que permitieran reconocer cuadrados, círculos y triángulos por sus características propias sin tener en cuenta el tamaño y posición de las mismas (lo cual constituía el mayor inconveniente de las técnicas estudiadas).

La figura a reconocer se encuentra en un archivo de imagen bi-level con formato BMP, la figura es negra y el fondo es blanco.

Como el primer paso del desarrollo, lo que se hace es extraer el borde de la figura, el cuál queda almacenado en memoria RAM y será usado para realizar el reconocimiento.

Durante esta recorrida sobre la imagen se extraen también la superficie y el perímetro. La superficie es la cantidad de puntos negros que se encuentran sobre la imagen, y el perímetro es la longitud del borde de la figura.

Para reconocer los objetos y poder clasificarlos, se tiene en cuenta el perímetro y la superficie de la figura, los cuáles en este caso particular son conocidos.

Con el borde de la imagen lo que se hace es buscar los dos puntos más extremos; una vez hallados se pasa a analizar si la figura es un círculo, un cuadrado, o un triángulo.

Técnica aplicada para reconocer círculos:

Dados los dos puntos extremos del borde de una figura (los más distantes), se supone que si es un círculo, la distancia entre los dos puntos será el diámetro. Sabiendo que el perímetro de un círculo es π * diámetro, y la superficie π * radio^2, siendo el radio = diámetro / 2, se calculan ambos, perímetro y superficie, y se comparan con la superficie y perímetro reales de la figura, que fueron tomados en el momento en que se extrajo el borde.

Si la comparación de ambas medidas es exitosa, es decir que tanto superficie como perímetro reales (los tomados de la figura), varían con respecto a las medidas de un círculo perfecto por sólo un error apropiadamente acotado, entonces se afirma que la figura es un círculo.

Técnica aplicada para reconocer cuadrados:

La distancia entre los dos puntos más extremos del borde constituye, si se supone que la figura es un cuadrado, la diagonal.

Para calcular la superficie y el perímetro, lo que se necesita es el lado, el cuál se obtiene a partir de la diagonal utilizando el Teorema de Pitágoras. La superficie del cuadrado es lado^2 y el perímetro es 4 * lado. Para decidir si es cuadrado se comparan superficie y perímetro reales (los extraídos de la figura) contra los de un cuadrado perfecto, análogamente a lo analizado en el círculo. Si esta comparación fuera exitosa no se asegura que la figura sea un cuadrado, ya que perímetro y superficie pueden compensarse de manera que un rectángulo pueda ser aceptado.

Para solucionar el problema mencionado, lo que se hace es verificar que los lados del cuadrado tengan la misma longitud. La forma de llevar a cabo este

lados del cuadrado tengan la misma longitud. La forma de llevar a cabo este cálculo consiste en hallar los cuatro puntos del cuadrado perfecto (dos de éstos son los puntos más extremos) y verificar que el borde los contenga.

Si los lados tienen la misma longitud se considera que la figura es un cuadrado.

Técnica aplicada para reconocer triángulos:

Dados los dos puntos más alejados del borde, si se supone que la figura es un triángulo, la distancia entre estos dos puntos es la base del triángulo.

El perímetro es la suma de las longitudes de los tres lados, y la superficie es (base * altura) / 2. Para calcular ambas medidas hace falta conocer el tercer punto del triángulo.

Para hallar el tercer punto se toma el punto más distante a la base en línea perpendicular a ella. Una vez obtenido este punto, se tienen la altura y las longitudes de los lados, los que permiten calcular la superficie y el perímetro del triángulo perfecto, y compararlos con los datos reales tomados durante la extracción del borde.

Análogamente a las técnicas anteriores, si la comparación es exitosa, se dice que la figura es un triángulo.

Si una figura no es reconocida por ninguna de estas técnicas, se dice que la figura no tiene clasificación.

6-Procesamiento paralelo:

Introducción:

En líneas generales las arquitecturas paralelas consisten en un conjunto de procesadores que se hallan distribuidos dentro de un sistema de cómputos para realizar una o más funciones. El punto a resolver con el uso de multiprocesadores es cómo codificar la solución de un problema dado para aprovechar las ventajas que ofrece este tipo de arquitectura.

La performance de una computadora depende directamente del tiempo requerido para ejecutar una operación básica y del número de operaciones básicas que puedan ser ejecutadas concurrentemente. El tiempo requerido para ejecutar una operación básica está limitado por el ciclo de reloj del procesador. Los tiempos del ciclo de reloj han ido decreciendo y parecen estar llegando a los límites físicos (la velocidad de la luz). Para superar estos límites surge la idea de utilizar concurrencia interna en un chip. Así, en una computadora simple tenemos, por ejemplo, las siguientes técnicas: pipeline, unidades de función múltiple (sumadores, multiplicadores, etc.), etc. Posteriormente surgen las computadoras múltiples, cada una con su procesador, su memoria y su lógica de interconexión asociada.

Hay tres factores importantes que se deben tener en cuenta en el desarrollo de software paralelo: la concurrencia, la escalabilidad y la localidad. La concurrencia hace referencia al manejo de los recursos compartidos en los casos en que varias instrucciones intentan acceder a dichos recursos al mismo tiempo; la escalabilidad indica la posibilidad que ofrece el software para adaptarse al incremento del número de procesadores; y la localidad establece que los accesos a memoria local tienen que ser más "barato" que el acceso a memoria remota.

Se necesitaría contar con algoritmos especiales y estructuras de datos que permitan especificar concurrencia. Aún hoy, programar con paralelismo es muy dificultoso para la mayoría de los programadores, debido a que los lenguajes de programación existentes fueron desarrollados originalmente para computadoras secuenciales. Los programadores se ven a menudo forzados a programar en forma dependiente del hard, en vez de programar paralelismo en forma más genérica y portable. Idealmente se necesitaría desarrollar un ambiente de programación paralelo, con lenguajes, compiladores y herramientas de soft independientes de la arquitectura.

Frente a la falta de lenguajes citada anteriormente, uno podría tratar de extender los lenguajes existentes o bien construir uno nuevo. La construcción de un nuevo lenguaje tiene la ventaja de usar constructores explícitos de alto nivel para especificar paralelismo. Sin embargo, a menudo son incompatibles con los lenguajes existentes y requieren nuevos compiladores. Debido a esto, la mayoría de los sistemas optan por extender los lenguajes convencionales.

Con respecto a los compiladores, hay tres opciones: pre-procesadores, pre-compiladores y compiladores paralelizables. Un pre-procesador usa un compilador secuencial y una librería de bajo nivel de la computadora para implementar constructores paralelos de alto nivel. Un pre-compilador, requiere un análisis del flujo del programa, chequeo de dependencias, y optimizaciones limitadas a través de la detección de paralelismo. Un compilador paralelizable, demanda un desarrollo completamente paralelizable o un compilador vectorizante que pueda automáticamente detectar paralelismo en el código fuente y transformar el código secuencial en constructores paralelos. Debido al comportamiento impredecible de los programas, ninguno de los compiladores existentes puede ser considerado totalmente automático o "inteligente" de forma tal que pueda detectar todos los

tipos de paralelismo. A menudo se hace necesario insertar directivas al compilador en el código fuente para ayudar al compilador a trabajar mejor. Los usuarios deben interactuar con el compilador para reestructurar los programas.

Además los sistemas operativos también deben ser extendidos de forma tal que soporten actividades paralelas. Así, los sistemas operativos deben ser capaces de manejar los recursos en un ambiente de paralelismo, teniendo en cuenta aspectos tales como scheduling paralelo de eventos concurrentes, alocación de memoria compartida, periféricos compartidos y links de comunicación.

Paralelismo de hardware y de software:

Podemos tener paralelismo en el hardware y paralelismo en el software. El paralelismo en el hardware se refiere al tipo de paralelismo definido por la arquitectura de la máquina y por la multiplicidad del hardware. Muestra los patrones de utilización de los recursos por las operaciones de ejecución simultánea. Una forma de caracterizarlo es el número de instrucciones simultáneas por ciclo de máquina.

Con respecto al paralelismo de software, el mismo está definido por el control y la dependencia de datos de los programas. El grado de paralelismo es revelado en el perfil del programa o en el grafo de control de flujo del programa. Depende muchísimo del estilo de programación, del algoritmo propiamente dicho y del grado de optimización del compilador. De los muchos tipos de paralelismo de software, dos son los más frecuentemente citados: el paralelismo de control, que permite la ejecución de dos o más operaciones simultáneamente; y el paralelismo de datos, en el cual la misma operación es ejecutada sobre muchos datos por muchos

procesadores simultáneamente.

