Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

LA ARQUITECTURA DE UN COMPUTADOR CONCURRENTE

Sergio Mujica
Javier Pinto
Departamento de Ciencia de la Computación
Instituto de Matemática
Pontificia Universidad Católica de Chile

INTRODUCCION

Un programa concurrente hace que se distribuya actividad en varios procesadores, es decir, dado un programa concurrente este será ejecutado por muchos procesadores, los que trabajarán en forma paralela, (también se habla de computación paralela) ejecutando simultáneamente diferentes computaciones, lo que obviamente reduce la complejidad de tiempo de los algoritmos a algo mucho menor (órdenes de magnitud) que en los algoritmos secuenciales basados en el modelo de von Neumman.

El propósito de este trabajo es dar una descripción de la arquitectura de un computador que soporte un lenguaje para escribir programas concurrentes.

REPRESENTACION DE LAS COMPUTACIONES

Normalmente se representan las computaciones como una secuencia de acciones que cambia el estado de un conjunto de objetos. También es posible representarlas individualizando el cambio de estado de los objetos, esto es, podemos pensar que se realiza una computación alterando o cambiando el estado de un número finito, aunque no necesariamente acotado, de objetos que representan el ambiente en que se realizan nuestras computaciones, estos cambios de estado puede significar acciones predeterminadas (E/S), o bien

---

Este trabajo ha sido financiado por la Dirección de Investigaciones de la Pontificia Universidad Católica de Chile (DIUC), contrato N82/79

resultar de la acción misma del objeto o de otro objeto.

OBJETOS DE DATOS

Para seguir adelante daremos una definición más precisa de lo que entenderemos por un objeto de datos.

- Tipo de Datos
  Un tipo de dato primitivo es un conjunto de datos primitivos definido por las operaciones que se pueden usar sobre ellos; a diferencia del concepto tradicional estas operaciones sólo pueden alterar el estado del objeto al que corresponde el tipo de datos. Los tipos de datos primitivos son aquellos propios de la máquina, es decir, son inherentes a los elementos de proceso, (i.e., la máquina es capaz de manejarlos).

  Un tipo de dato derivado debe representarse a nivel de programa mediante una estructura de objetos de datos cuyos tipos pueden ser primitivos o derivados, esto se verá con más claridad una vez que se defina objeto de datos.

- Objeto de Datos
  Un objeto de datos tiene asociado un tipo de dato que dice cuales son las operaciones que pueden modificar su estado. Además estos objetos poseen un comportamiento.

  Esquemáticamente (fig. 1), podemos representar un objeto de datos mediante un contorno rectangular el cual tiene buzones (sockets) de entrada y buzones de salida; el objeto de datos será activado en el momento en que se haya depositado información en alguno de los buzones de entrada, lo que motivará la ejecución de una acción por el objeto de dato y que generará resultados que el objeto depositará en los buzones de salida.

  Llamaremos token al ente que transporta información, que es posible depositar en un buzón, además cada buzón tendrá asociado un tipo de dato que indicará el tipo de información que por él puede circular o que él puede manejar.

INTERCONEXION DE OBJETOS

Para que exista un flujo de información (flujo de datos) se hará necesario interconectar objetos, esta interconexión se hará entre objetos (que no necesariamente estará asociado al mismo tipo de datos) mediante enlaces que asocian un buzón de salida de un objeto, y un buzón de entrada de otros objetos, donde obviamente los buzones así asociados deberán corresponder al mismo tipo de datos para que haya un correcto flujo de datos.

ESTRUCTURA DE UN PROGRAMA

Un programa es una estructura de objetos de datos. Para definir una estructura de este tipo tenemos que seguir los siguientes pasos:

1. Definir las clases de objetos que van a formar parte de nuestra estructura.

2. Definir la estructura: como se relacionan los objetos entre sí; estas relaciones corresponden a enlaces de comunicación.

3. Si al definir el programa hemos usado una clase de objetos cuyo tipo asociado es derivado, tenemos que especificar una representación del tipo de datos.

Ejemplo:
Supongamos que tenemos un archivo de datos y deseamos contar el número de caracteres que éste contiene.
Podemos identificar claramente tres objetos de datos, estos son:

- Archivo de entrada: donde se encuentra el texto a leer
- Contador : cuenta el número de caracteres
- Archivo de salida : donde se imprimirán los resultados.

