CAPITULO 9

ARQUITECTURAS NUEVAS

9.0 - Breve introducción

Como habíamos anticipado en el final del capítulo 6 existen una serie de arquitecturas difíciles de acomodar en una clasificación genérica de arquitecturas paralelas, ellas eran, a saber :

- Arquitecturas Dataflow
- Arquitecturas híbridas MIMD/SIMD
- Arquitecturas de Reducción
- Arquitecturas de Wavefront Array

En este capítulo veremos en más detalle tales arquitecturas poniendo especial énfasis en las máquinas Dataflow debido a la filosofía en que se basan.

9.1 - Introducción a DATAFLOW

Uno de los objetivos primordiales en el desarrollo de los sistemas es alcanzar la más alta velocidad y performance posibles. Este objetivo ha sido y sigue siendo alcanzado por dos medios :

- explotando las posibilidades tecnológicas de los componentes del computador, y
- adecuando estructuras y organizaciones del computador.

El intento de producir computadoras de arquitectura paralela con alta velocidad y performance obedece a las necesidades de resolver grandes problemas (la mayoría de características paralelas) tales como reconocimiento de patrones (pattern recognition), procesamiento de señales e imágenes (signal & image processing), problemas de inteligencia artificial, física nuclear, predicción meteorológica, control de tráfico aéreo, procesos de control de producción, etc. ; muchos de los cuales deben resolverse en tiempo real.

Computadoras tales como la ILLIAC IV y computadoras asociativas como la STARAN tienen una muy alta performance del orden de 10^2 MIPS para datos de una estructura adecuada o grandes sistemas de vectores en los cuales la misma operación se produce simultáneamente.

Su eficiencia para resolver problemas en donde la estructura de datos no es tan regular disminuye rápidamente, aún cuando son naturalmente paralelas. Estas computadoras son absolutamente ineficientes para procesar problemas seriales.

Otra desventaja substancial es la gran dependencia de la performance de estas computadoras del método de programación del problema. El programador debe conocer la estructura interna del computador para ser capaz de poder utilizar su paralelismo, y debe él mismo identificar el paralelismo en el problema o utilizar un programa especial para hacer esto último.

Muchos autores han llegado a la conclusión de que el problema central de la utilización del paralelismo reside en el modelo básico para expresar y describir el mismo.

El modelo de cómputo de Von Neumann aparece inapropiado a este respecto debido a ser un modelo serial.

Sorprendentemente todos los lenguajes de programación convencionales y las arquitecturas de las computadoras existentes, tanto SISD, SIMD, o MIMD están basados en este modelo de cómputo.

La naturaleza serial del modelo de Von Neumann y los problemas resultantes de ello son el principal obstáculo en la utilización del paralelismo en las computadoras paralelas que procesan programas escritos en lenguajes convencionales de alto nivel.

Luego, se hace necesario crear primero un modelo de sistema computador el cual permita expresar algoritmos de un paralelismo natural.

Uno de tales modelos es el modelo Dataflow conocido como sistema Data-Driven. Estrechamente relacionado con esto está el trabajo sobre utilización del principio de asignación única para expresar el paralelismo.

9.1.1 - El modelo de cómputo Dataflow

Cuáles son las diferencias entre el modelo de cómputo Dataflow (DF) y el modelo convencional de cómputo Von Neumann (Control flow, CF) ?.

En principio, la diferencia radica en qué es lo decisivo en el proceso de cómputo en cada modelo:

- en el CF, es la secuencia de las instrucciones.
- en el DF, es la disponibilidad de los datos.

En una computadora convencional CF, el programa se almacena en la memoria como una secuencia de instrucciones. El programa se ejecuta extrayendo las sucesivas instrucciones desde la memoria y ejecutándolas en el procesador.

Luego, el curso que sigue el cómputo está dado por la secuencia de las instrucciones del programa (es decir, el flujo de control del programa).

No es posible ejecutar cualquier instrucción hasta que todas las instrucciones previas hayan sido ejecutadas. Si los operandos necesarios están disponibles, pueden existir en el programa instrucciones que podrían ejecutarse ya, pero ellas deben esperar a que les toque su turno en la secuencia.

Este es el obstáculo principal en la utilización de algoritmos de un paralelismo natural.

En las computadoras DF, el curso del cómputo está controlado por el flujo de los datos en el programa. Una instrucción puede ejecutarse solamente cuando están disponibles todos sus operandos.

Estos operandos pueden ser obtenidos como datos de entrada o por ser resultado de instrucciones previas en el programa.

La precedencia de una instrucción respecto de otra está dada aquí exclusivamente por la estructura natural del algoritmo que se está realizando y no depende de la ubicación de las instrucciones en la memoria.

Utilizando este modelo de cómputo es posible llegar a ejecutar tantas instrucciones en paralelo como pueda el computador en cuestión.

Luego de la ejecución de la instrucción el resultado se distribuye a todas las instrucciones subsiguientes que hagan uso de ese resultado parcial como operando.

9.1.2 - **Representación de programas Dataflow**

El diseño del modelo de cómputo DF se explica mejor con un ejemplo concreto.

Una de las mejores representaciones de un programa dataflow es mediante un grafo dirigido el cual puede ser visto como una simplificación del lenguaje DF.

Los nodos en el grafo representan operadores (instrucciones de programa) y los arcos representan caminos de datos unidireccionales a través de los cuales los resultados se transmiten en el programa.

En general, es posible representar la ejecución de una instrucción como una función sobre n inputs con m outputs, como se ve en la Fig. 9.1.

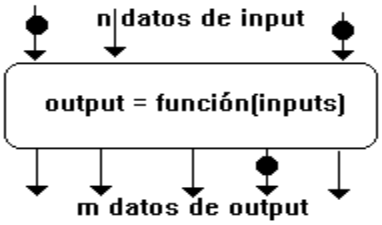


Fig. 9.1. - Una representación simbólica del cómputo DF.

Un punto negro sobre el camino del dato representa la presencia del operando apropiado o del resultado parcial. Una instrucción sólo puede ejecutarse cuando todos sus inputs contienen operandos.

El operador "consume" sus inputs y libera un conjunto de resultados sobre los caminos de output.

9.1.3 - **Tipos de Operadores**

Muchos de los lenguajes dataflow están limitados a dos tipos fundamentales de instrucciones (combinativa y separativa) con una cantidad de inputs y outputs iguales a 1 o 2 (Fig. 9.2).

Las instrucciones combinativas tienen dos inputs y un output. El valor del output es una función de los valores de los inputs. Este operador puede ejecutar funciones sencillas como la suma, OR,

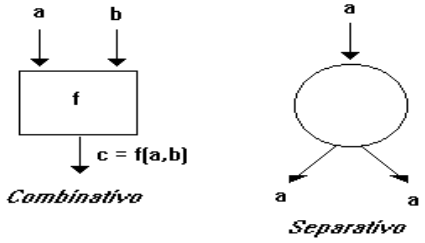


Fig. 9.2. - Los tipos básicos de operadores Dataflow.

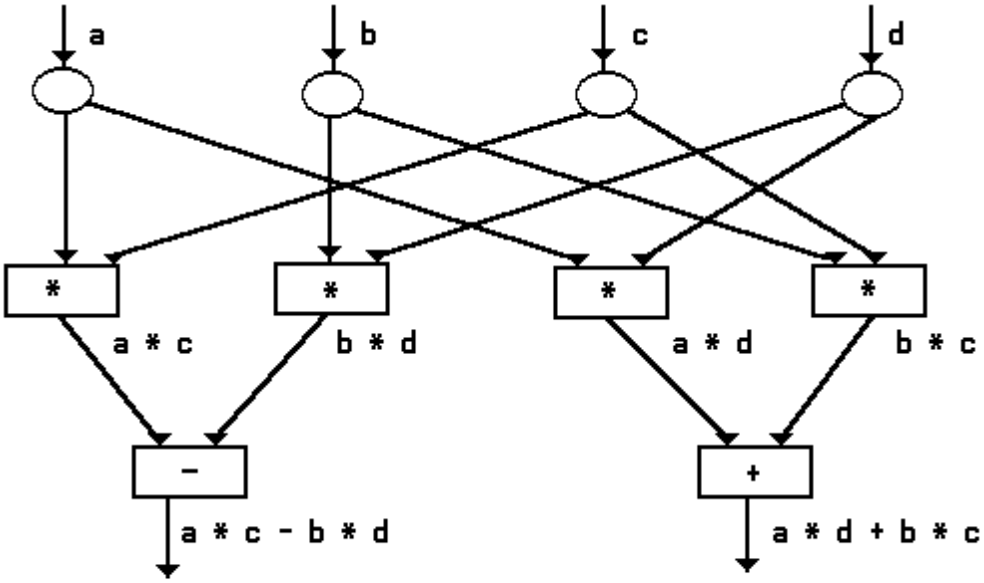


Fig. 9.3. - Un programa sencillo dataflow para multiplicar dos números complejos (a+bi) * (c+di).