Clasificación de las arquitecturas paralelas:

Hay dos grandes clases de computadoras paralelas: los multi-pocesadores con memoria compartida y las multi-computadoras que utilizan pasaje de mensajes. La principal diferencia entre multi-procesadores y multi-computadoras reside en la memoria compartida y en los mecanismos usados para la comunicación entre los procesadores. Los procesadores en un multi-procesador se comunican con otros a través de variables compartidas en una memoria común. En cambio, en una multi-computadora los procesadores tienen memoria local, no comparten memoria con otros. La comunicación inter-procesador se realiza a través del pasaje de mensajes.

En nuestro caso particular, los dos tipos de arquitecturas paralelas utilizadas responden al modelo de pasaje de mensajes.

Otras dos maneras para clasificar las arquitecturas paralelas: [NIE90]

1 Clasificación de Flynn: las clasifica en base al grado de paralelismo de los flujos de instrucción y de datos.

2 Clasificación basada en la organización de la memoria y del procesador.

1. a) SISD: flujo simple de instrucción y simple de datos (estructura básica de von Neuman)

b) SIMD: flujo simple de instrucción y flujo múltiple de datos.

c) MISD: flujo múltiple de instrucción y flujo simple de datos.

d) MIMD: flujo múltiple de instrucción y flujo múltiple de datos.

2. a) Procesadores Vectoriales: son usados para realizar operaciones sobre vectores de datos. La característica principal es que se realiza la misma operación sobre todos los elementos del vector al mismo tiempo (simd).

b) Arreglo de procesadores: una serie de procesadores que están conectados en una grilla rectangular. Cada procesador tiene su set de instrucciones y memoria local, y el conjunto de procesadores es controlado por una computadora o Main-Frame. Hay un arreglo de memoria que refleja la estructura del arreglo en procesamiento. Cada procesador se puede comunicar con sus vecinos adyacentes. Los procesadores son de propósito especial y se encuentran acoplados a través de intercambio de datos sincronizado. Los arreglos de procesadores son máquinas SIMD, ya que las instrucciones recibidas son operadas simultáneamente sobre todos los procesadores de la grilla.

c) Hipercubos: contienen un conjunto de n procesadores, donde n es potencia de 2, con los procesadores ubicados en las esquinas del cubo virtual y las conexiones entre los procesadores forman los bordes del cubo. La dimensión del cubo se expresa como $\log_2 n$, y las arquitecturas de dimensión mayor que 3 son denominadas hipercubos.

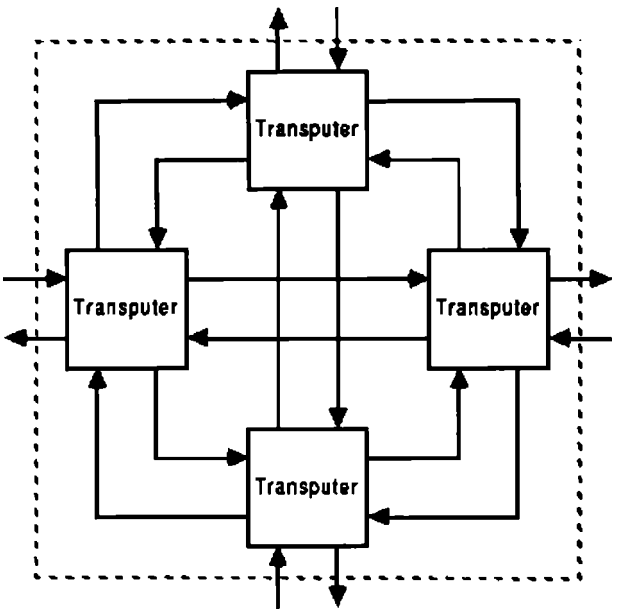
Los nodos pueden compartir memoria o tener su propia memoria local. Pueden formar máquinas SIMD o MIMD. Los procesadores pueden ser agregados con mejoras de performance casi lineales. Una ventaja del hipercubo es que otras topologías pueden ser consideradas subconjuntos del hipercubo; esto implica que programas escritos para otras topologías pueden ser transferidos con relativa facilidad.

d) Data Flow: está basada en el concepto de que la disponibilidad de datos

controla el flujo de ejecución. Las operaciones sobre los operandos tienen lugar cuando los operandos están listos y no cuando son requeridos por la secuencia de instrucción; esto implica que las sentencias de programa no son necesariamente ejecutadas en el orden textual.

Arquitecturas para el tratamiento de señales:

Transputers: un transputer puede ser clasificado como un miniprocesador RISC de alta performance, con memoria on-chip y links seriales. Un único transputer no representa una arquitectura paralela distribuida pero provee un bloque de construcción atractivo para construir sistemas de procesamiento paralelo. Sus características de diseño hacen que sea una arquitectura potente para el tratamiento de señales. Algunas de estas características son: simplicidad, regularidad, ausencia de interfaz de bus entre los procesadores, un alto grado de escalabilidad, alta performance y alta conectividad. Además las instrucciones son implementadas en microcódigo, lo que permite que el set de instrucciones sea extendido para una aplicación específica.



DSP: el dsp es un procesador CISC dedicado diseñado específicamente para el procesamiento digital de señales. Las características salientes de este tipo de procesador son: operaciones adaptadas al procesamiento de señales implementadas en hardware, modos de direccionamiento adaptados al procesamiento de señales, gran espacio de direccionamiento, interface multi-procesador, memorias y buses independientes, paralelismo interno. Además, gracias a los dos ports de interface externa, se pueden conectar varios dsps para formar una máquina multi-dsp.

7-El DSP TMS320C3x

La arquitectura del dsp TMS320C3x responde a las demandas de sistemas que están basados sobre algoritmos de aritméticas sofisticadas que requieren soluciones tanto desde el punto de vista del hardware como del software. La alta performance de este tipo de arquitectura de 32 bits está lograda a través de las siguientes características: la alta precisión y el amplio rango dinámico de las unidades de punto flotante, memoria on-chip, un alto grado de paralelismo y un controlador de acceso directo a memoria.

El TMS320C3x tiene una arquitectura de cpu basada en registros. La cpu consta de los siguientes componentes: multiplicador de enteros/punto flotante, ALU, shifter barrer de 32 bit (usado para desplazamientos de hasta 32 bits en un ciclo simple de máquina), buses internos, registros auxiliares de las unidades aritméticas y el CPU file register que es un conjunto de 28 registros de propósito general que pueden ser usados tanto por el multiplicador como por la ALU. Además el dsp posee una cache de programa, memorias internas de acceso dual, un canal DMA que soporta entrada/salida concurrente y un ciclo de máquina corto.

El TMS320C3X puede ejecutar en paralelo operaciones de multiplicación y operaciones de la ALU en un ciclo simple de máquina, tanto sobre enteros como sobre punto flotante. Además, los buses separados de programa, datos y DMA permiten a los programas realizar en paralelo búsquedas, accesos a datos y accesos a DMA. El tiempo de ciclo simple de 40 ns permite ejecutar operaciones a una performance promedio de hasta 60 millones de instrucciones de punto flotante por segundo (MIPFS) y de 30 millones de instrucciones por segundo (MIPS). [TMS94]

El espacio total de memoria disponible del TMS32C3x es de 16 millones de palabras de 32 bits. Programas, datos y el espacio de entrada/salida están incluídos en esos 16 millones. Cada bloque de RAM y ROM es capaz de soportar dos accesos de CPU en un ciclo simple de máquina.

El ambiente de simulación:

En el laboratorio se cuenta con un simulador del dsp TMS320C3x, el cual permite realizar el debugging del código desarrollado para el dsp, ya sea en lenguaje C o en lenguaje assembler.

Además del simulador se cuenta con un compilador del lenguaje C, el cual nos permite obtener a partir del código fuente escrito en el C particular del DSP (compatible con el ANSI C, pero más restringido), el código que será utilizado por el linker. A su vez se cuenta con un optimizador del compilador C, el cual permite realizar optimización en dos niveles: optimización general y optimización específica de la arquitectura. El código resultante es tomado por el linker, que se encargará de generar el código a ejecutarse en el simulador.

El simulador cuenta con dos ambientes de debugging: un ambiente básico de propósito general y un ambiente de comportamiento. El ambiente básico permite seguir la ejecución del programa línea por línea, pudiendo observar el contenido de las variables, de los registros de la cpu y de la memoria en cada punto de la ejecución. El ambiente de comportamiento permite recolectar estadísticas acerca de la ejecución del código, como por ejemplo el tiempo de ejecución de una función en particular, o de una porción del programa, la cantidad de veces que es invocada una función, la dirección de memoria de una línea en particular, y otros.