En base al distinto tipo de información que requieren los objetos para actuar, podemos definir los enlaces de comunicación de los tres tipos de datos antes especificados, además podemos identificar en cada uno de los objetos los buzones correspondientes (fig. 2).

- Archivo de entrada: no requiere de un buzón de entrada, es decir, bastará crear este objeto para que comience a actuar, tendrá un buzón de salida, en el cual el objeto depositará los tokens que lleva la información (caracteres de entrada) contenida en el archivo de entrada.

- Contador: tendrá un buzón, el cual tendrá asociado el tipo de caracter (char), y será donde se depositarán los tokens con los caracteres que el contador contará; el contador recibe un caracter especial (Z) que le indica que debe terminar de contar, entonces deposita el token con el resultado de su acción en el buzón de salida.

- Archivo de Salida: tendrá un buzón de entrada, por el que ingresa un entero, que éste imprime.

Esquemáticamente (fig. 2) podemos ver la forma de interconexión de los objetos ya mencionados; que forman la estructura del programa que deseamos construir.

ESPECIFICACION DE OBJETOS

Para que un objeto quede completamente especificado, es necesario, definir en términos del lenguaje (para los ejemplos usaremos sintaxis ad hoc) el estado inicial del objeto, los buzones (sockets), el tipo de dato a que está asociado el objeto y además debemos describir su comportamiento.

En el ejemplo los objetos se puededen especificar de la siguiente manera:

```
object infile : input
init infile ← open ('file name')
sockets out (c : char)
(eof)    c←' Z'
(¬ eof) c ← read
end
```

```
object contador : integer
init contador ← 0
sockets in (c : char), out (K : integer)
(c = ^Z) K ← contador
(c ≠ ^Z) new contador ← contador + 1
end
```

```
object outfile : output
init outfile ← open ('file name')
sockets in (K : integer)
( ) print (K)
end
```

DESCRIPCION FORMAL DE UN OBJETO

1. Identificación del objeto

   Se da el nombre del objeto y se especifica el tipo de dato a que éste está asociado, en el ejemplo:

   ```
   object infile : input;
   ```

   donde input corresponde a un tipo de dato primitivo

2. Estado inicial

Se especifica el estado inicial del objeto, en el ejemplo:

init infile ←open ('file name')

Especifica que el estado inicial del objeto es open, siendo open una operación que corresponde al tipo de dato input, que requiere un parámetro (nombre del archivo).

3. Especificación del contorno

Se especifican los buzones (sockets) que el objeto dado posee, indicando si ellos corresponden a buzones de entrada o de salida e indicando el tipo de dato asociado a cada buzón; en el ejemplo:

sockets out (c : char)

4. Especificación del comportamiento

El comportamiento de un objeto es un conjunto de pares ordenados B : <p, A> donde p es un predicado (condición) y A es un conjunto de acciones. Una acción posible es depositar el token en un buzón.

En el ejemplo el conjunto es:

$$B = \{ \langle eof,\ c \leftarrow '\hat{}Z' \rangle\ ,\ \langle M\ eof,\ c \leftarrow read \rangle \}$$

donde los predicados son p_1 = eof y p_2 = eof y las acciones correspondientes son:

$$a_1 = c \leftarrow '\hat{}Z';\ a_2 = c \leftarrow read$$

Al crearse una instancia de un objeto se ejecuta lo especificado en init, es decir, pasa al estado inicial.

Una instancia de objeto entra en actividad al recibir un token en uno de sus buzones de entrada. Entonces el objeto evalúa los predicados del conjunto B. La evaluación de un predicado es exitosa si él es verdadero y se dispone de todos los elementos necesarios para evaluarlo; falla si no hay tokens suficientes para evaluarlo o es falso.

La máquina selecciona no determinísticamente un par tal que su predicado se haya evaluado exitosamente y ejecuta las acciones de A.

Además cada objeto tiene un buzón de salida asociado con un buzón de entrada a sí mismo (fig. 3), el que es utilizado para cambiar el estado del objeto, este cambio se realiza especificando, como en el ejemplo anterior:

new object name ← expression

Que equivale a poner un token, con el valor resultante de evaluar la expresión, en el buzón new X cambiando así el estado del objeto X (ver especificación del objeto "contador" en el ejemplo anterior.

