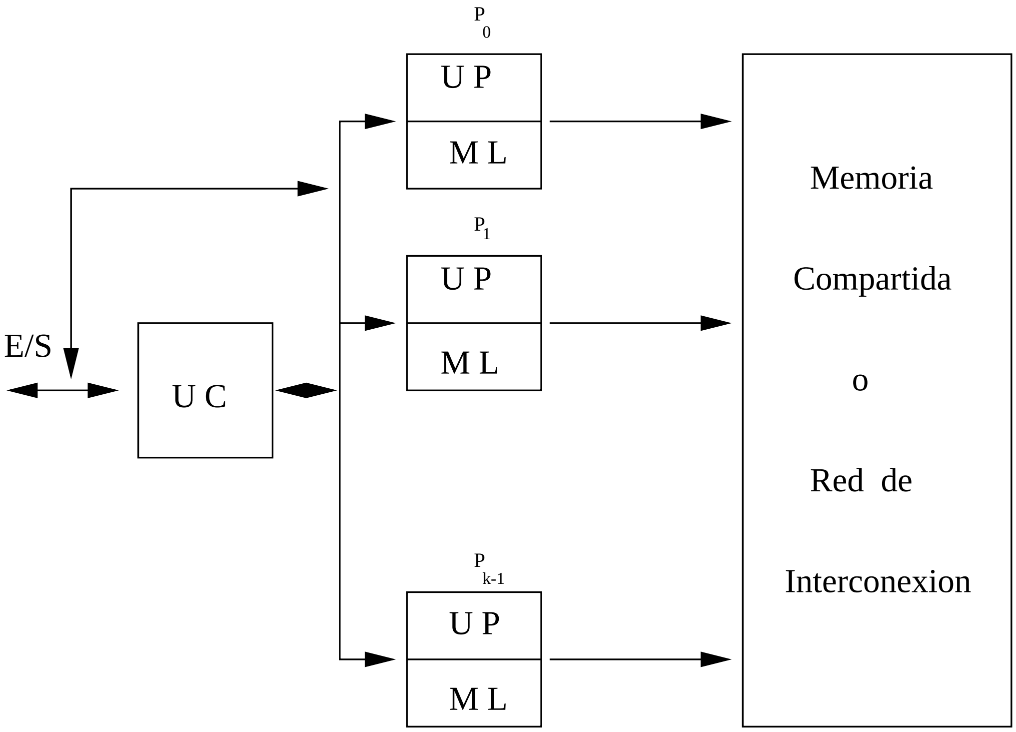
Tema III: Modelos de Computación SIMD

Array Processor Machines: Constan de un número finito de procesadores idénticos: $P_0, P_1, \ldots, P_{k-1}$, gobernados por una unidad de control UC.

Estas máquinas pueden variar:

- Por el número de procesadores que utilice (fijo o variable).
- Por el mecanismo de comunicación entre los procesadores:
  - Memoria compartida.
  - Red de interconexión.



Ejecución de un algoritmo paralelo en una **SIMD**:

- La UC selecciona una instrucción que comunica simultáneamente a todos los procesadores.

- Cada procesador ejecuta simultáneamente la instrucción seleccionada.

- Cada procesador tiene un dispositivo que le permite aceptar (*activo*) o ignorar (*inactivo*) una instrucción.

Algunos modelos **SIMD**

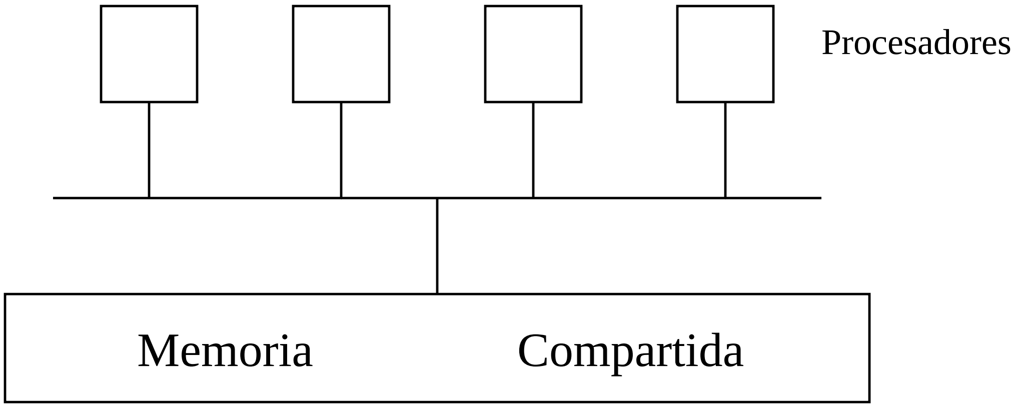
- Memoria compartida: **PRAM**.