El ambiente de simulación es fácil de usar y de aprender, con una interfaz orientada a ventanas; permite crear ventanas para visualizar variables, estructuras, arreglos, etc. Permite además direccionar posiciones de memoria y posee actualización continua.

El simulador que describimos puede ejecutarse tanto en un ambiente windows como sobre DOS. Si bien tiene una interface amigable, tiene la desventaja de ser lento.

El ambiente de simulación para DOS no posee el ambiente de debugging de comportamiento descripto anteriormente. [DEB93]

Generación de código

El compilador C acepta código fuente C y produce código fuente assembler TMS320C3x. El compilador es un paquete compuesto por un shell, un optimizador y un listador. El shell permite compilar, ensamblar y linkear los módulos fuentes. El listador permite ver el código assembler generado por cada sentencia C original. El optimizador usa fases de optimización sofisticadas que emplean varias técnicas avanzadas para generar código más eficiente y compacto a partir del fuente C. Las optimizaciones generales pueden ser aplicadas a cualquier código C, existiendo además las optimizaciones específicas de la arquitectura del dsp TMS320C3x. [COM95] Estas optimizaciones específicas de la arquitectura lo que hacen es detectar, en el código fuente original, las secciones que pueden ser ejecutadas en paralelo, y traducir esas secciones a las instrucciones paralelas provistas en el conjunto de instrucciones del procesador (paralelismo interno).

El assembler traduce archivos fuente en lenguaje assembler a archivos objeto en lenguaje de máquina COFF.

El archiver permite extraer módulos para modificar las librerías existentes.

El library build utility permite construir nuevas librerías.

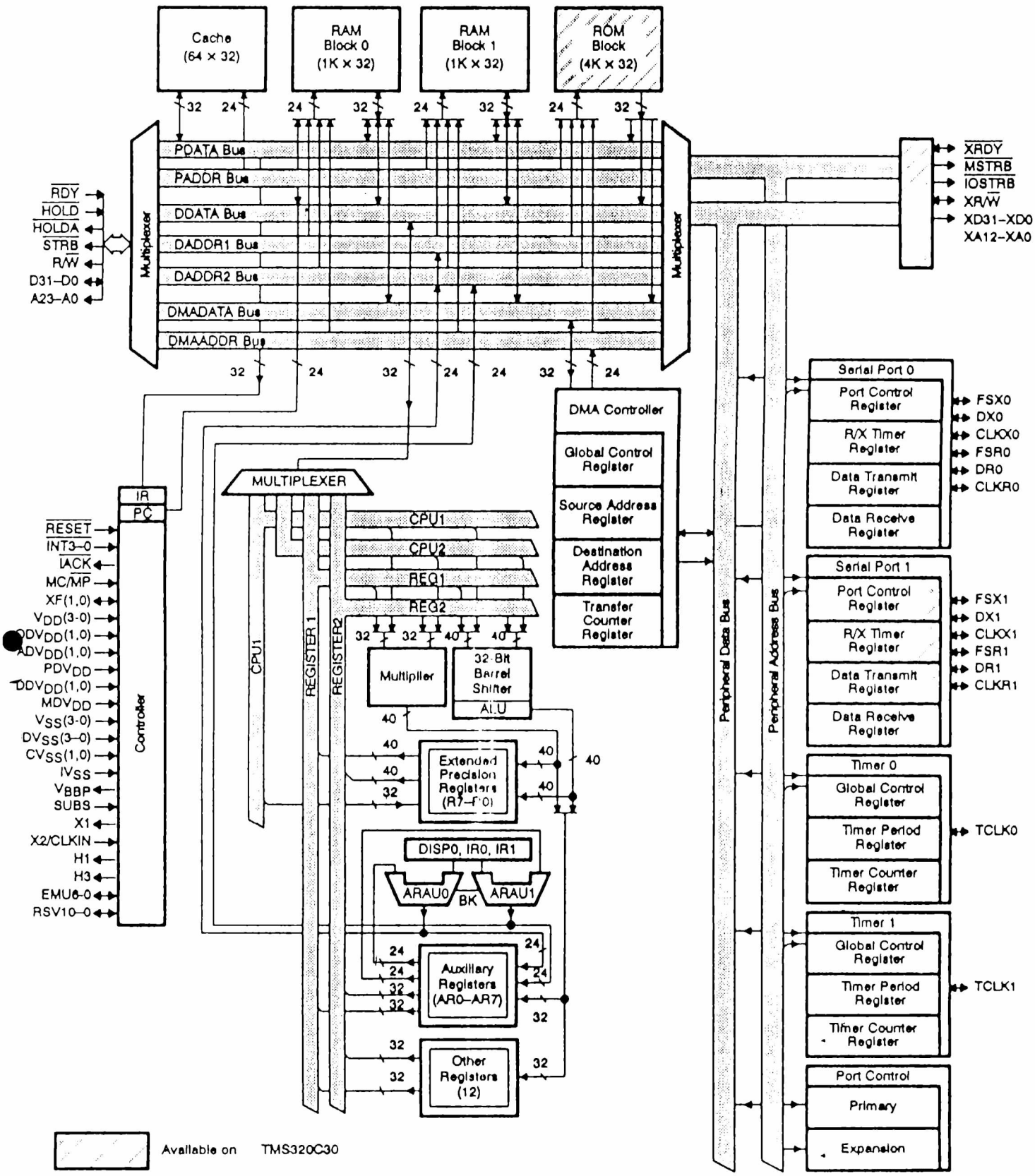
El linker combina archivos objeto en un módulo objeto ejecutable.

Implementación y prueba del algoritmo reconocedor:

La ejecución de nuestro algoritmo reconocedor en el simulador del dsp no fue tan simple como migrar el código del lenguaje Borland C al lenguaje C del dsp TMS320C3x, dadas las restricciones que presentan el compilador C y el ambiente de simulación. Las restricciones a las que hacemos referencia fueron la carencia por parte del lenguaje C del dsp de rutinas que permitan realizar el manejo de entrada y salida (necesarias para poder procesar las imágenes), dado que no se cuenta con un sistema operativo del dsp que soporte estas operaciones. Para solucionar esto, el simulador nos provee una manera que es definir ciertas direcciones de memoria como puertas de entrada y salida conectadas a archivos, y luego implementar en assembler las funciones de lectura y escritura que las utilizan. Además, el formato requerido para la información es que la misma esté en un archivo, donde cada línea del mismo sea una palabra en formato hexadecimal para lo cual se tuvo que realizar un algoritmo que transformara las imágenes en formato bmp a dicho formato. Para poder ejecutar el programa sobre el simulador hubo estudiar el manejo del mapa de memoria del mismo. Este se mapea a través de un archivo en el cual se especifica el mapa de memoria a través de comandos que permiten definir los bloques, la longitud, la dirección y el tipo de memoria. En cuanto al tamaño de la memoria, el mismo está limitado sólo por la capacidad del disco rígido de la PC donde se está ejecutando el simulador.

Una vez implementadas las funciones para manejar los archivos y ampliada la memoria de tal forma que permita cargar la imagen, el algoritmo fue pasado con la misma lógica con la que se desarrolló para la ejecución secuencial.

Figure 2–1. TMS320C3x Block Diagram



8-Extensión del ambiente de simulación del dsp TMS320C3x:

Introducción:

En este punto se plantea como objetivo realizar la extensión del simulador con el propósito de simular una arquitectura multi-DSP.

Dado que el simulador descripto anteriormente no permite simular una arquitectura multi-DSP, lo que se propone es tener varias sesiones del simulador ejecutándose simultáneamente en distintas PCs con algún mecanismo que les permita a dichas sesiones comunicarse entre sí a fin de que puedan sincronizarse así como también intercambiar información.

Interface del simulador con el exterior:

La única manera en la cual el simulador permite comunicarse con el exterior para realizar la entrada/salida de datos es a través de archivos. Los mismos, para poder permitir una comunicación elemental, deberían poder compartirse entre las diferentes sesiones que se están ejecutando; pero el simulador abre los archivos que utiliza para escribir en forma exclusiva, con lo cual hay que descartar esta forma comunicación elemental, ya que otra sesión no podría leer dichos archivos.

Se plantea entonces utilizar algunos archivos como canales de comunicación, implementando un programa residente que intercepte la escritura y la lectura sobre estos archivos, realizando en realidad una comunicación e intercambio de datos entre las sesiones que los utilizan como canales.