DEFINICION DE ESTRUCTURAS

Una estructura es un grafo dirigido, que puede variar a medida que la computación progresa.

Así, una estructura S estará compuesta por arcos y nodos. Los arcos son enlaces de comunicación entre objetos a través de los cuales los tokens transportan datos de un objeto a otro. Los arcos conectan un buzón de salida en un objeto con un buzón de entrada en otro objeto.

En la estructura existen dos tipos de nodos: objetos y reproductores.

Los objetos corresponden a la descripción anterior y los reproductores son nodos a través de los cuales se puede hacer variar la estructura.

Formalmente los podemos definir como sigue:

Def. Un reproductor es un par ordenado [O, S] en que:
S es una estructura en que uno de los objetos tiene un buzón distinguido B y O es un objeto tal que:

i) su procedimiento de inicialización pone su estado en valor nulo.

ii) tiene sólo un buzón de entrada denominado in.

iii) su comportamiento es el siguiente:

(se recibió el token)

- se crea instancias de los objetos y reproductores que forman parte de la estructura S.
- se instalan los enlaces definidos en S
- se instala el enlace que llega al buzón del reproductor en el buzón distinguido B.
- se deposita el token recibido por el reproductor en el buzón B.

Entonces, podemos decir que la definición de una estructura consta de las siguientes partes:

a) Definición de las instancias de objeto que forman parte de la estructura, identificándolas adecuadamente.

b) definición de los reproductores

c) definición de enlaces.

Por ejemplo, podemos definir la estructura del programa que utilizáramos anteriormente, como sigue

```
Structure
Objects
 ⟨i : infile, cont : contador, o : outfile⟩
links
⟨i.c to cont.C⟩
⟨cont.K to 0.K⟩
end
```

Como un segundo ejemplo podemos considerar un arbol binario. Para que la estructura crezca, un reproductor debe generar un nodo con dos hijos reproductores, como esquematiza la figura 4.

Esta estructura se puede formalizar como sigue:

```
Structure
gen⟨F (objects  n : node
gens  ⟨e, r : self⟩
links ⟨F.in to n.root⟩
      ⟨n.ls to l.in⟩
      ⟨n.rs to r.in⟩
 )⟩
links   ⟨ ... to F.in⟩
end
```

REPRESENTACION DE TIPOS DE DATOS

Como se dijo anteriormente un tipo de datos está definido por un conjunto de operaciones que pueden tener parámetros.

Representaremos un tipo de datos como una estructura en la que existirá un contorno externo que contendrá buzones para los parámetros de las operaciones.

Al definir la estructura, se deben especificar, enlaces entre los buzones de parámetros y buzones de entrada en la

estructura (entry points).

Por ejemplo, una representación para objetos del tipo bynary-search-tree, con la única operación insert;

```
type bynary-search-tree
  op insert (n : integer);
object node : integer
sockets in (root : integer)
out        (ls, rs : integer);
init new node ← root
(root > node) rs ← root
(root < node) ls ← root

Structure

    gens <F (objects  <n : node>
             gens     <l, r : self>
             links    <F. in to n.root>
                      <n.ls to l.in>
                      <n.rs to s.in>
             )>
    links
   <insert.n to F.in>
    end structure
end type
```

ELEMENTOS DE PROCESO

La arquitectura que se propone consiste básicamente en una red de elementos de proceso, cada uno de los cuales puede representar a un objeto.

Un elemento de proceso (EP) está formado por tres componentes (figura 5).

E/S representa los buzones, P es capaz de realizar las acciones del comportamiento y M es una memoria.

Denominaremos n-EP a un EP tal que su unidad E/S tenga n líneas de comunicación (una línea de comunicación tiene en

general una cantidad de líneas paralelas).

Cada p_i en la unidad E/S se asocia con una tupla $\langle d, t, n, Q, S \rangle$ en que:

.d es entrada o salida

.t es el tipo de datos del buzón

.n es el nombre del buzón

.Q a) si d = entrada

entonces Q es en cada instante la secuencia de tokens que existe en el buzón.

b) si d = salida

entonces Q es la secuencia vacía.