AND, etc. o funciones complejas tales como la multiplicación, división, o incluso procedimientos completos.

Las instrucciones separativas producen dos copias del dato de input que pueden ser de tipo diferentes.

Los caminos de los datos representan el flujo de los datos desde una instrucción hacia la próxima. Este es el sistema de comunicación de la ingeniería de implementación. Este sistema de comunicación puede ser un simple bus del sistema, pero puede también almacenar datos en memoria utilizando un sistema FIFO.

Como ejemplo de un cálculo DF se muestra en la Fig. 9.3 la ejecución de un programa sencillo que simula la multiplicación de dos número complejos.

La Fig. 9.4 muestra la ejecución de este programa sobre una computadora DF en cuatro fases diferentes, a saber :

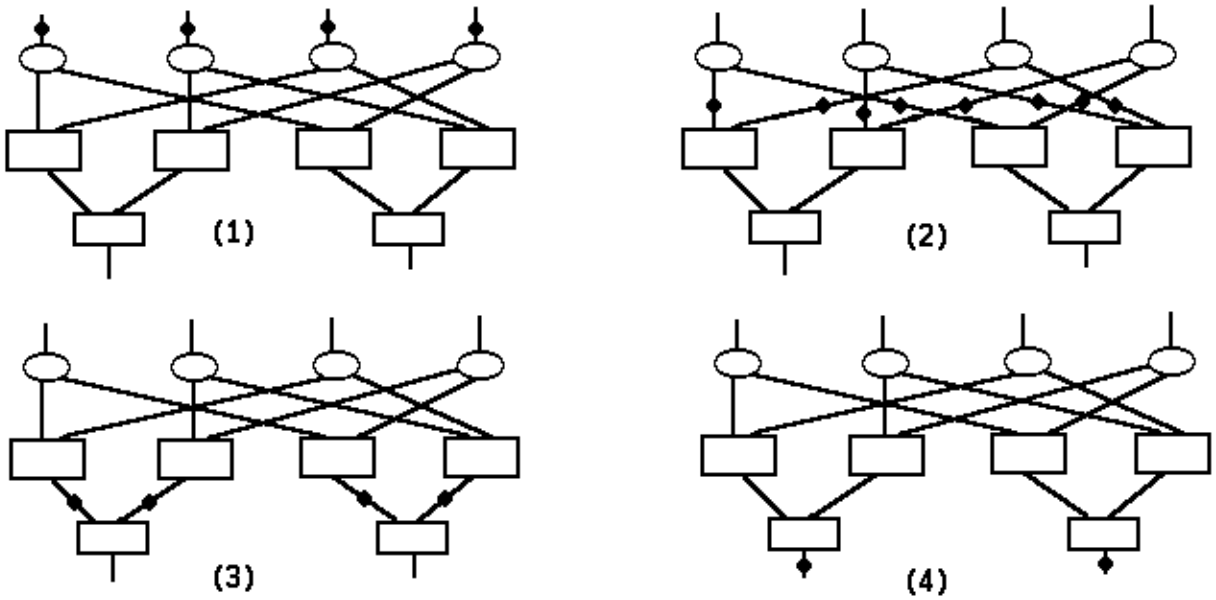


Fig. 9.4. - La ejecución del programa de la multiplicación de dos números complejos.

(1) el estado antes del comienzo del cálculo : todos los operandos están presentes y los inputs de las cuatro funciones están cargados.

(2) estado luego del primer ciclo : se producen copias de los operandos aislados y se distribuyeron a los operadores que los utilizarán.

(3) estado luego del segundo ciclo : los productos parciales ad, bc, ac, y bd se han obtenido y se distribuyen a los siguientes operadores.

(4) estado luego del tercer ciclo : se ejecutaron la suma de ad con bc y la resta de bd a ac ; estos dos outputs forman el resultado complejo.

El estado del cálculo está representado por puntos en los caminos de los datos representado tokens, los que corresponden tanto a variables de input como a resultados parciales. Mediante estos tokens es posible representar la dinámica del cómputo.

9.1.4 - **TOKENS**

Pero la palabra Token es un término aún más amplio :

Token *: es el valor de una variable sobre un camino de dato la cual representa el resultado actual de una operación que ha sido realizada por el operador precedente.*

Un token puede contener dos tipos de datos :

- el dato : uno o varios operandos que pueden ser de diferentes tipos : entero, real, booleano, string, incluso estructuras más complejas como procedimientos completos que pueden dibujarse con grafos de flujos separados ;
- dato de control : el código de operación, direcciones de los resultados, señales, código de programa, cantidad de iteraciones, etc.

El contenido real y el formato de cada token depende del lenguaje del programa DF concreto y de la arquitectura del computador.

Un token es una unidad independiente completa. En principio, normalmente contiene el dato y no direcciones de memoria. No contiene referencias a otros tokens y unidades comunes del computador.

Luego, un token nos dice :

QUE se debe hacer - un token contiene el código de operación.

DONDE debe entregar el resultado - un token contiene la dirección en donde debe entregarse el resultado. Este mecanismo asegura la continuidad del programa.

CUANDO un cálculo debe ejecutarse - un método para sincronización. El cálculo se realiza solamente cuando un token en el input del operador está completo.
CON QUE se ejecuta el cálculo - el token contiene operandos sobre los cuales la instrucción dada se ejecutará.

9.1.5 - **Estados de los operadores**

El operador puede estar en uno de los cuatro siguientes estados (Fig. 9.5), a saber :

- ocioso, el operador tiene por lo menos un token (en el caso de ser un operador combinativo) o ningún token (de ser separativo) en el input y ningún token en el output ;
- habilitado, todos los tokens de inputs requeridos están presentes y está listo para ejecutar la función;
- activo, está ejecutando la función ;
- generando, genera el resultado, un nuevo token, y lo entrega al próximo operador.

En la literatura, se han mencionado hasta ahora dos posibles implementaciones :

- el operador no debe pasar desde el estado de habilitado al estado activo mientras exista un token previo en su output;
- el operador puede pasar del estado habilitado al estado activo prescindiendo de que tenga uno o varios tokens previos en su output. En este caso, pueden estar presentes colas de tokens en los caminos de los datos.

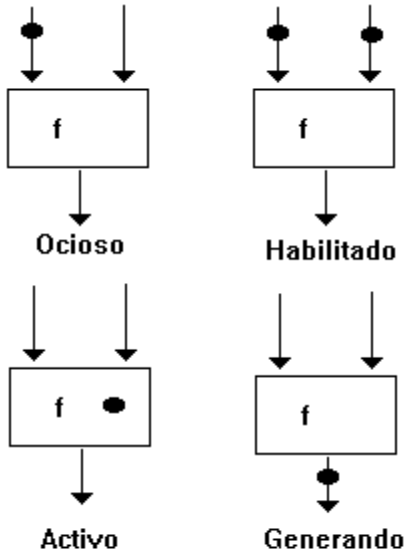


Fig. 9.5. - Estados de un operador Dataflow.

9.1.6 - **Intérpretes Feedback y No-Feedback**

El asunto de la generación de colas en los caminos de los datos es la cuestión clave para los lenguajes de los programas DF, la arquitectura de los computadores DF y la organización de las acciones en las computadoras DF.

En el primer caso, cuando el operador debe esperar hasta que su output haya sido limpiado, es necesario que el procesador pueda representar la información mediante la cual un operador pueda informar a los operadores precedentes el hecho de que ya ejecutó el cálculo actual, es decir, que ya ha consumido el token.

La acción de una computadora de este tipo está basada en un intérprete feedback FI (o de retroceso).

Una acción en la cual el operador puede generar tokens independientemente de si los tokens precedentes han sido ya consumidos o no, está basada en un intérprete no-feedback, NFI.

Esta acción supone la posibilidad de almacenar series de resultados parciales, correspondientes a los tokens, durante la transición de un operador a otro.

Como regla, la memoria trabaja en un principio FIFO.