- Red de interconexión:
  - Malla (**MC**).
  - Hipercubo (**CC**) e Hipercubo de ciclos.
  - Red de intercambio–mezcla (**PS**)
  - Pirámide (**P**).
  - Mariposa (**B**)
  - Árbol (**TCC**).

Máquinas de memoria compartida SIMD–PRAM

- Todo procesador puede acceder a cualquier registro de memoria.

- La rapidez de acceso provoca problemas.



Clasificación según el grado de concurrencia:

- **PRAM–EREW**,
- **PRAM–CREW**,
- **PRAM–ERCW**,
- **PRAM–CRCW**.

Simulación de máquinas PRAM

Sean M_1 y M_2 modelos de computación paralela:

- M_1 es *menos potente* que M_2 *para resolver un problema* X ($M_1 <_X M_2$) si el tiempo de ejecución de cualquier algoritmo de M_1 para resolver X es mayor o igual que el tiempo que necesita algún algoritmo de M_2.

- M_1 es *menos potente* que M_2 ($M_1 < M_2$) si para cada problema X, $M_1 <_X M_2$.

- M_1 es *estríctamente menos potente* que M_2 ($M_1 << M_2$) si $M_1 < M_2$ y existe un problema que M_2 lo resuelve estríctamente mejor que M_1.

- M_1 puede ser *simulado* por M_2 en tiempo t ($M_2 = M_1 \cdot t$) si un paso de ejecución de M_1 puede ser ejecutado sobre M_2 en tiempo t.

Se tiene que

$$\text{EREW-PRAM} << O(\lg k) \text{ CREW–PRAM} << \\ << O(\lg k) \text{ CRCW–PRAM}$$

(cada modelo con k procesadores).

Simulación de una CREW–PRAM en una EREW–PRAM

Sea A un algoritmo CREW–PRAM que usa $k = 2^r$ procesadores y M registros de memoria.

Simularemos A mediante un algoritmo EREW–PRAM que usa k procesadores y $M \cdot (2k-1)$ registros de memoria.

- Cada registro usado proporciona un árbol binario completo con k hojas.
- Si en un instante, varios procesadores P_i necesitan leer el contenido, C, del registro R_j, colocarán un puntero de petición de lectura, L, en la hoja i–ésima del árbol de raíz R_j.

El algoritmo de simulación consta de dos fases:

- Una fase hacia adelante y otra hacia atrás (ambas con tiempo ejecución lg$k$).

La simulación de lectura concurrente en una EREW–PRAM tiene un tiempo de ejecución $O(\text{lg}k)$ utilizando un espacio $M \cdot (2k-1)$, siendo M el espacio utilizado por la CREW–PRAM.

EREW–PRAM << CREW–PRAM

Problema: *Sea un árbol binario completo con* $n = 2^s$ *nodos, cada uno de ellos con dos punteros,* q_i *y* r_i*, de tal manera que* q_i *señala a su padre (la raíz =NIL). Se trata de conseguir que cada* r_i *señale a la raíz.*

Un algoritmo CREW–PRAM que resuelve el problema con n^2 procesadores en tiempo de ejecución $\theta(1)$:

procedimiento $buscar{-}raiz_{CREW}(T[1, \ldots, n])$

  para cada P_{ij} tal que $1 \le i, j \le n$ **hacer**

    si $q_j = NIL$ **entonces**

      $r_i \leftarrow j$

Ningún algoritmo EREW–PRAM resuelve el problema en tiempo de ejecución $\theta(1)$, aunque tanga un número ilimitado de procesadores.

Simulación de una ERCW–PRAM en una EREW–PRAM

Sea A un algoritmo ERCW–PRAM que usa $k = 2^r$ procesadores y M registros de memoria.

Simularemos A mediante un algoritmo EREW–PRAM que usa k procesadores y $M \cdot (2k-1)$ registros de memoria.

- Cada registro usado proporciona un árbol binario completo con k hojas.
- Si en un instante, varios procesadores P_i intentan escribir en el registro R_j, colocará un puntero de petición de escritura, E_i, en la hoja i–ésima del árbol de raíz R_j.
- En cada paso, la petición se traslada al padre con prioridad para el hijo izquierdo.
- Tras $r = \lg k$ pasos, el procesador cuyo índice es menor, escribe en la correspondiente raíz.