.S a) si d = entrada

entonces S es la secuencia vacía

b) si d = salida

entonces S es una secuencia de nombres de EP que indica el camino que un token depositado en el buzón debe seguir para llegar a su destino.

Este concepto se desarrollará más adelante al definir la representación y creación de enlaces

Aunque el objeto no necesita conocer los nombres de sus vecinos, se incluye S por eficiencia.

Los datos necesarios para representar un objeto se definen usando una sintaxis similar a PASCAL como sigue:

```
type objeto
     record
       t : tipo;
       b : Array of buzón;
       I : inicialización
       c : Array of comportamiento
end
type buzón = Record n . nombre; ti : tipo end
type comportamiento =
     record
       C : condición;
       A : set of acción
end
```

LA ESTRUCTURA DE LOS RETICULADOS

Los elementos de proceso son los nodos de un reticulado (grafo no dirigido en que de cada nodo sale el mismo número de arcos). Un arco que conecta dos nodos representa una línea de comunicación entre ambos elementos de proceso.

Clasificaremos los reticulados según el número de arcos que salen de cada nodo, de acuerdo a la siguiente definición.

Def. Un n-reticulado es aquel en que cada nodo tiene n arcos de salida, y cada par de nodos tiene a lo más una conexión.

La figura 6 ilustra ejemplos de n-reticulado para n = 4 y 6.

Consideraremos que los elementos de proceso de un n-reticulado se agrupan en unidades que llamaremos n-células.

Def. Una n-célula es una agrupación de n + 1 elementos de proceso en que:

a) Se distingue un elemento central que tiene todos sus arcos conectados a otro nodo de la n-célula.

b) Cada elemento distintio del central está conectado al nodo central y a dos vecinos de la manera que ilustra la figura 7.

c) $n \geqslant 4$

Construiremos reticulados interconectando un conjunto de n-células.

Físicamente, un reticulado de elementos de proceso será un único chip (pastilla).

CREACION DE OBJETOS

Se asigna un objeto a un elemento de proceso, cargando la representación correspondiente al objeto, en el elemento de proceso. Cada instancia de objeto asociada a un elemento de proceso tiene un nombre único, el que corresponde exactamente al nombre del elemento de proceso (su "dirección")

REPRESENTACION DE ENLACES

Un enlace entre dos objetos se representa como una secuencia de nombres de objeto que conectan los dos EP involucrados en el enlace dentro del reticulado

Por ejemplo, un enlace entre los objetos A y B en la figura 8 se puede representar como la secuencia ⟨6,10,11,15⟩ otra sería ⟨6,7,11,15⟩ o ⟨6,7,8,12,16,15⟩

Debido al método de creación de enlaces que se enuncia en el próximo punto, siempre los EP que se nombran en la representación de un enlace tendrán la instancia de un objeto asignada a ellos.

CREACION DE ESTRUCTURAS

Un reproductor, ejecutándose en un EP es capaz de crear una estructura. La creación de una estructura involucra crear instancias de objetos de enlaces.

Para crear una instancia de un objeto, el reproductor envía la representación a un EP vecino al suyo. El EP destino se selecciona de acuerdo a:

a) si algún EP vecino está desocupado (i. e., no hay una instancia de objeto en él), se selecciona.

b) si todos los vecinos están ocupados, se selecciona uno al azar y se le envía el objeto. Este lo redirige de acuerdo a estas mismas reglas.

Durante el viaje de la representación del objeto hasta un EP desocupado, se registra el camino seguido. Una vez instalada la instancia de objeto, se envía de vuelta por el mismo camino la localización (nombre) del EP en que se instaló la instancia.

Para establecer un enlace, el reproductor despacha una sonda, la que viaja en el reticulado buscando los objetos que se deben interconectar. Una vez que ubicó uno de ellos comienza a registrar el camino seguido de tal manera que al encontrar el segundo de ellos, ya conoce un camino que los une.

SOBRE LOS ELEMENTOS DE PROCESO

Es necesario, admitir una posible heterogeneidad de los EP en relación con las operaciones que cada uno es capaz de realizar. Esto se debe a que si todos fueran homogéneos, tendrían que ser capaces de realizar todas las operaciones posibles y serían, cada uno de ellos, demasiado complicados, lo que implicaría que el número de EP por chip sería pequeño.