Existen también extensiones del concepto previo en las cuales una de las series generada de tokens no se almacena en el sistema FIFO, pero puede ser extraída de la cola en un orden arbitrario, por ejemplo, tokens aislados pueden "saltar de la cola", lo que permite una aceleración de la velocidad del cálculo debida a una mejor utilización de los varios niveles de paralelismo.

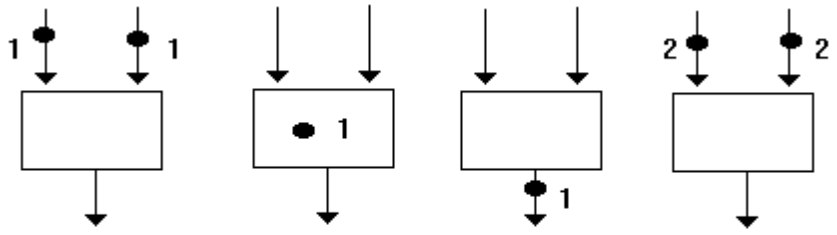


Fig. 9.6. - La ejecución de un cálculo bajo un intérprete feedback.

El camino de los datos puede estar en alguno de los siguientes estados :

- cargado en FI : uno y solo un token están presentes en él;
- cargado en NFI : uno o más tokens

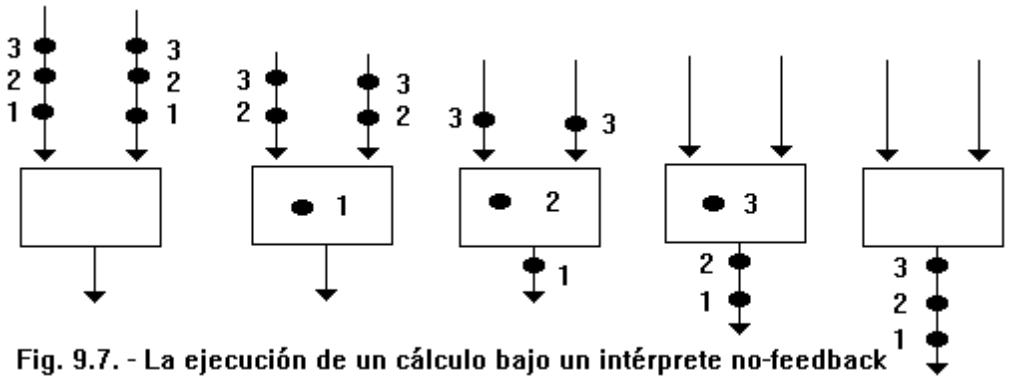


Fig. 9.7. - La ejecución de un cálculo bajo un intérprete no-feedback con encolamiento FIFO de tokens.

están presentes en él.

- descargado - ningún token está presente en él.

Para una mejor comprensión del comportamiento bajo intérpretes FI y NFI mostramos los siguientes tres ejemplos :

Fig. 9.6 muestra el curso de un cálculo bajo FI.

Fig. 9.7 muestra el curso de un cálculo bajo NFI conjuntamente con encolamiento de tokens FIFO en los caminos de datos.

Fig. 9.8 muestra un cálculo bajo NFI conjuntamente con encolamiento arbitrario de los tokens sobre el camino de datos.

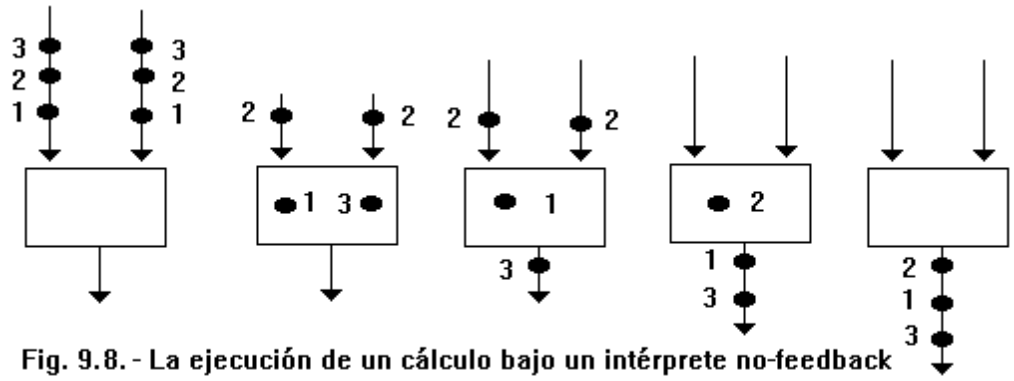


Fig. 9.8. - La ejecución de un cálculo bajo un intérprete no-feedback con encolamiento arbitrario de tokens.

Nótese en la Fig. 9.8 que en el segundo ciclo ocurre un procesamiento simultáneo de dos tokens.

Esto no significa que en el computador ellos hayan sido procesados por el mismo procesador simultáneamente, sino más bien que fueron procesados por diferentes procesadores al mismo tiempo.

El token 2 pudo ser "superado" por el token 3 por varias razones. Por el momento, el procesador puede estar especializado en el procesamiento de estructuras de datos particulares y el dato en el token 3 puede procesarse más rápidamente, o el volumen de los datos en los tokens pueden ser distintos.

Recalcamos que la estructura de un token puede ser bastante compleja.

9.1.7 - **ARQUITECTURAS ESTATICAS Y DINAMICAS**

Las arquitecturas Dataflow pueden dividirse en Estáticas y Dinámicas

9.1.7.1 - **Arquitecturas Estáticas**

En este tipo de arquitecturas los data token se desplazan por los arcos del grafo del programa y, la operación se ejecuta cuando todos los datos (token) están presentes en los arcos de entrada de los operadores.

Sólo se permite que un solo token esté presente en cualquier arco en un instante dado, sino el conjunto de tokens **no** podría distinguirse.

Las implementaciones estáticas cargan todos los nodos del grafo en memoria durante la inicialización y solo permiten que una instancia de un nodo se ejecute por vez.

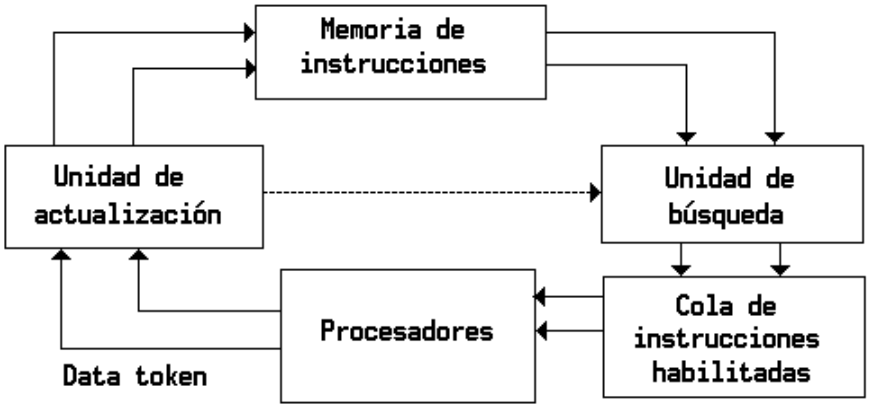


Fig. 9.9. - Arquitectura Dataflow Estática.

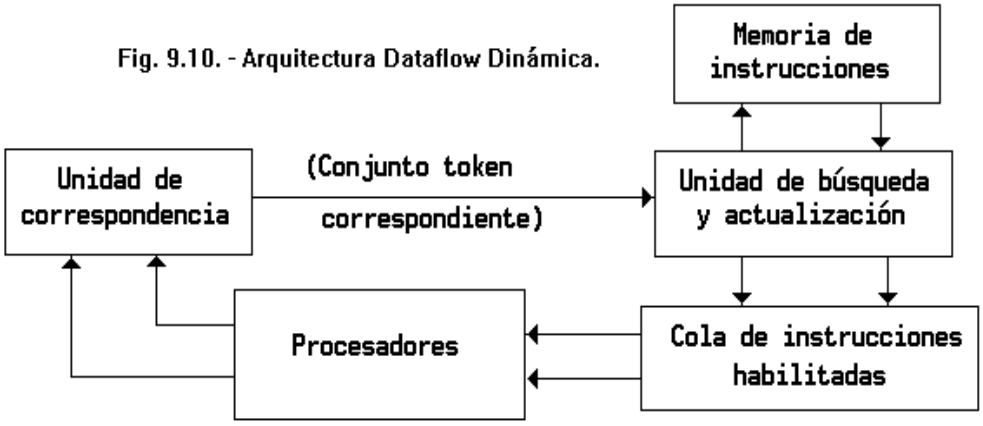
En la Fig. 9.9 podemos visualizar la forma de esta arquitectura.