Surge también la necesidad de implementar una librería en el lenguaje C del dsp TMS320C3x, la cual provee las funciones de comunicación a nivel de programa

entre los procesos (sesiones del simulador) que necesitan comunicarse utilizando los canales.

Comunicación entre diferentes sesiones de simulación:

Como se dijo antes, el objetivo es implementar un programa residente que intercepte la escritura y la lectura que realiza el simulador en determinados archivos (que en realidad son utilizados como canales), y que realice la comunicación real entre las distintas sesiones de simulación.

Soporte de comunicaciones:

NetBios:

El programa residente, implementado en C, utiliza como soporte de comunicación las rutinas provistas por NetBIOS, ajustándose al paradigma de pasaje de mensajes sincrónico.

NetBIOS (sistema Básico de Entrada y Salida de Redes de IBM) es una interface de comunicación "peer to peer" que provee una conexión virtual entre computadoras en una red. Es un software de internace para aplicaciones que requieran intercambio de información en una red de computadoras. No existe el concepto de Server o la relación Amo/Esclavo. Netbios brinda una serie de servicios que permiten a las máquinas conectarse y comunicarse sobre una red LAN. No es necesario un Server, ni archivos compartidos ni espacio de disco. La función principal de Netbios es establecer una conexión virtual entre computadores en una red y pasar datos a través de dicha conexión.

NetBIOS es la más ampliamente implementada y emulada interface entre

programas de aplicación LANs en el mundo de las PCs.

Realizando llamadas a NetBIOS se puede transmitir instantáneamente información a una aplicación en otra máquina de la red.

Las llamadas a NetBIOS son independientes del hardware; los mismos comandos que trabajaron originalmente con la red PC LAN de IBM trabajan sin cambios en Ethernet, Token Ring, etc.

Los comandos de NetBIOS son, además, independientes del software, dado que se pueden ejecutar sobre Netware de Novell, eComs's+, OS/2, etc.

Hay dos formas de transferir datos utilizando NetBios:

- transferencia de datos confiables provista por el soporte se sesión. Si hay pérdida de datos NetBios retorna un código de error. Se pueden enviar hasta 64K de datos en una sesión, y hay comprobación de errores.
- soporte de datagrama. Se utiliza para enviar mensajes cortos (de hasta 512 bytes) y no hay comprobación de errores.

La mayoría de las órdenes de NetBios se pueden ejecutar en modo espera o en modo no espera.

- Modo espera: el programa que llama espera a que se ejecute la orden antes de continuar con la próxima instrucción.
- Modo no espera: el programa sigue con la próxima instrucción antes de finalizar con la orden. Se usan para obtener un rendimiento máximo y cuando se desea que NetBios ejecute varias órdenes, una después de otra.

El programa residente que utiliza NetBios como soporte de comunicación utiliza la transferencia de datos a través de sesiones, debido a que es un modo de comunicación seguro. Además, los comandos de NetBios son utilizados en modo espera, ya que de esta manera se ajusta al modelo de pasaje de mensajes sincrónico.

El programa residente:

Se debe tener un programa residente por cada sesión de simulación.

El simulador trabaja con los archivos a través de los servicios que brinda la interrupción 21. Las funciones que utiliza son las siguientes:

- ***INT 21, 3C:*** crear un archivo utilizando un handler
- ***INT 21, 3D:*** abrir un archivo utilizando un handler
- ***INT 21, 3E:*** cerrar un archivo utilizando un handler
- ***INT 21, EF:*** leer un archivo o dispositivo utilizando un handler
- ***INT 21, 40:*** escribir un archivo o dispositivo utilizando un handler

Para poder interceptar los llamados que el simulador realiza a estas funciones, lo primero que realiza el programa residente es cambiar la rutina de atención de la interrupción 21, de forma tal que si el simulador llama a estas funciones para alguno de los archivos que van a utilizarse como canales, no se ejecute la rutina de atención original sino la implementada por nosotros (es decir, la que realiza la comunicación entre las diferentes sesiones); en otro caso (llamado a estas funciones con archivos que no son canales), se llama a la rutina de atención original. También se deben definir los nombres de los archivos que serán utilizados como canales, de forma tal que el programa pueda identificarlos. Los "canales" son unidireccionales, por lo tanto en los archivos correspondientes se efectuará un sólo tipo de operación: lectura (canal de entrada) o escritura (canal de salida).

Además, se debe establecer la sesión de comunicación entre los programas residentes de las sesiones de simulación que desean comunicarse (utilizando las funciones *call* y *listen* de netbios). La sesión permanece activa hasta que la

aplicación finalice su ejecución y cierre los archivos definidos como canales.

Luego se implementan las nuevas rutinas de atención de las funciones de la INT21 citadas antes. A saber:

- ***apertura:*** cuando el simulador llama a la función de apertura para un archivo definido como un canal, lo que se hace es obtener y almacenar el handler que el sistema operativo le asignó a dicho archivo al realizar su apertura (la apertura se realiza a través de la rutina original).
- ***creación:*** ídem apertura
- ***lectura:*** cuando el simulador llama a esta función para un archivo definido como canal de entrada (utilizando el handler del archivo), el programa residente espera recibir datos de otra sesión cuyo canal de salida esté asociado con este canal de entrada. La comunicación con la otra sesión se realiza utilizando la función *receive* de NetBios, y los datos recibidos son colocados en el buffer de lectura correspondiente, el cual será leído por el simulador.
- ***escritura:*** cuando el simulador llama a esta función para un archivo definido como canal de salida, el programa residente envía los datos que el simulador intenta escribir en el archivo (utilizando el handler del archivo) a la sesión que tenga asociado su canal de entrada con este canal de salida. La comunicación con la otra sesión se realiza utilizando la función *send* de NetBios; los datos a enviar se obtienen del buffer de escritura correspondiente, en el cual son colocados cuando el simulador intenta escribir utilizando esta operación.
- ***cierre:*** el programa residente finaliza la sesión de comunicación.

Dado que el simulador realiza la lectura y escritura a través de buffers de longitud fija, se debió implementar una librería (en el lenguaje C del dsp tms320c3x) que provee las funciones de comunicación a nivel de programa entre

los procesos (sesiones del simulador) que necesitan comunicarse utilizando los canales. Es decir, si un proceso quiere enviar o recibir un solo dato, se debe rellenar el espacio restante del buffer con información inválida, de forma tal que al escribir en el archivo definido como canal el dato que se intenta enviar, el envío sea realizado en ese instante y no se deba esperar a que se complete el buffer. Una situación análoga se da con la lectura.