Podemos especializar los EP de acuerdo a los tipos de dato que sean capaces de manejar. Esta política tiene la ventaja de reducir considerablemente la complejidad de un elemento de proceso.

ELEMENTOS FRONTERA

Es evidente que el número de EP que se puede poner en un chip está limitado. Para interconectar chips para formar un reticulado mayor se ponen en los extremos de un reticulado pequeño elementos frontera. La figura 9 ilustra un 4-reticulado de 9 elementos de proceso, con sus nodos de frontera.

Un elemento frontera puede tener uno ó dos vecinos:

a) si tiene un vecino, actúa como un muro, haciendo rebotar los mensajes que le envía su único vecino.

b) si tiene dos vecinos, uno de ellos debe ser un elemento frontera en un chip distinto. La comunicación entre elementos frontera en chips distintos es obviamente mucho más lenta que con otro del mismo chip.

Los elementos frontera permiten la comunicación entre elementos de proceso ubicados en chips diferentes.

CONCLUSIONES

La tecnologia VLSI actual [Mead-Conway 80], hace posible la realizacion fisica de una arquitectura como la propuesta.

Por otro lado el lenguaje que hemos descrito y que es aquel definido en [Mujica 80], nos permite escribir programas altamente concurrentes utilizando el enfoque de objetos, cuya deseabilidad es evidente [Wegner 79].

REFERENCIAS

[Mead-Conway 80] Carver Mead. Lynn Conway
"Introduction to VLSI systems"
Addison-Wesley, Reading, Massachussets, 1980.

[Mujica 80] Sergio Mujica, "Un modelo semántico para computación concurrente", Actas de la séptima conferencia latino-americana de informática, Caracas-Venezuela, Febrero 1980.

[Wegner 79] Peter Wegner, "Programming Languages-Concepts and research Directions", in Research Directions in Software Technology, (p. Wegner, ed.), The MIT Press, Cambridge, Mass, 1979.