9.1.7.2 - **Arquitecturas Dinámicas**

En esta arquitectura se utilizan tokens identificados por medio de un rótulo (tags) de modo de permitir que más de un data token pueda estar sobre un arco, permitiendo de esta manera un mayor grado de paralelismo.

Una instrucción se ejecuta cuando tenga todos

Fig. 9.10. - Arquitectura Dataflow Dinámica.



los data tokens presentes de un mismo tag (nivel).

Las implementaciones dinámicas permiten la creación de instancias de nodos en tiempo de ejecución y además que múltiples instancias de un nodo se puedan ejecutar concurrentemente.

Un ejemplo concreto de esta arquitectura es la Máquina de Manchester.

En la Fig. 9.10 esquematizamos esta arquitectura.

9.1.8 - **Principio de asignación única**

El modelo dataflow está estrechamente asociado al principio de asignación única (SA single assignment) lo que nos lleva al diseño de las arquitecturas de tipo DF.

El concepto esencial en el principio SA es la variable y cómo ésta se asigna en el programa.

En las computadoras tradicionales una posición de memoria está asignada a cada variable. Pueden asignarse diferentes valores de la misma variable durante el cálculo, entonces varios valores de la misma variable pueden aparecer en memoria en diferentes momentos. Por lo tanto, cuando una variable es accedida por una instrucción el momento del acceso adquiere significación así como el valor del identificador de la variable.

En la aplicación del principio de asignación única vale la siguiente regla para una variable :

Una variable puede tener un único valor durante el cálculo (y en un cálculo iterativo lo mantiene para un nivel de iteración).

Para el almacenamiento de una variable en el programa existen dos períodos :

- en el primer período se reserva una ubicación para la variable en memoria, pero no contiene ningún valor ; luego su valor no puede ser leído desde la memoria pero sí puede ser escrito,
- en el segundo período la variable ha adquirido su valor y ha sido grabada en la ubicación de memoria reservada; puede ser leída un número arbitrario de veces pero no puede ser modificada nunca más (consiguientemente no se puede realizar ninguna entrada en esa ubicación de memoria).

9.1.9 - **Programación Dataflow**

Los lenguajes de Data Flow pueden tener forma gráfica o léxica. Pueden estar a la altura de lenguajes de alto nivel, pero también pueden estar a la altura de lenguajes máquina.

Estos lenguajes tienen algunas propiedades dignas de atención :

- la escritura del programa es clara y de la misma forma es posible escribir un programa de "alto nivel" así como uno de "bajo nivel" ;
- los programas DF expresan de una manera simple el paralelismo natural de los algoritmos permitiendo la explotación del paralelismo a diferentes niveles ;
- parece posible trasladar en forma sencilla desde lenguajes de alto nivel programas ya hechos a lenguajes DF. Esto permite la utilización de programas existentes y hacer uso de los resultados que fueron obtenidos mediante la programación convencional en la programación DF.

Uno de los problemas más importantes con el que hay que vérselas y que tiene una gran influencia en la utilización del paralelismo, es las estructuras de los datos en los lenguajes de programación DF.

9.1.10 - **Modelos básicos de sistemas Data Flow**

Miller y Cocke (1972-1974) diseñaron dos modelos de sistemas que se han dado en llamar :

- Computador configurable de Modalidad Búsqueda (SM search mode).
- Computador configurable de Modalidad Interconexión (IM interconnection mode).

Estos modelos pueden ser considerados como los modelos básicos de la arquitectura de computadores DF.

Es posible considerar estos modelos básicos como casos extremos y en implementaciones particulares crear sistemas híbridos utilizando los principios de SM e IM en diferentes niveles de jerarquías de un sistema computador.

La mayor propiedad característica de ambos modelos es la posibilidad de la reconfiguración dinámica de la estructura del computador para procesar la tarea tan rápido como sea posible.

La computadora adapta dinámicamente su configuración a la estructura del algoritmo. Esto se logra interconectando (según grafo) los procesadores que se corresponden a operadores en el programa dataflow del problema.

En el tipo IM , la interconexión de los procesadores se implementa a través de un gran circuito (es decir vía hardware).

En el tipo SM la interconexión de los procesadores se simula utilizando un formato de instrucción especial (es decir vía software).

9.1.11 - **La arquitectura Data Flow de Modalidad Búsqueda**

La computadora de modalidad Búsqueda (Fig. 9.11) consiste de una memoria, una unidad funcional y una unidad de control que se ha dado en llamar "un buscador" (searcher).

La unidad funcional está compuesta por un cierto número de procesadores (por ejemplo : una red de microprocesadores).

Un buscador es una unidad especializada en generar tareas para los procesadores que pertenecen a la unidad funcional.

La memoria puede tener diversas estructuras y frecuentemente consiste de un conjunto de diferentes bloques especializados. Existen dos tipos posibles de memorias :

- memoria donde se almacenan datos e instrucciones
- memoria con almacenamiento separado para datos e instrucciones

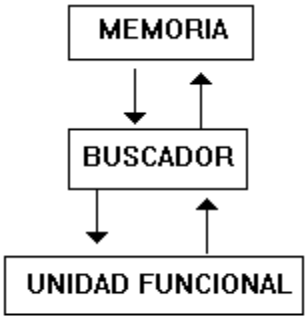


Fig. 9.11. - Computador configurable modalidad búsqueda.

Un procesador que esté libre consulta al buscador para que le dé una tarea. El buscador encuentra una tarea apropiada en memoria o compone una a partir de varios elementos almacenados en diferentes partes de la memoria y se la entrega al procesador seleccionado en la unidad funcional para su ejecución.

Pueden servir como procesadores :

- sumadores,
- multiplicadores,
- testers de errores,
- módulos especializados en macroperaciones,
- procesadores booleanos,
- procesadores de E/S,
- microprocesadores universales, etc.

La performance de un computador dependerá de la cantidad y tipo de procesadores que tenga, de la velocidad de la memoria, pero primordialmente del rendimiento del buscador del cual depende la efectiva utilización de la totalidad de sus procesadores.

El buscador realiza más de la mitad del trabajo que se requiere en un computador tradicional para la ejecución de una instrucción.

Para una descripción más detallada veamos el formato de una instrucción típica, en este caso una instrucción de dos operandos (Fig. 9.12).

Nombre	Operando_1	Operando_2	Dirección_del_resultado

Fig. 9.12. - Formato de instrucción en un dataflow SM.

El nombre y la dirección del resultado constituyen la parte de control de la instrucción, en tanto que los operandos constituyen la parte de datos de la instrucción.

El nombre contiene el código de operación, el nombre del programa y otros datos necesarios para sincronización, información de estado del cálculo, señales, etc.

Los campos de operandos contienen los valores de los operandos, aunque también pueden tener las direcciones de los mismos en memoria, y la dirección del resultado es la dirección donde debe entregarse el resultado luego de realizado el cálculo (podría ser la dirección en memoria, en alguna memoria auxiliar, o la dirección del próximo procesador, etc.).

Es decir, de esta forma, la instrucción tiene todos los datos necesarios para ejecutarse.

La instrucción puede almacenarse de esta forma en la memoria o puede ser generada así en el buscador. Este mecanismo asegura la "interconexión lógica" de los procesadores individuales en el programa y la continuidad del mismo.

Luego de que un procesador ejecutó una instrucción solicitará la siguiente al buscador; por lo tanto la velocidad del procesador depende de la disponibilidad de los datos y del flujo de datos del algoritmo.

Cada instrucción puede procesarse en el momento en que se encuentren disponibles sus operandos, pero si un procesador no está libre en ese momento, la sincronización del cómputo no se pierde. Más aún, las instrucciones pueden ejecutarse en un orden arbitrario. Esta propiedad es muy prometedora cuando se está diseñando una computadora paralela, ya que permite la explotación del paralelismo natural de las tareas así como una buena utilización de la memoria, ya que ambos, tanto las instrucciones como los datos, pueden estar distribuidos arbitrariamente en la memoria.

Cuál es la conexión entre los términos "token" e "instrucción ejecutable" (EI) ? Un token se utiliza principalmente para representar la dinámica del cálculo en el programa DF que se escribe en forma de grafo ; en tanto que una instrucción ejecutable se utiliza al describir la actividad en la computadora DF.