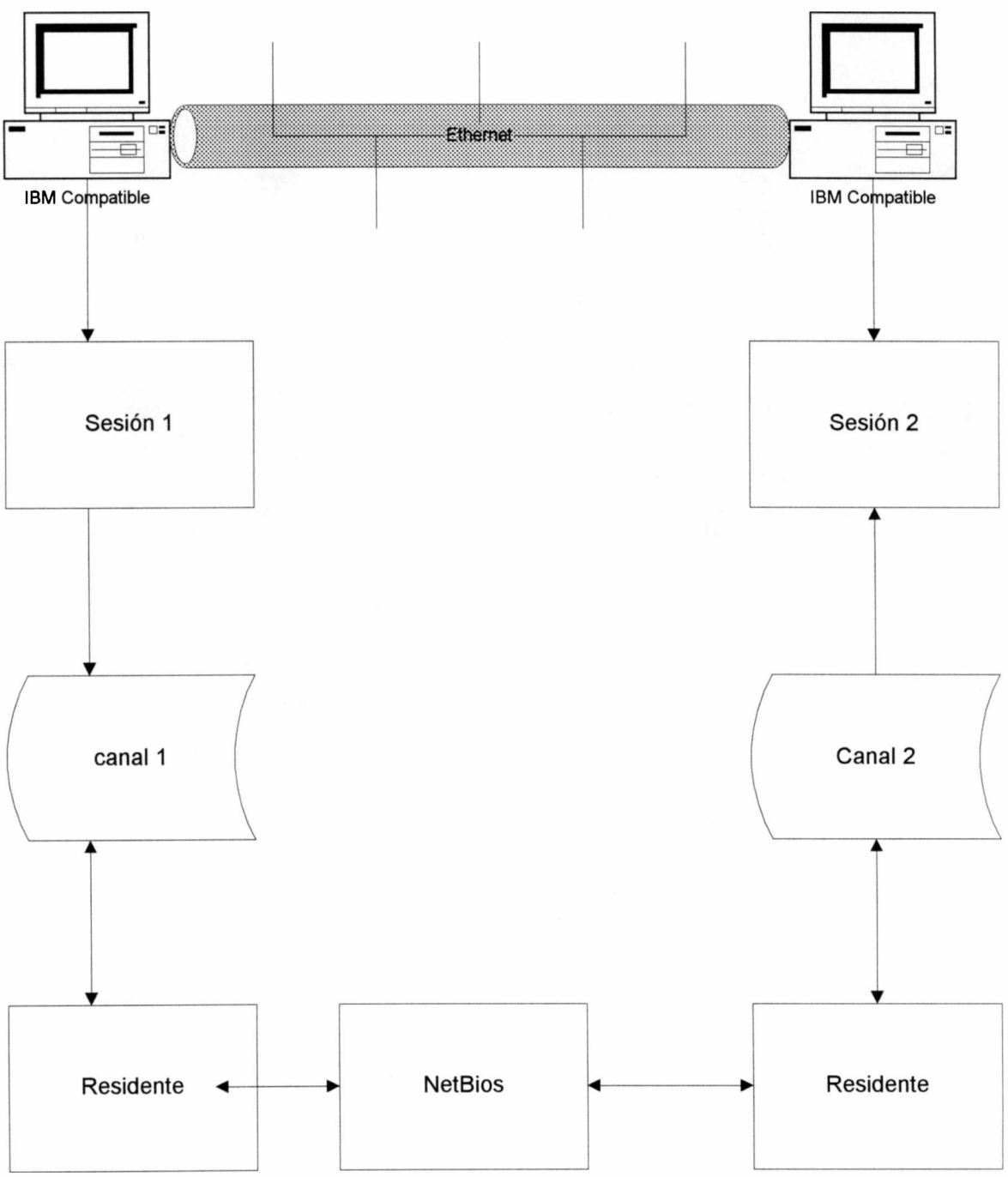
Además se debió implementar un protocolo de comunicación en el programa residente, de forma tal que al recibir el buffer a ser transmitido, descarte la información inválida y envíe sólo los datos útiles y la información de control que le permita al receptor determinar cuándo finaliza el envío. Asimismo la función del programa residente que realiza la recepción de los datos, debe poder establecer cuando el envío ha finalizado en base a esa información de control, y en ese momento debe devolver los datos y el control al programa que está efectuando la lectura.

En el lenguaje a nivel de proceso, podemos definir entonces las siguientes operaciones basadas en el paradigma de pasaje de mensajes sincrónico:

- enviar(cant, buffer): se traduce como la escritura de los datos que están en buffer en un archivo definido como un canal de salida. Para que esta escritura se realice en forma inmediata, aquí se efectúa el agregado de información inválida a fin de completar el tamaño de buffer requerido por el simulador para efectuar la escritura, además de agregar la información de control correspondiente (para que el programa residente envíe sólo los datos válidos). Al realizarse esta escritura, los datos son interceptados por el programa residente por medio de la función de escritura descripta antes.

- int recibir(buffer): se traduce como la lectura de los datos del archivo definido como un canal de entrada. Esta operación llama a la función lectura descripta antes, la cual se queda bloqueada hasta que se establezca la comunicación con la sesión que debe enviar los datos. Una vez que el programa residente los recibe y retorna el control al simulador, esta operación lee los datos hasta que encuentra la información de control que indica que el mensaje ha terminado y coloca esos datos en buffer.

Extensión del simulador a un multi-DSP



9-Paralelización de algoritmos:

Un aspecto saliente de las aplicaciones de tratamiento de señales (en particular imágenes) es el procesamiento masivo de datos. A fin de minimizar el tiempo de respuesta de los algoritmos se tiende a explotar la concurrencia inherente a los mismos. [LAW92] [AND91]

Desde el punto de vista de software, la transformación de algoritmos secuenciales en paralelos significa identificar las tareas que se pueden ejecutar simultáneamente así como los recursos compartidos y las relaciones de interdependencia (dependencia de datos, dependencia de control y dependencia de recursos). A partir de esta identificación se puede especificar la concurrencia en un lenguaje adecuado. [OLS83] [MAY88]

Desde el punto de vista del hardware, la explotación de la concurrencia inherente a un algoritmo dado requiere de paralelismo, es decir capacidad de multiprocesamiento en un instante dado de tiempo. En las aplicaciones de tratamiento de señales dos modelos de arquitecturas diferentes tienen la mayor difusión:

- Multiprocesadores basados en arquitecturas convencionales RISC como los transputers [INM90], que se conectan con diferentes topologías como redes o hipercubos y que requieren lenguajes orientados como el OCCAM. [MAY88]
- Multiprocesadores basados en arquitecturas CISC con un set de instrucciones dedicadas y redundancia de hardware a fin de optimizar tiempos como los DSPs.

En este trabajo se analiza una clase de algoritmos de reconocimiento de patrones simples en el plano, y la paralelización de los mismos sobre las

arquitecturas mencionadas precedentemente.

Paralelización del algoritmo de reconocimiento:

La clase de algoritmo presentada plantea interesantes posibilidades de paralelización:

Dada una figura **F**, se cuenta con un proceso **Master** que la particiona por filas en tantas partes como unidades de procesamiento se dispongan.

En cada una de estas unidades reside un proceso **E** que extrae el borde y calcula la superficie de la porción de la imagen que le fue asignada.

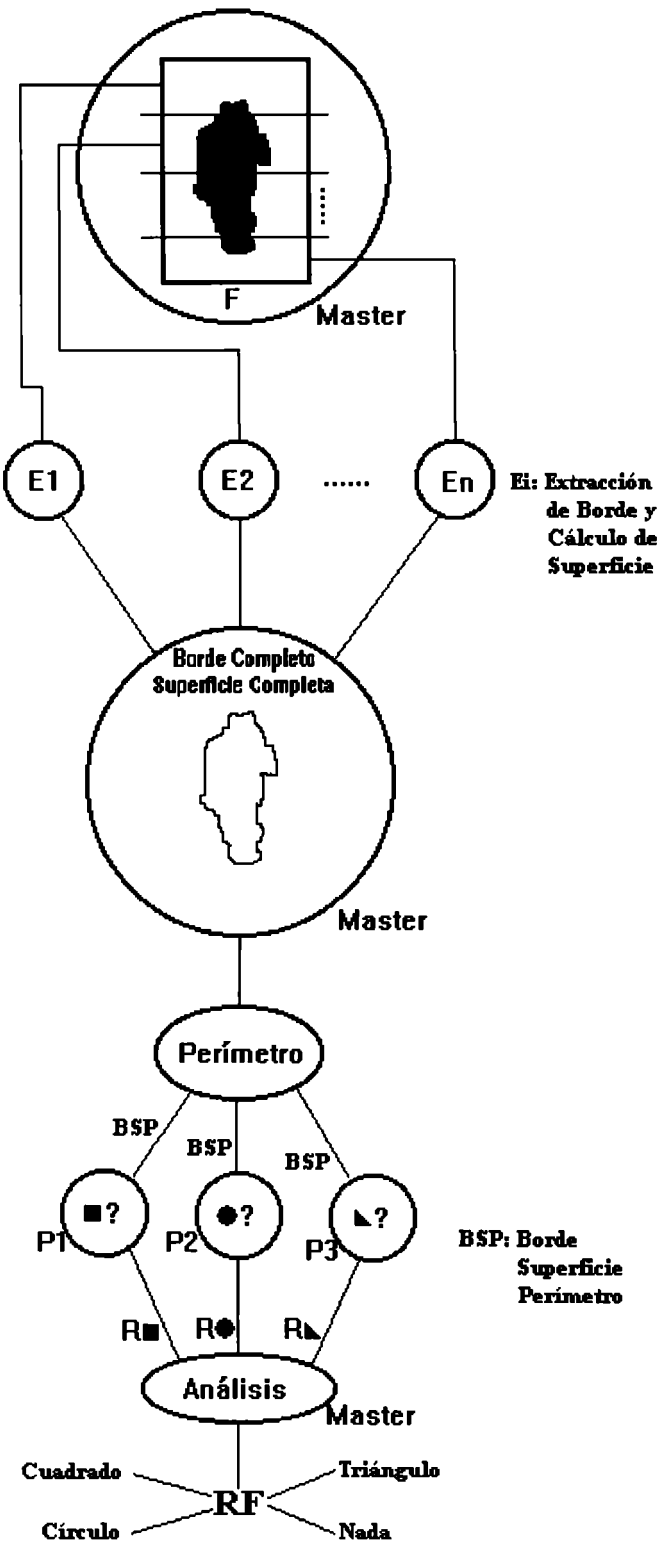
A continuación, los datos extraídos son devueltos a un proceso **Master** donde se arma el borde tomando los subbordes recibidos; y se calcula la superficie total. Además, se determina la longitud del perímetro.