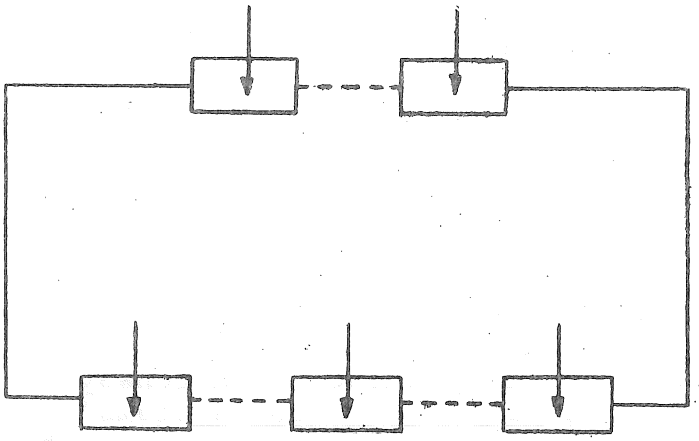


FIG. 1

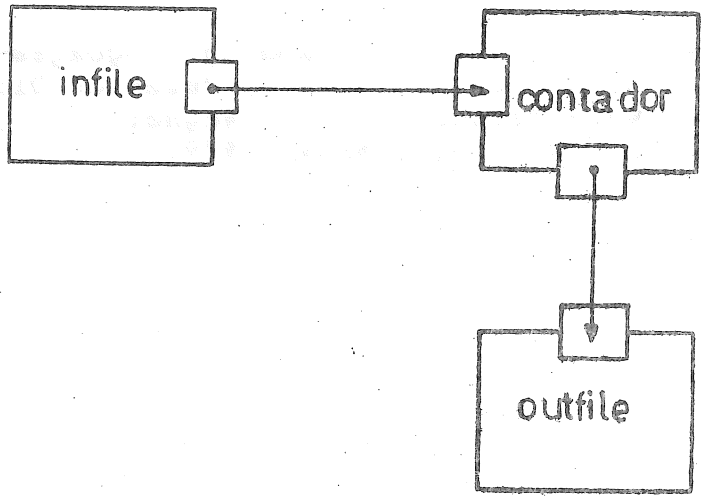


FIG. 2

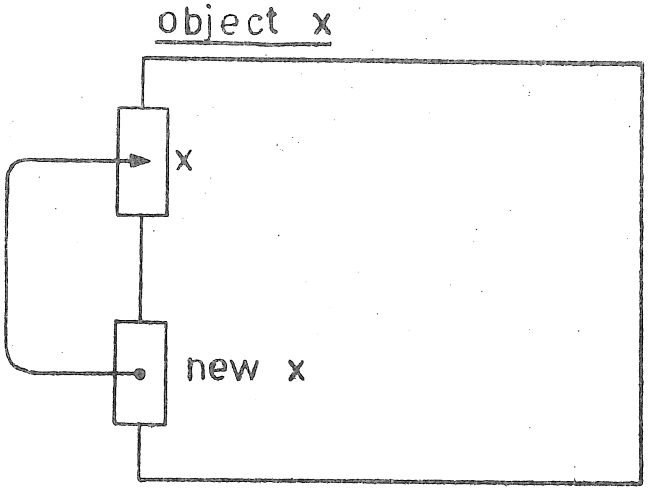


FIG. 3

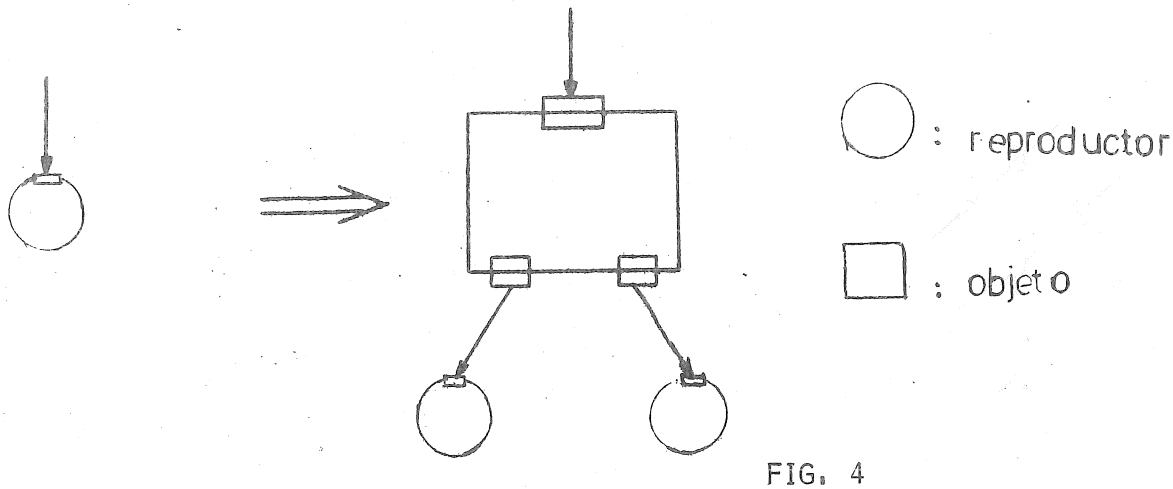


FIG. 4

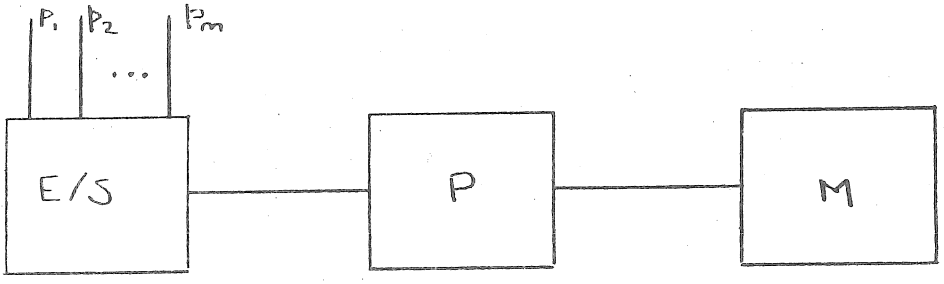


FIG. 5