Por ejemplo en la Fig. 9.13 vemos la ejecución de una función f sobre dos operandos a y b como se la representa en el programa, y por otro lado se muestra cómo la ejecución de una función nos lleva a la generación de una EI en el computador.

Nótese que la EI es equivalente al token cuando el token está dentro del operador, es decir, cuando el operador se encuentra en estado activo, y también EI es equivalente al conjunto de tokens a y b cuando estos se encuentran en el input del operador, o sea, cuando este se encuentra en estado habilitado.

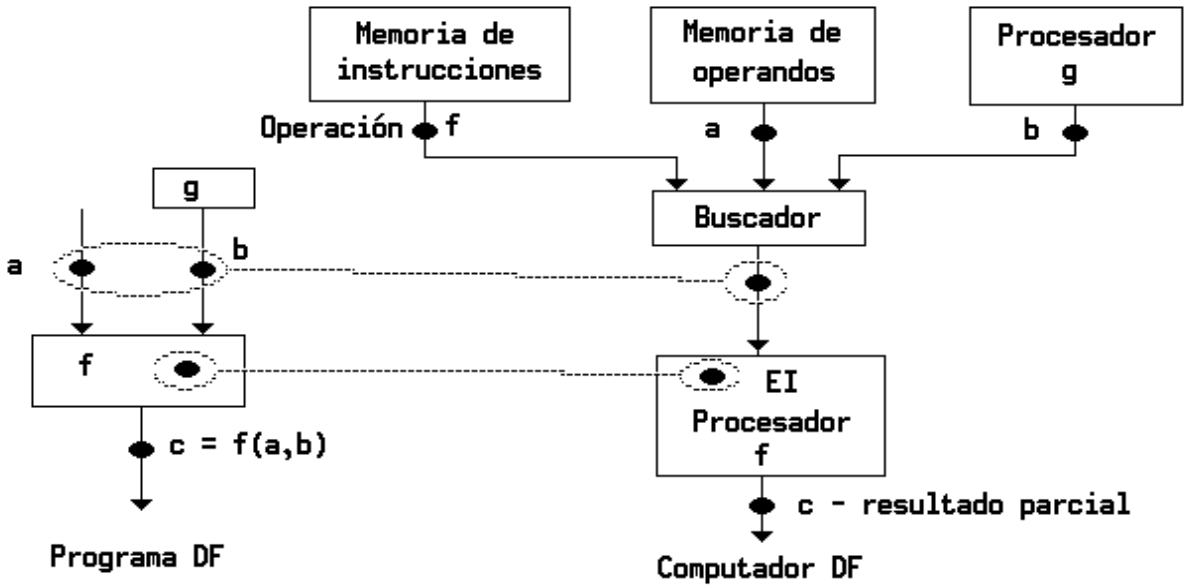


Fig. 9.13. - Token versus Instrucción Ejecutable (EI).

9.1.12 - **La arquitectura Data Flow de Modalidad Interconexión**

Al igual que en las computadoras SM las computadoras IM son también reconfigurables. En las IM la reconfiguración es más rígida.

El elemento característico del computador IM es un circuito o red de interconexión que ejecuta esta función de reconfiguración.

Este circuito provee una conexión directa entre los outputs de un grupo de procesadores y los inputs del próximo grupo de procesadores y crea de esta forma una red de procesadores correspondiente al programa DF que está siendo ejecutado.

Como regla, no es suficiente un conjunto de procesadores para crear una estructura de cómputo que pueda ejecutar el programa entero de una sola vez.

Entonces se hace necesario dividir el programa en bloques de un tamaño apropiado con respecto a la cantidad de procesadores disponibles por vez.

Esto se realiza a través de un compilador que determina la interconexión de los procesadores de manera tal que la estructura de cómputo corresponda al grafo del programa DF o a alguna parte del mismo. Esta interconexión se codifica y se almacena en memoria como una instrucción de seteo del circuito. Esta instrucción es accedida en primer término y llevada al control de seteo del circuito el cual realiza la interconexión de los procesadores. Así la estructura de cómputo está lista para poder comenzar a ejecutar el bloque de programa correspondiente.

Los operandos y los resultados parciales del cómputo de los bloques precedentes se toman de la memoria a través del control de acceso de datos antes de comenzar el cálculo.

Durante la ejecución del bloque no es necesario obtener ninguna instrucción ni datos desde la memoria, lo cual reduce la carga sobre los caminos de datos y sobre la memoria.

Luego de ejecutar un bloque los procesadores involucrados y la parte del circuito pertinente se liberan para poder ser nuevamente seteadas para otro bloque.

De esta manera la computadora IM aparece como un computador especializado mientras se encuentra ejecutando un bloque.

Resulta claro que otro bloque puede ser seteado antes de la finalización de la ejecución del precedente, y que los procesadores individuales pueden liberarse tan pronto como hayan terminado

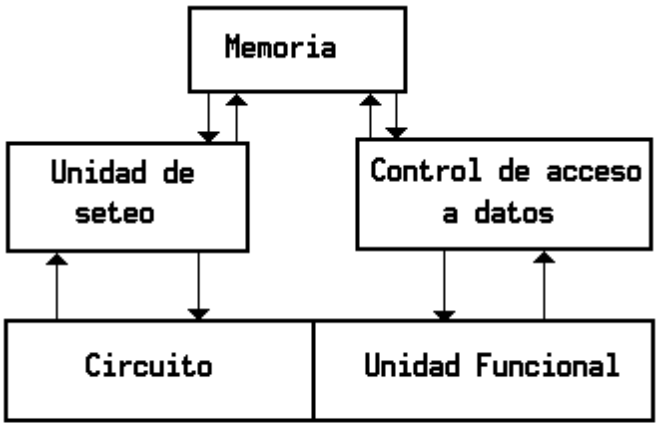


Fig. 9.14. - Computador configurable de modalidad interconexión.

su actividad en el bloque relevante, lo que puede llegar a ocurrir mucho antes de que el cálculo completo del bloque en su totalidad haya finalizado.

En la Fig. 9.14 podemos ver un esquema típico de la estructura de estos computadores.

9.1.13 - **Un mecanismo para implementar computadoras Dataflow**

Algunas arquitecturas almacenan directamente los tokens que contienen los resultados de las instrucciones en una plantilla o modelo (template) de la instrucción que los utilizará como operandos.

Otras arquitecturas utilizan esquemas de apareo de tokens (token matching), en las cuales una unidad de apareo almacena tokens y trata de hacerlos concordar o aparear con las instrucciones.

Cuando el conjunto completo de tokens (es decir todos los operandos) se reúnen para una instrucción, se crea y encola para su ejecución una plantilla que contiene los operandos relevantes.

La Fig. 9.15 muestra como una arquitectura simplificada de apareo de tokens puede procesar el fragmento de programa que se ve en la Fig. 9.16.