Ahora, estos datos (borde, superficie y perímetro) son transmitidos a tres procesos (en lo posible, residentes en tres procesadores distintos).

Cada uno de los procesos está encargado de reconocer una de las clasificaciones de las figuras:

* **P1**: Analiza si es círculo.
* **P2**: Analiza si es cuadrado.
* **P3**: Analiza si es triángulo.

Estos tres procesos devuelven el resultado de éxito o no en el reconocimiento al proceso **Master**, el cual da el resultado final.


F
Master
E1
E2
.....
En
Ei: Extracción de Borde y Cálculo de Superficie
Borde Completo
Superficie Completa
Master
Perímetro
BSP
BSP
BSP
P1
P2
P3
BSP: Borde Superficie Perímetro
Análisis
Master
Cuadrado
Triángulo
RF
Círculo
Nada

Evaluación y eficiencia en el paralelismo:

Una forma de medir la eficiencia en la performance de cualquier algoritmo paralelo, es utilizando el *speed-up*. El speed-up surge del cociente entre el tiempo que tarda el algoritmo al ejecutarse en un único procesador (algoritmo secuencial) y el tiempo que tarda el algoritmo en ejecutarse sobre n procesadores. En una situación ideal, se esperaría que el speed-up fuera n; en tal caso el aprovechamiento del paralelismo sería máximo. Pero, esta situación no se da en la realidad, ya que generalmente hay partes del algoritmo que son secuenciales por naturaleza y además hay que tener en cuenta el tiempo que se gasta en la comunicación de los procesadores. Así, surgen dos factores a tener en cuenta en el momento de la evaluación:

- *latencia de sincronización:* es el tiempo requerido por dos procesadores para sincronizarse.
- *latencia de comunicación:* es el tiempo que requieren los diversos procesadores para comunicarse entre sí. Es un parámetro importante para minimizar. Está determinada por los algoritmos usados así como también por la arquitectura que sirve de soporte para la comunicación.

10-Resultados Obtenidos

Prueba del algoritmo:

Se probó el funcionamiento del sistema utilizando un conjunto de imágenes capturadas por medio de un *scanner* y generadas manualmente utilizando un graficador, retocándolas de modo tal que cumplieran con las hipótesis de que el borde no tuviera ramificaciones y que el cuerpo de la figura no tuviera huecos. Luego de ajustar apropiadamente las cotas del error admisibles para cada atributo de cada figura, se consiguió que el sistema reconociera cada una de las imágenes presentadas de la misma manera que lo hace la percepción humana.

El algoritmo secuencial:

El algoritmo secuencial fue implementado en lenguaje C y ejecutado en una PC 486 con coprocesador matemático.

El tiempo que tarda el sistema en reconocer una figura que se encuentra en una imagen de 10000 pixels es de aproximadamente 1 segundo.

Luego, el algoritmo fue implementado en lenguaje OCCAM y probado en un transputer. En promedio, el tiempo requerido para procesar las imágenes fue de 1200 milisegundos.

Posteriormente se realizó la migración del algoritmo secuencial de reconocimiento de figuras geométricas al compilador C del dsp TMS320C3x, para realizar luego las pruebas en el simulador, evaluando en este caso el paralelismo interno que brinda el dsp, el cual es explotado por el compilador C del mismo.

Los resultados obtenidos sobre un conjunto de imágenes procesadas dieron un

promedio de 80 milisegundos; demostrando ser un DSP, en promedio, 12 veces más veloz que un transputer.

El algoritmo paralelo:

Para poder probar la performance de las arquitecturas paralelas se tendría que disponer físicamente de las mismas. Además, para poder probar la escalabilidad de las diferentes arquitecturas tendríamos que contar con un alto número de arquitecturas idénticas (procesadores). Esto está limitado por el costo económico de las mismas.

En el LIDI se probó el algoritmo paralelo sobre una red con 2 placas transputer interconectadas sobre un bus de PC. Los resultados obtenidos utilizando el mismo conjunto de imágenes con el que se probó el algoritmo secuencial fueron, en promedio, de 770 milisegundos, resultando un speed-up de 1,55.

Posteriormente se realizó la extensión del ambiente de simulación del dsp para que pudiera simular un multi-dsp. Una vez hecha esto, se probó el algoritmo paralelo sobre el simulador extendido con dos sesiones de simulación. Los resultados obtenidos, en promedio, fueron de 45 milisegundos, resultando un speed-up de 1,77.

No se probó el algoritmo reconocedor con más de dos sesiones de simulación debido al impacto negativo en la performance que esta prueba ocasionaría como consecuencia del incremento considerable en los tiempos de sincronización entre los procesos que intentan comunicarse. Considerando que NetBios provee una interface de comunicación "peer to peer", si se tienen más de dos sesiones de simulación que necesitan comunicarse entre sí, se hace necesario establecer la

sesión de comunicación cada vez que un proceso quiere comunicarse con procesos diferentes, y finalizar la sesión corriente antes de establecer una nueva. Estas operaciones de establecer la sesión y finalizarla insumen un tiempo de ejecución considerable, más aún teniendo en cuenta que se realizan cada vez que la aplicación efectúa una operación para comunicarse con un proceso que no sea con el que se comunicó la última vez. Esto no ocurre cuando se tienen dos sesiones de simulación, ya que un proceso se comunica siempre con el mismo, y por lo tanto la sesión de comunicación se establece una sola vez al comienzo de la aplicación, y se mantiene establecida hasta que la aplicación termina su ejecución.

Análisis Conceptual Transputers vs. DSP

El procesador dsp está orientado a aplicaciones específicas del procesamiento de señales, particularmente imágenes, mientras que los transputers son procesadores de propósito general.

El diseño interno de un dsp provee un mayor nivel de paralelismo que el de un transputer.

Gran parte de la alta performance del dsp se debe a los buses y paralelismo internos. Los buses separados de programas, datos y DMA permiten realizar en paralelo búsquedas de programas, accesos a datos y accesos a DMA. Estos buses conectan todo el espacio físico del dsp: memoria on-chip, memoria off-chip y periféricos off-chip. El paralelismo que tiene un transputer está dado por un scheduling que permite intercalar la entrada/salida de un proceso con la ejecución de otro.

En conclusión, con la arquitectura dsp conseguimos mayor performance para realizar procesamiento de señales.

En mediciones de tiempo, el transputer es en promedio 5 veces más veloz que una PC 486 DX2 con 8 mega de RAM; y un dsp es 12 veces más veloz que un transputer. Esto prueba que el dsp si bien tiene un mayor costo, es un procesador eficiente para el desarrollo de algoritmos específicos que requieran un tiempo de respuesta limitado. Ver anexo.

11-Conclusiones

El trabajo desarrollado cumple satisfactoriamente con el objetivo propuesto de desarrollar un algoritmo paralelo que reconozca círculos, triángulos y cuadrados en imágenes digitales. Si bien se trabaja con imágenes sencillas en el sentido de que ellas sólo tienen una figura, el proyecto puede extenderse a un sistema que analice imágenes digitales conteniendo muchas figuras e identificando cada una de ellas. Asimismo, se podría contar con una etapa de preprocesamiento que adecúe las figuras a las hipótesis mencionadas.

Por otro lado, se podrían caracterizar nuevas figuras y de esta manera se generalizaría más el reconocedor; de todos modos dejando siempre un rango limitado de figuras a reconocer debido a las características utilizadas para ello.

Se ha analizado la evaluación del "speed-up" al paralelizar el algoritmo reconocedor, utilizando DSP tipo TMS320C3x, comparando la solución equivalente sobre transputers.

Se han explicado las características del ambiente de simulación DSP disponible en el LIDI, mostrando los resultados de evaluación de perfomance obtenidos para esta aplicación. Además, se realizó la extensión de este ambiente y se la evaluó utilizando el algoritmo reconocedor paralelo.