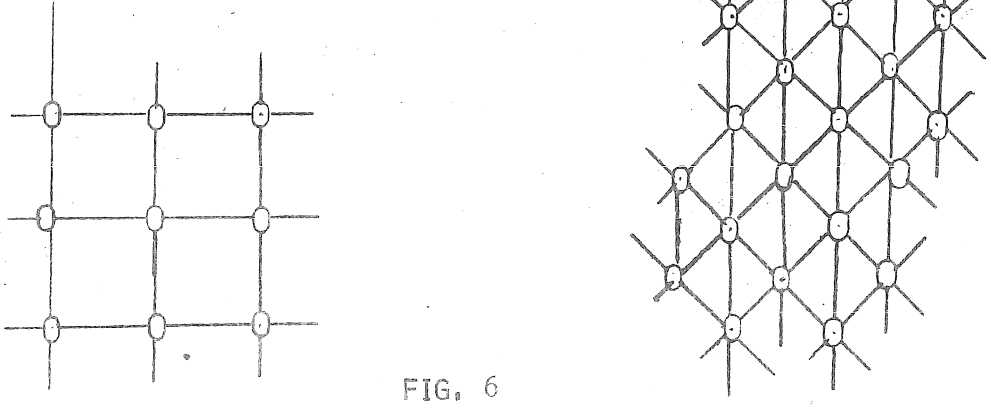


FIG. 6

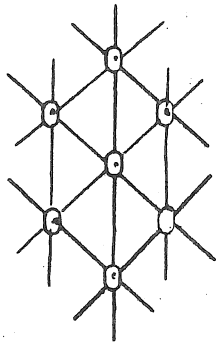
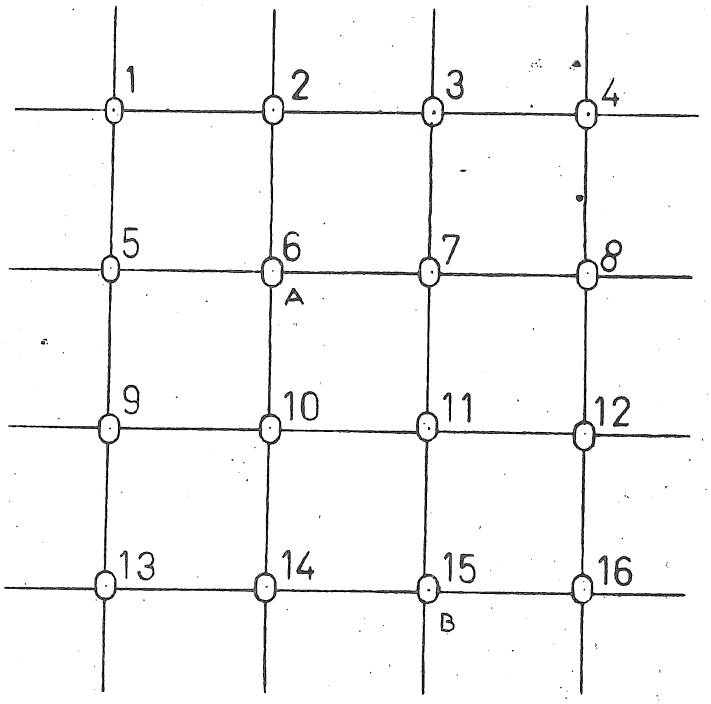


FIG. 7



Un reticulado FIG. 8

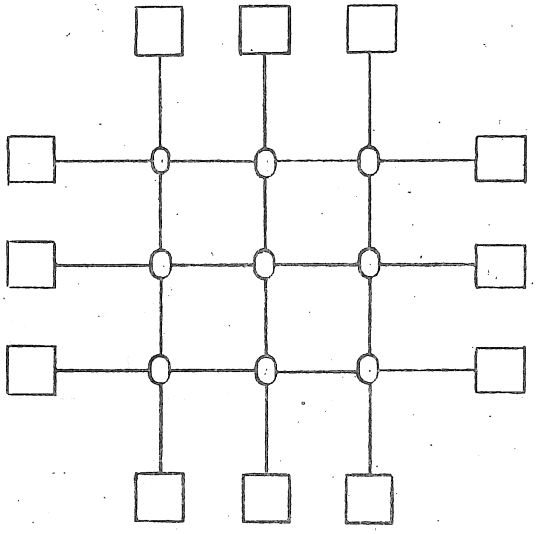


FIG. 9

O : Elemento de proceso

□ : Elemento frontera