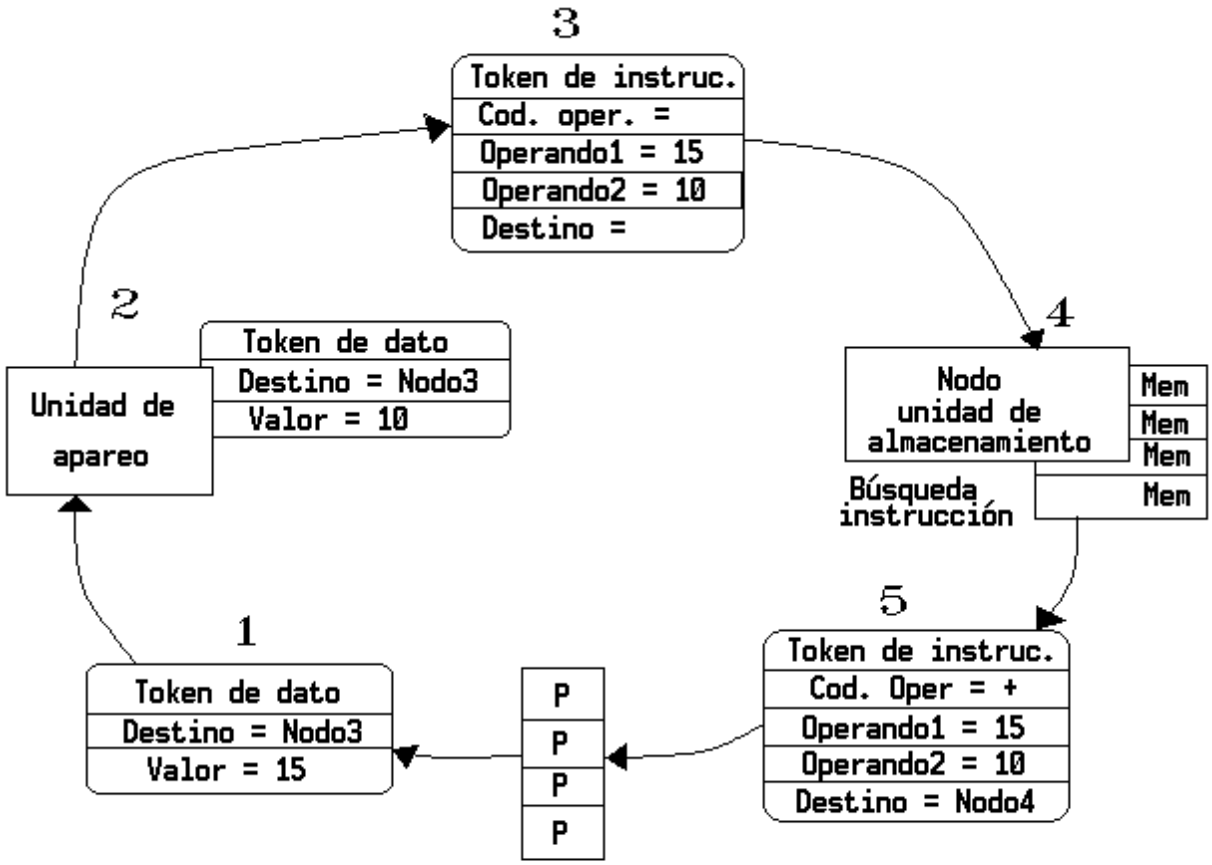


Fig. 9.15. - Ejemplo de apareo de tokens.

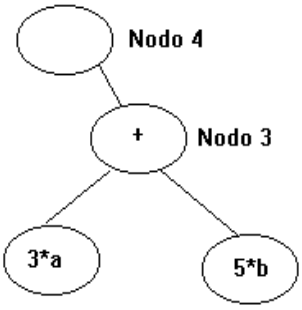


Fig. 9.16. - Fragmento de un grafo de un programa Dataflow.

En el paso 1, la ejecución de (3*a) resulta en la creación de un token que contiene el resultado (15) y una indicación de que la instrucción en el nodo 3 necesita de él como operando.

El paso 2 muestra como la unidad de apareo hace coincidir este token con el otro token que es resultado de realizar (5*b) y con la instrucción del nodo 3.

Esta unidad de apareo crea el token instrucción (la plantilla) que se ve en el paso 3.

En el paso 4 el nodo que hace de unidad de almacenamiento obtiene el código de operación relevante desde la memoria. Llena entonces los campos apropiados del token (paso 5) y asigna la ejecución a un procesador.

La ejecución de la instrucción creará un nuevo token resultado que se usará de input en el nodo 4.

9.2. - **ARQUITECTURAS MIMD/SIMD**

Durante los años 80 se construyeron una cantidad experimental de arquitecturas híbridas que permitían que partes de una arquitectura MIMD se controlaran en una modalidad SIMD (por ejemplo la DADO, la Non-Von, la Pasm y el Computador Array Reconfigurable de Texas o TRAC).

Los mecanismos explorados para reconfigurar arquitecturas y controlar la ejecución SIMD fueron asaz diversos. Usaremos como ejemplo de ayuda para poder ilustrar este concepto un sistema computador de pasaje de mensajes estructurado en forma de árbol (Ver Fig. 9.17).

La relación maestro/esclavo del controlador de la arquitectura SIMD y los procesadores puede mapearse hacia los nodos descendentes del subárbol.

Cuando el procesador raíz de un subárbol opera como un controlador SIMD se encarga de transmitir instrucciones a los nodos descendientes que, a su vez, las ejecutan en sus memorias locales.

La flexibilidad de las arquitecturas MIMD/SIMD las convierte en candidatas muy atractivas para que se continúe la investigación sobre ellas.

De entre los incentivos más recientes para invertir sobre nuevos desarrollos se cuentan el procesamiento paralelo de imágenes y las aplicaciones de sistemas expertos.

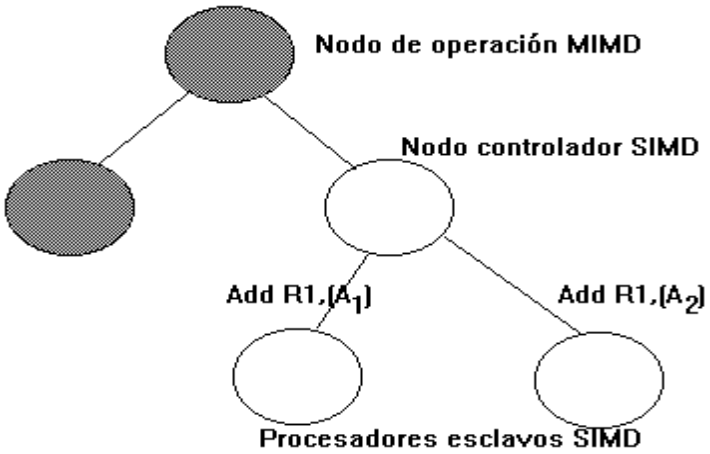


Fig. 9.17. - Operación MIMD/SIMD.

9.3 - **ARQUITECTURAS DE REDUCCION**

Las arquitecturas de Reducción o Demand-Driven (que se comportan según los requerimientos) implementan un paradigma de ejecución en el cual una instrucción está habilitada para realizarse cuando sus resultados son requeridos como operandos de otra instrucción que ya estuviera habilitada para ejecución.

Las investigaciones sobre estos tipos de arquitecturas comenzaron en los últimos años de la década del 70 con la finalidad de buscar nuevos ejemplos de ejecución paralela y para dar soporte a lenguajes de programación funcional.

Las arquitecturas de reducción ejecutan programas que consisten en expresiones anidadas. Las expresiones se definen recursivamente como literales o funciones de aplicación sobre argumentos que pueden ser literales o expresiones.

Los programas pueden "referenciarse" a expresiones por su nombre, las cuales devuelven siempre el mismo valor. Esta propiedad se denomina **propiedad de transparencia referencial**.

Los programas de reducción son aplicaciones de funciones construidos sobre funciones primitivas

La ejecución de un programa de reducción consiste en el reconocimiento de expresiones reducibles, y en reemplazarlas luego por sus valores calculados. Por lo tanto, el programa de reducción completo queda finalmente reducido a su resultado.

Debido a que el método general de ejecución solamente permite que una instrucción se ejecute cuando sus resultados son necesarios para una instrucción previamente habilitada, se requiere lógicamente de alguna regla adicional para habilitar la primera instrucción (o instrucciones) y poder comenzar el cálculo.

Existen objeciones prácticas para implementar las arquitecturas de reducción. Por ejemplo, la sincronización de los requerimientos para los resultados de una instrucción (ya que preservar la transparencia referencial exige que el cálculo de una expresión se realice solo una vez) y el mantenimiento de copias de los resultados de evaluación de las expresiones (ya que un resultado puede ser referenciado más de una vez pero puede ser consumido en las subsiguientes reducciones una vez que la primera se ha disparado).

Las arquitecturas de reducción utilizan tanto reducción de strings como reducción de grafos.

La reducción de strings implica manejar literales y copias de valores, que se representan como strings que pueden expandirse o contraerse dinámicamente.

La reducción de grafos implica manejar literales y referencias (pointers) a valores.

Luego, un programa se representa como un grafo, y un método de "garbage collection" obtiene dinámicamente memoria a medida que la reducción avanza.

Las Fig. 9.18 y 9.19 muestran una versión simplificada de una arquitectura de reducción de grafos que mapea el programa que sigue en una estructura arbórea de procesadores y que pasa tokens a requerimiento y devuelve resultados.