Si bien se pudo realizar un procesamiento distribuído con múltiples sesiones del simulador ejecutándose concurrentemente, el tiempo requerido por las sesiones para comunicarse e intercambiar datos está condicionado por la performance que brinda Netbios como soporte de comunicación sobre una red Lan, con lo cual la performance del algoritmo paralelo es seguramente menor a la que se obtendría si dicho algoritmo se ejecutara sobre una máquina multi-dsp real, en la cual el tiempo requerido para la comunicación sería probablemente mucho menor. Además, se

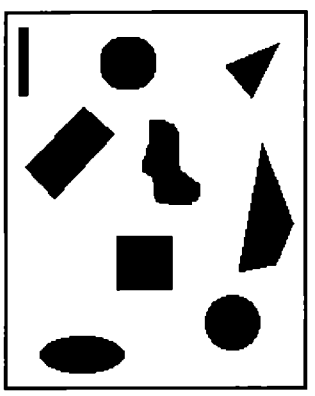
podrían aprovechar las ventajas del paralelismo interno del dsp pudiendo, por ejemplo, conectar un dsp con otros dos y realizar operaciones de entrada/salida en paralelo entre estos procesadores, obteniendo en tal caso mejores resultados.

Además, la extensión del ambiente de simulación está limitada por el soporte de comunicación utilizado (NetBios), y si bien se obtuvieron resultados aceptables en las pruebas utilizando dos sesiones de simulación, la posibilidad de simular más de dos sesiones y obtener resultados igualmente aceptables es baja, debido al incremento del tiempo de sincronización entre los procesos (ver capítulo *"Resultados Obtenidos"*).

De todos modos queda la impresión de que la optimización de algoritmos sobre arreglos de procesadores DSP se vuelve difícil de generalizar a partir de los compiladores y de las herramientas disponibles para generar código existentes hoy, cuando el número de los procesadores crece. Desarrollar aplicaciones paralelas para una máquina multi-dsp que brinden un alto grado de escalabilidad, se torna una tarea bastante compleja y requiere un esfuerzo considerable.

12-Extensión del Proyecto:

A partir de la implementación del algoritmo reconocedor paralelo y teniendo en cuenta que el mismo resuelve el problema de clasificación de una figura dentro de una imagen, se podría en un futuro realizar una extensión del sistema considerando la clasificación de más de una figura.



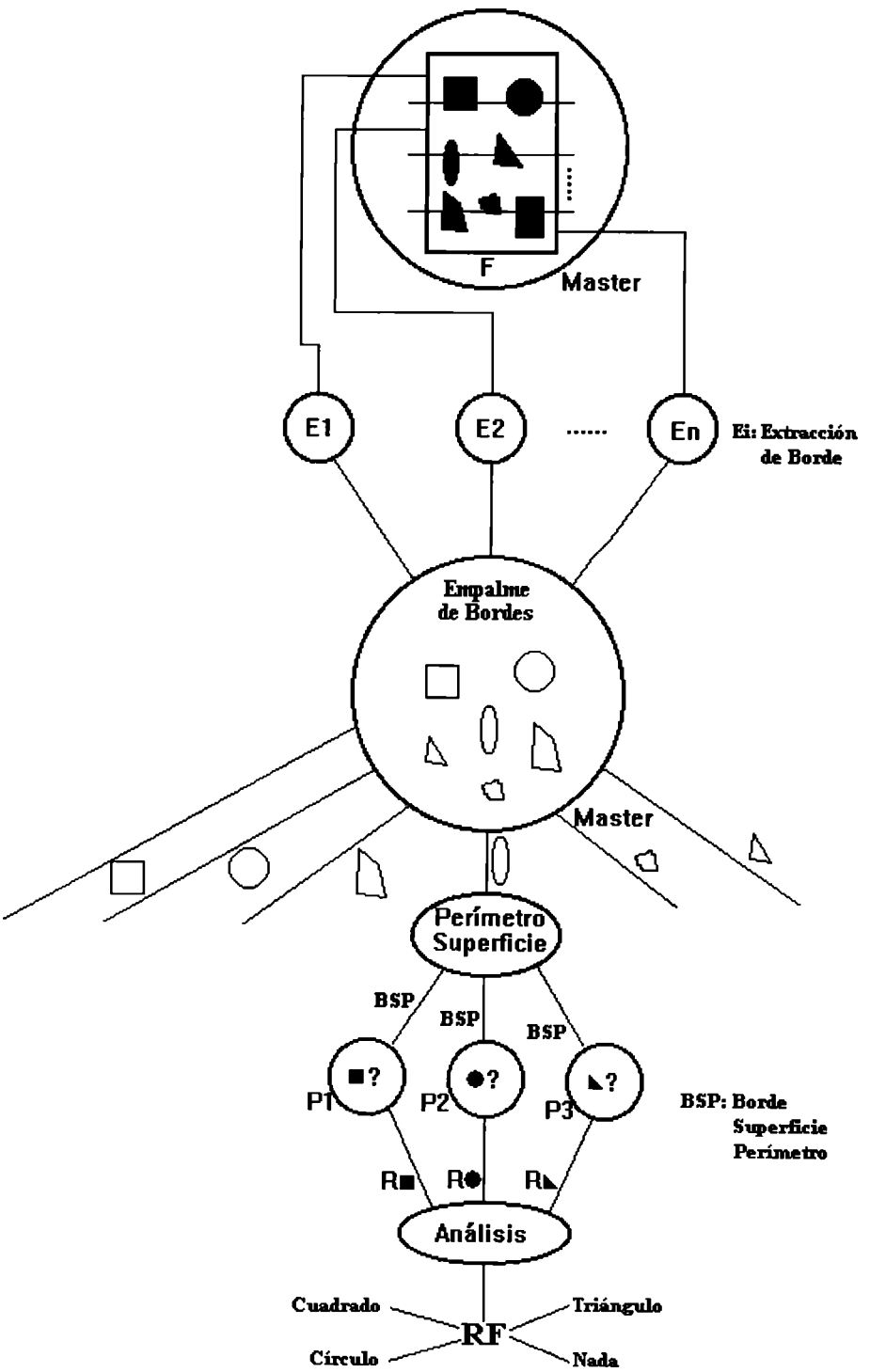
Para resolver esta extensión, la idea es llevar a cabo un preprocesamiento sobre la imagen de entrada donde se aísle cada figura (segmentación) y una vez aislada, sean procesadas separadamente siguiendo el algoritmo reconocedor explicado antes.

Descripción del Preprocesamiento:

La manera de aislar las figuras consiste en dividir la imagen de entrada por filas en tantas partes como procesadores se tengan para que en cada uno de ellos se extraigan las porciones de borde de las figuras que allí se encuentren.

A continuación, cada proceso **Esclavo** retorna al **Master** el conjunto de bordes que halló. El **Master** entonces, con todos los subbordes hallados en la imagen, realiza el empalme de los bordes consiguiendo, de esta manera, obtener el marco de cada figura.

Ahora, los bordes son distribuidos a distintos procesos que se encargarán de realizar la clasificación como se explicó anteriormente, con la salvedad de que el cálculo de la superficie se lleva a cabo luego de la extracción del borde.



La eficiencia de esta paralelización depende fundamentalmente del hardware

disponible. Para el proceso simple de análisis de una figura, resulta aceptable utilizar **3** procesadores (o más); a partir de la extensión a varias figuras, es conveniente contar con **3 * n** procesadores, siendo **n** el número de figuras en la imagen, para mantener el mismo grado de paralelismo.

13-Anexo:

Comparación de tiempos de ejecución en una muestra de 16 imágenes transputer vs Dsp.