Nota: Resolución análoga si el conflicto de escritura se resuelve en función de si todos desean escribir lo mismo.

CREW–PRAM << CRCW–PRAM

Problema: *Hallar el elemento máximo de una lista numérica* $A[1,\ldots,n]$.

Un algoritmo CRCW–PRAM que resuelve el problema con n^2 procesadores en tiempo de ejecución $\theta(1)$:

procedimiento $maximo_{CRCW}(A[1,\ldots,n], B[1,\ldots,n])$

    para cada P_{ij} tal que $1 \leq i, j \leq n$ **hacer**

        si $A[i] < A[j]$ **entonces**

            $B[i] \leftarrow 0$

    para cada P_{i1} tal que $1 \leq i \leq n$ **hacer**

        si $B[i] = 1$ **entonces**

            $max \leftarrow A[i]$

    devolver max

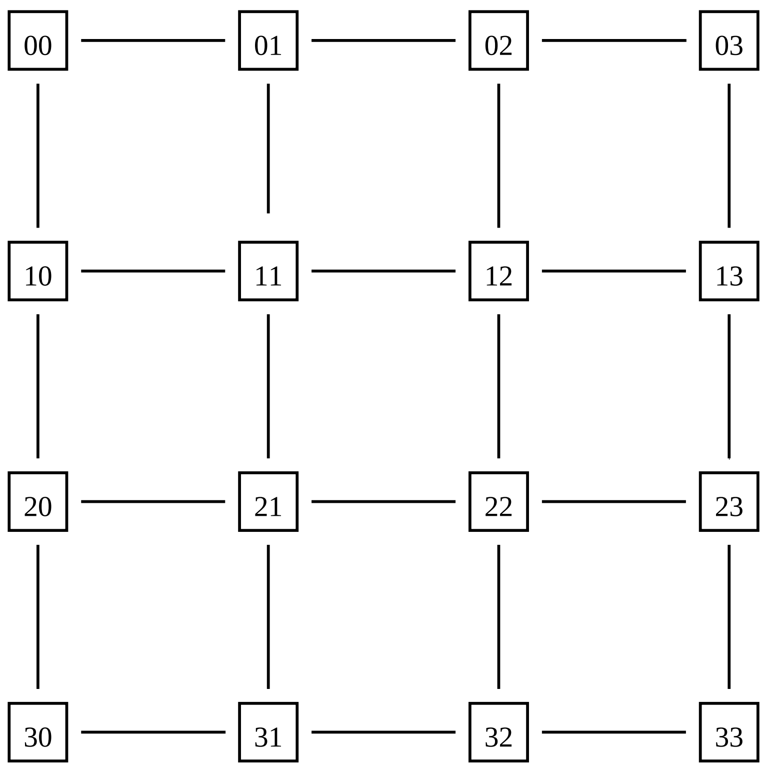
Se prueba que **cualquier** algoritmo CREW–PRAM que resuelva el problema necesitará un tiempo de ejecución $\Omega(\lg n)$, incluso usando un número ilimitado de procesadores.

Malla (Mesh Connected Computer)

En una malla de dimensión q, los procesadores están dispuestos en forma de retículo q–dimensional.