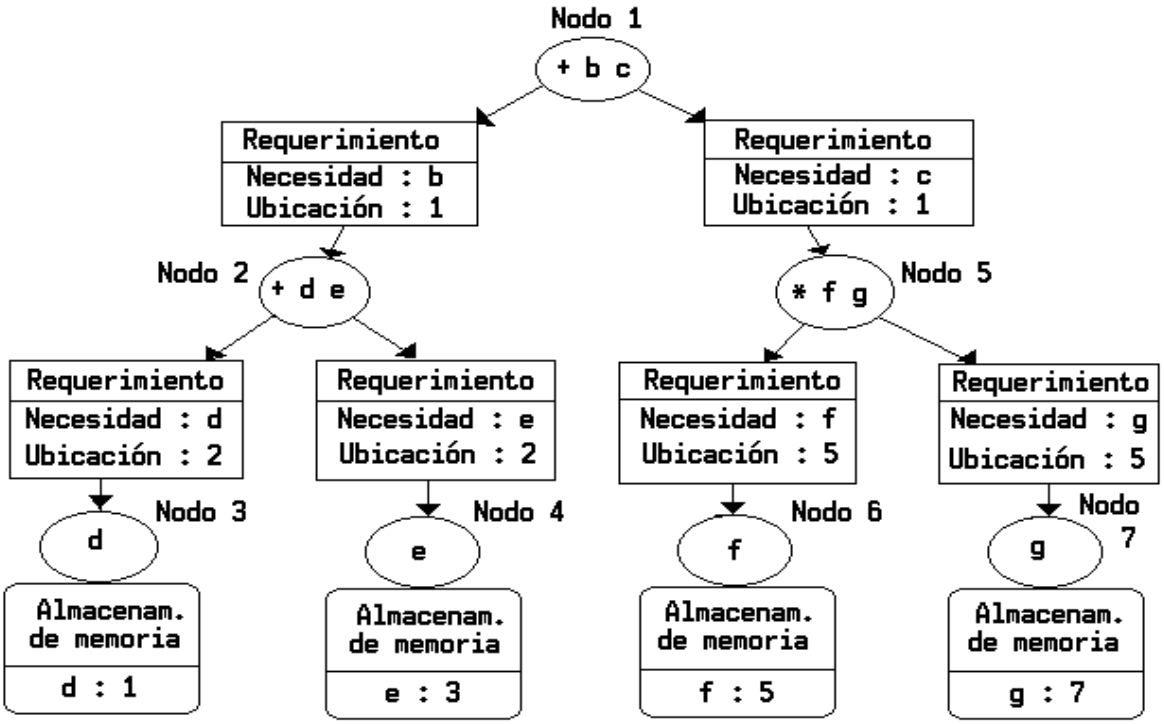


Fig. 9.18. - Arquitectura de Reducción, tokens producidos a requerimiento del programa.

a = + b c ;
b = + d e ;
c = * f g ;
d = 1;
e = 3;
f = 5;
g = 7.

La Fig. 9.18 muestra todos los tokens producidos a requerimiento del programa, así como su propagación hacia abajo en el árbol. En la Fig. 9.19 se muestra como los dos últimos tokens de resultados producidos son pasados al nodo raíz.

Se han propuesto arquitecturas muy disimilares para soportar reducción de string y de grafos, por ejemplo la Máquina de Reducción Newcastle, la Máquina North Carolina Cellular Tree, y el Sistema Utah Multiprocesamiento Aplicativo.

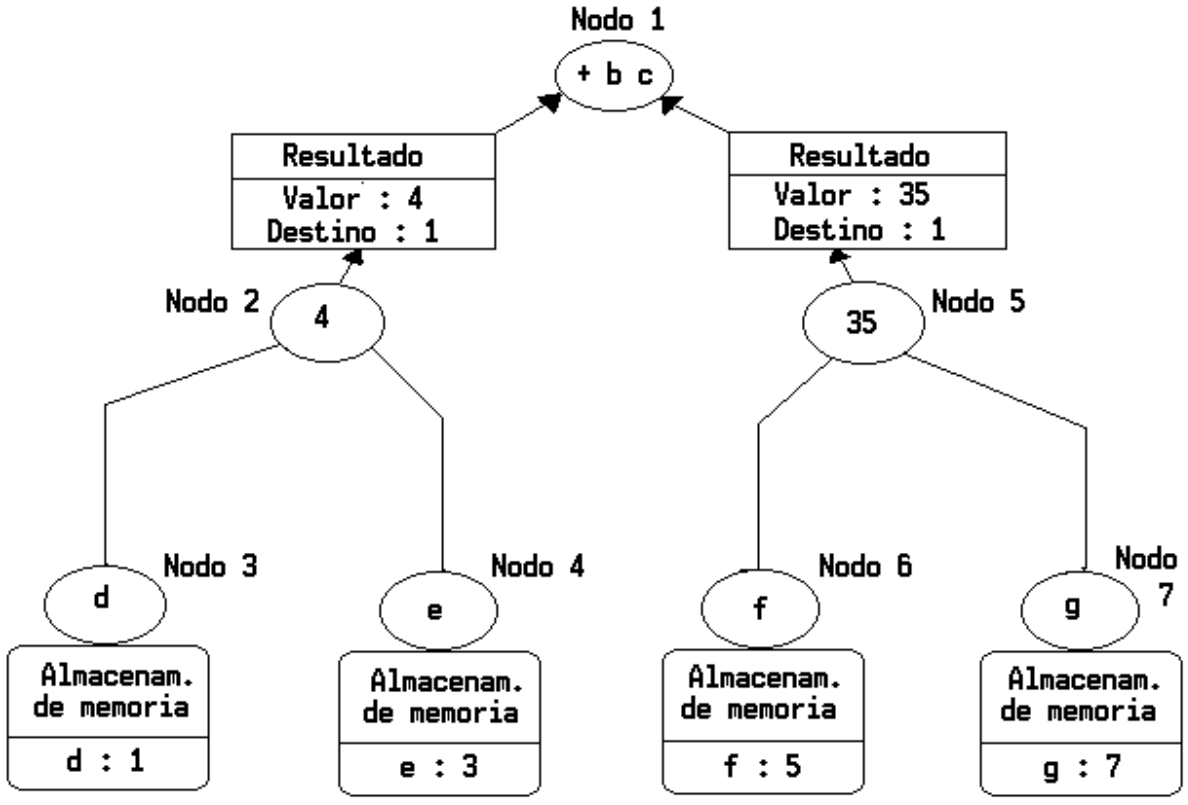


Fig. 9.19. - Arquitectura de Reducción, tokens de resultado.

9.4 - **ARQUITECTURAS WAVEFRONT ARRAY**

Los Wavefront array combinan el pipelining de datos de los sistólicos con el flujo asincrónico de la ejecución de las Dataflow.

S. Y. Kung desarrolló los conceptos de las Wavefront a fines de los años 80 para resolver el mismo tipo de problemas que estimuló el desarrollo de los sistólicos : producir arquitecturas eficientes de un costo aceptable para sistemas de propósito específico que balancearan una gran cantidad de cálculos con un amplio bandwidth de E/S.

Ambos, Wavefront y Sistólicos, se caracterizan por poseer procesadores modulares y regulares, y una red de interconexión local.

Sin embargo los wavefront reemplazan el reloj global y los mecanismos explícitos de retardo para sincronizar el pipelining de los datos por un mecanismo de handshaking (acuerdo) asincrónico para coordinar los movimientos de los datos entre los procesadores.

Luego, cuando un procesador ha realizado su cálculo y está listo para pasar los datos a su sucesor, informa al sucesor y cuando éste le dice que se encuentra listo mediante la señal de acknowledge, le envía los datos.

El mecanismo de handshaking hace que el cálculo en el wavefront se realice fácilmente a través del array sin que ocurran intersecciones (o colisiones) y de esta forma el procesador array se comporta como un medio de propagación de olas (wave).

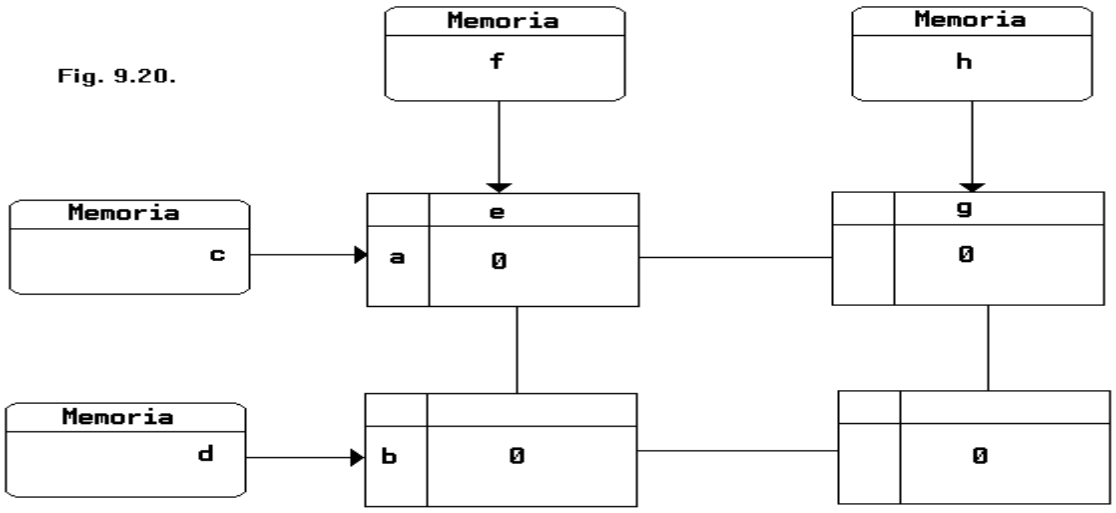
De esta forma un secuenciamiento correcto del cálculo reemplaza al correcto sincronismo de las arquitecturas sistólicas.