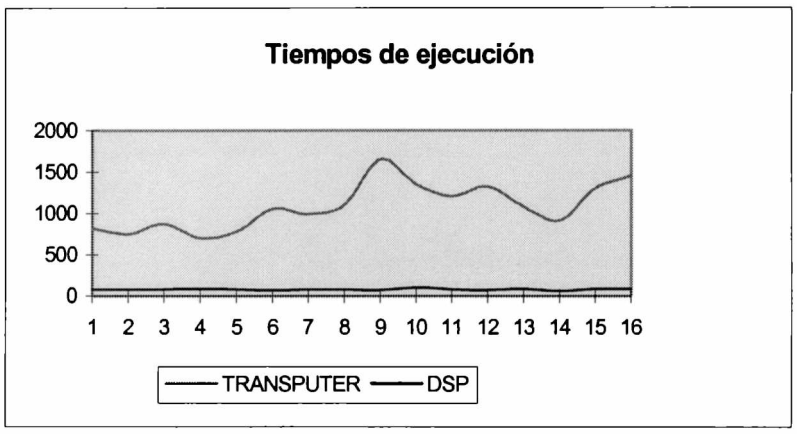
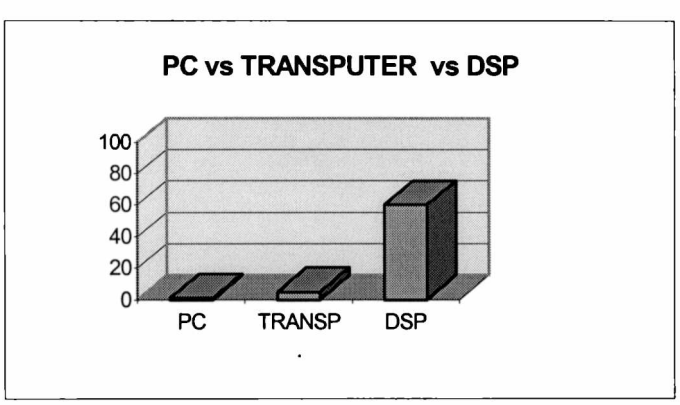
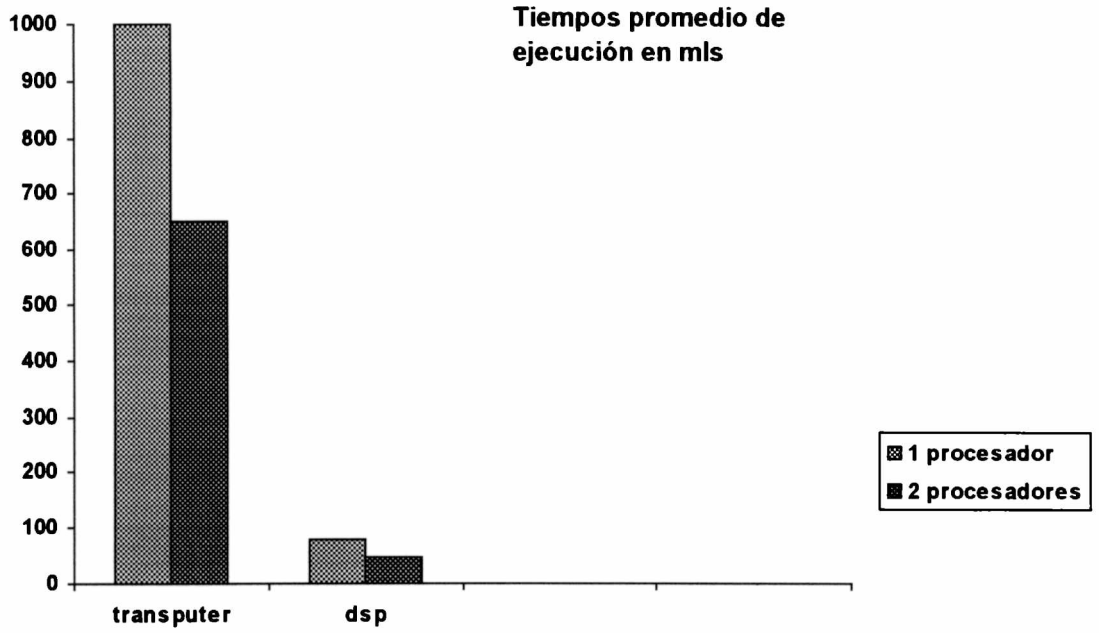
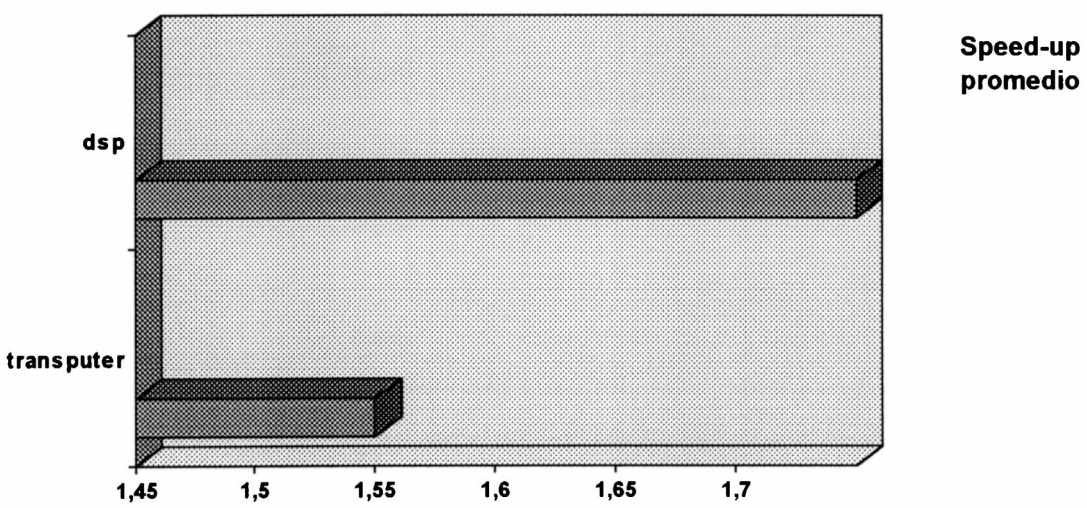


IMAGEN	TRANSPUTER	DSP
circ1	820	77
circ2	750	83
circ3	875	77
circ4	700	86
cuad1	776	78
cuad2	1056	68
cuad3	996	80
cuad4	1100	78
tria1	1650	69
tria2	1350	100
tria3	1210	81
tria4	1325	75
nada1	1080	91
nada2	910	67
nada3	1300	89
nada4	1450	90
promedio	**1084,25**	**80,5625**
speedup	**13,45849496**	

Resultados generales




Tiempos promedio de ejecución en mls
1000
900
800
700
600
500
400
300
200
100
0
transputer
dsp
1 procesador
2 procesadores


Speed-up promedio
dsp
transputer
1,45
1,5
1,55
1,6
1,65
1,7

13-Bibliografía:

[ACO95] - "Algoritmo paralelizable para el reconocimiento de patrones de figuras geométricas"

A. Cosentino - L. Acosta Burllaile - M. De Andrea - A. De Giusti

Second International Congress of Information Engineering 1995.

[AND91] - "Concurrent Programming" - Gregory R. Andrews

The Benjamin/Cummings Publishing Company Inc. 1991

[BAX94] - "Digital image processing"

Gregory A. Baxes

John Wiley & Sons, Inc. 1994

[COM95] - "TMS320C3x Floating Point Dsp Optimizing C Compiler."

Texas Instruments 1995.

[COS94] - "Algoritmo paralelo para el afinado de una imagen digital"

Alejandra Cosentino - Marcelo De Andrea

First International Congress of Information Engineering 1994

[DEA94] - "Algoritmo paralelo para el reconocimiento de curvas en planimetría"

Alejandra Cosentino - Marcelo De Andrea

First International Congress of Information Engineering 1994

[DEA96] - "Evaluación de algoritmos de procesamiento paralelo sobre DSP y

multi-DSP"

A. Cosentino - L. Acosta Burllaile - M. De Andrea - A. De Giusti

II Congreso Argentino de Ciencias de La Computación 1996 .

[DEB93] - "TMS320C3x C Source Debugger."

Texas Instruments 1993.

[FOS95] - "Designing and building parallel programs"

Ian Foster

Internet, 1995

[GON92] - "Digital Image Processing" - Rafael C. González. - Richard E. Woods

Addison - Wesley Publishing Company, Inc. 1992

[HWA93] - "Advanced Computer Architecture: Parallelism, Scalability, Programmability"

Kai Hwang

Mc Graw-Hill 1993.

[INM90] - "Transputer Technical Specifications" - Inmos

CSA - Computer System Architects. 1990

[JAI89] - "Fundamentals of Digital Image Processing" - Anil K. Jain

Prentice-Hall. 1989

[KON94] - "The Khoros Software Development Environment for Image and

Signal Processing" - K. Konstatinides - J. Rasure

IIE Trans. on Image Processing, Vol. 3, No. 3, pp. 243-252, 1994.

[LAW92] - "Parallel Processing in Industrial Real-Time Applications"

Harold W. Lawson

Prentice Hall. 1992

[MAY88] - "Occam2 language definition" - David May

Inmos. 1988

[NIE90] - "Ada in Distributed Real-Time Systems"

Kjell Nielsen.

Mc Graw-Hill 1990.

[RAM96] - "Analysis and Extension of a Simulation Tool for Multi-Dsp parallel processing"

F. Tinetti - H. Ramón - C. Russo - A. De Giusti.

II Congreso Argentino de Ciencias de La Computación 1996 .

[TMS94] - "TMS320C3x User's Guide."

Texas Instruments 1994.