En las Fig. 9.20, 9.21 y 9.22 se muestran los conceptos del array wavefront utilizando un ejemplo de multiplicación de dos matrices de 2 * 2 que son de la siguiente forma :

A =	a	c	B =	e	g
	b	d		f	h

La arquitectura de ejemplo utiliza elementos de procesamiento (PE) que tienen un buffer de un operando para cada punto de entrada. Toda vez que el buffer de entrada está vacío y la memoria asociada a ese buffer contiene un nuevo operando, entonces el operando es inmediatamente leído. Los operandos de los otros PEs se obtienen usando el protocolo de handshaking.

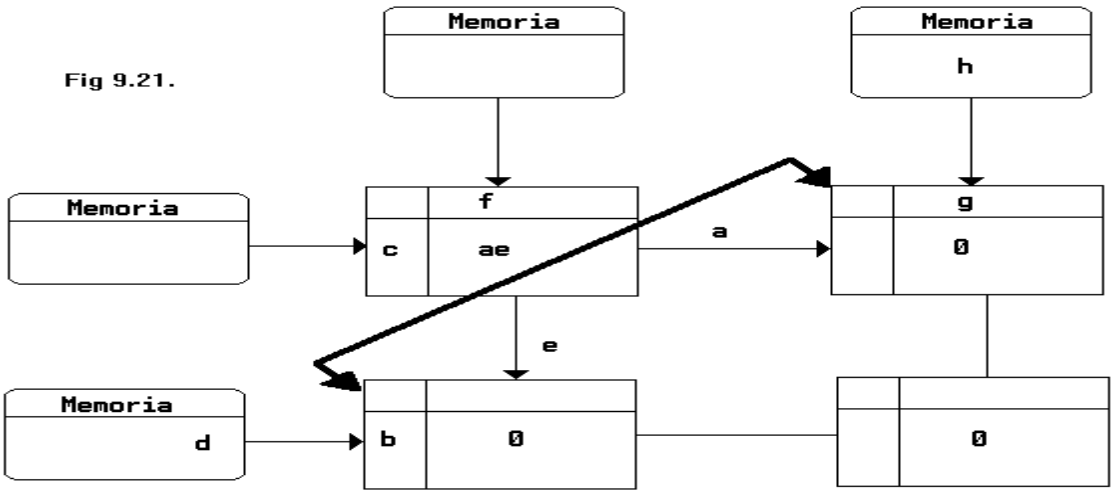
Fig. 9.20.



La Fig. 9.20 muestra la situación inicial cuando se encuentran llenos todos los buffers de entrada.

En la Fig. 9.21 el PE(1,1) suma el producto **ae** a su acumulador y transmite el operando **a** y el **e** a sus veci-

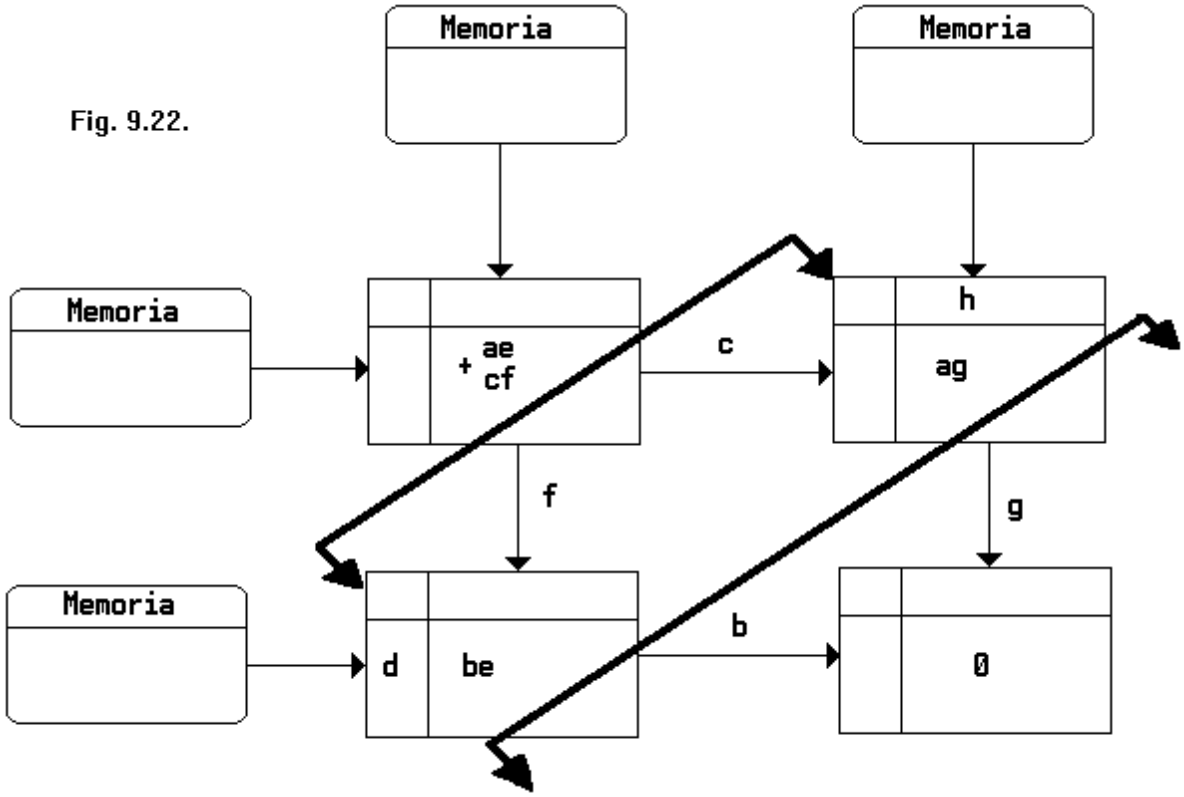
Fig 9.21.



nos; entonces puede verse el primer cálculo wavefront propagándose desde el PE(1,1) al PE(1,2) y al PE(2,1).

La Fig. 9.22 muestra como el primer cálculo continúa propagándose mientras que la segunda ola se empieza a propagar desde el PE(1,1).

Fig. 9.22.



Kung argumenta que los wavefront gozan de diferentes ventajas respecto de los array sistólicos, como por ejemplo, que son más escalables (pueden crecer más fácilmente), son más simples de programar y son más tolerantes a fallas.

EJERCICIOS

1) Cómo es el modelo de cómputo Dataflow y en qué se diferencia del modelo de cómputo de Von Neumann ? Cuál es el elemento clave que los diferencia exactamente ?
2) Cuáles son los tipos fundamentales de operadores en el modelo Dataflow ?
3) Qué es un Token ? Qué clase de información puede contener ?
4) Qué significa que un operador se encuentra en estado "activo" ?
5) Cuál es la importancia de los intérpretes no-feedback ?
6) Qué es una arquitectura dataflow estática ?
7) Qué tipo de intérprete se utilizará en una arquitectura dataflow dinámica ? Justifique.
8) Cuál es el principio de asignación única ?
9) Cómo es la arquitectura Dataflow de modalidad búsqueda ?
10) En qué situación una instrucción ejecutable es equivalente a un token en una máquina Dataflow ? Justifique.
11) Cuál es la diferencia entre una arquitectura Dataflow de modalidad búsqueda y una de modalidad interconexión ?
12) En qué se basan las arquitecturas híbridas de tipo MIMD/SIMD ?
13) En qué principio se basan las arquitecturas de Reducción ?
14) Qué significa la propiedad de transparencia referencial ?
15) Cuáles son las dos características fundamentales de las arquitecturas Wavefront array ?