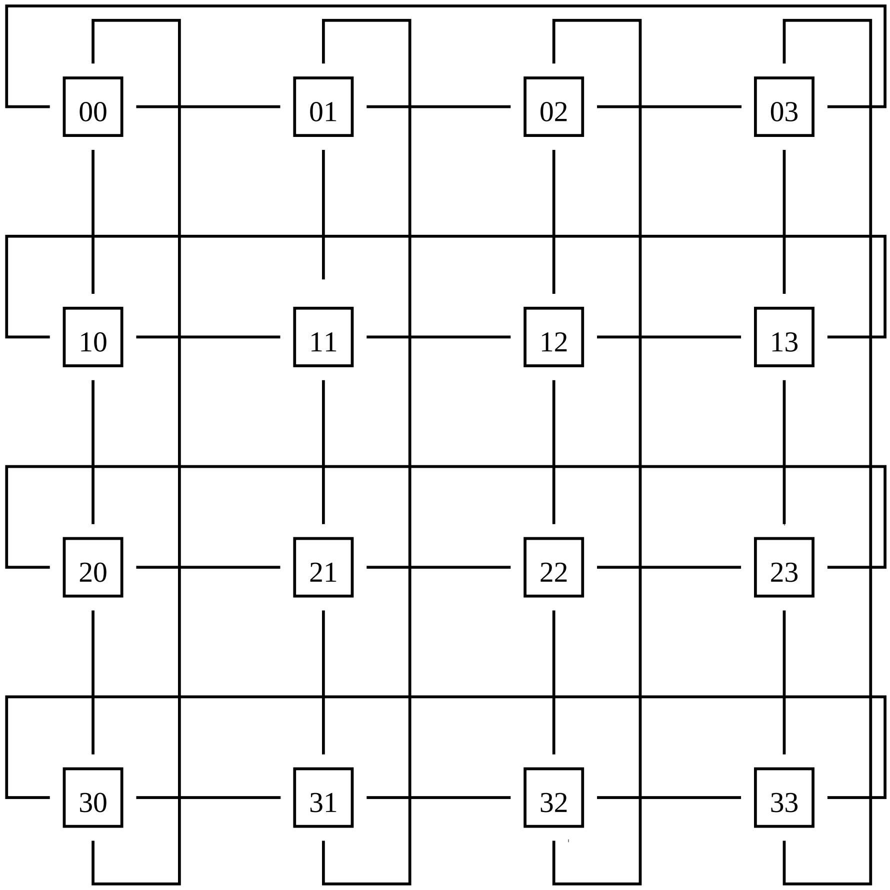
Se puede identificar con un array $A[n_0, n_1, \ldots, n_{q-1}]$, en donde n_i es el número de procesadores de la componente i–ésima.

Ejemplo de una malla **MC**^{2} con $k = 16$ procesadores:



Mallas por filas y columnas.

Ejemplo: $q = 2$ y $k = 16$.

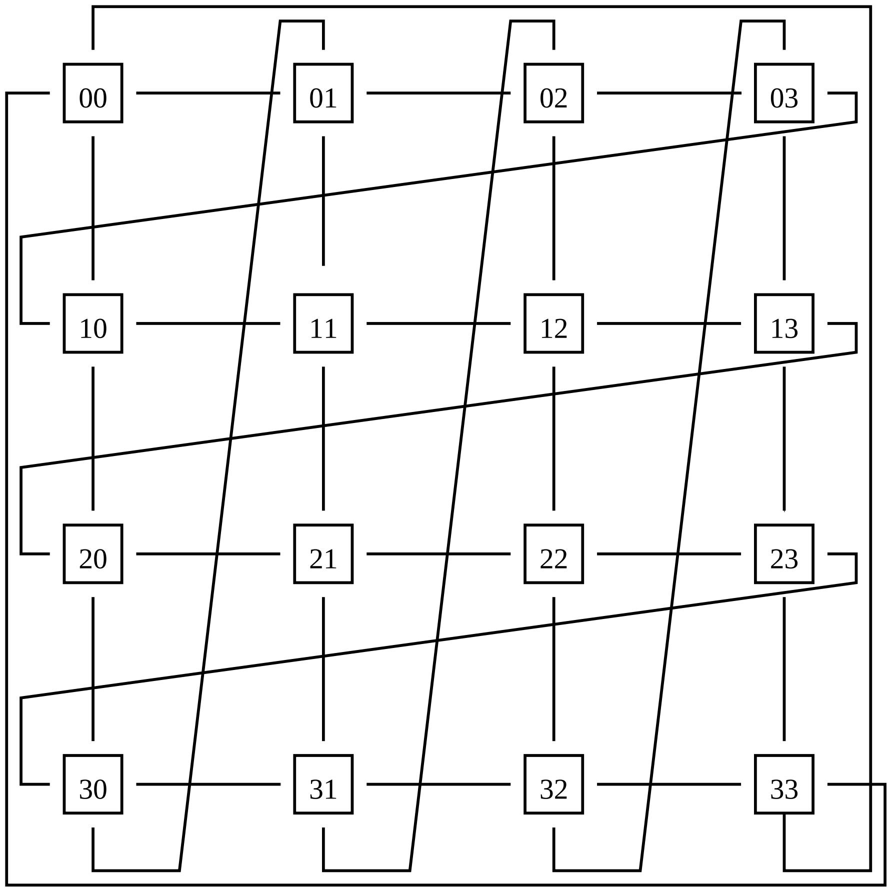


- Cada nodo está conectado con otros 4.

- Cada fila (resp. columna) determina un ciclo de orden 4.

Mallas con conexión toroidal.

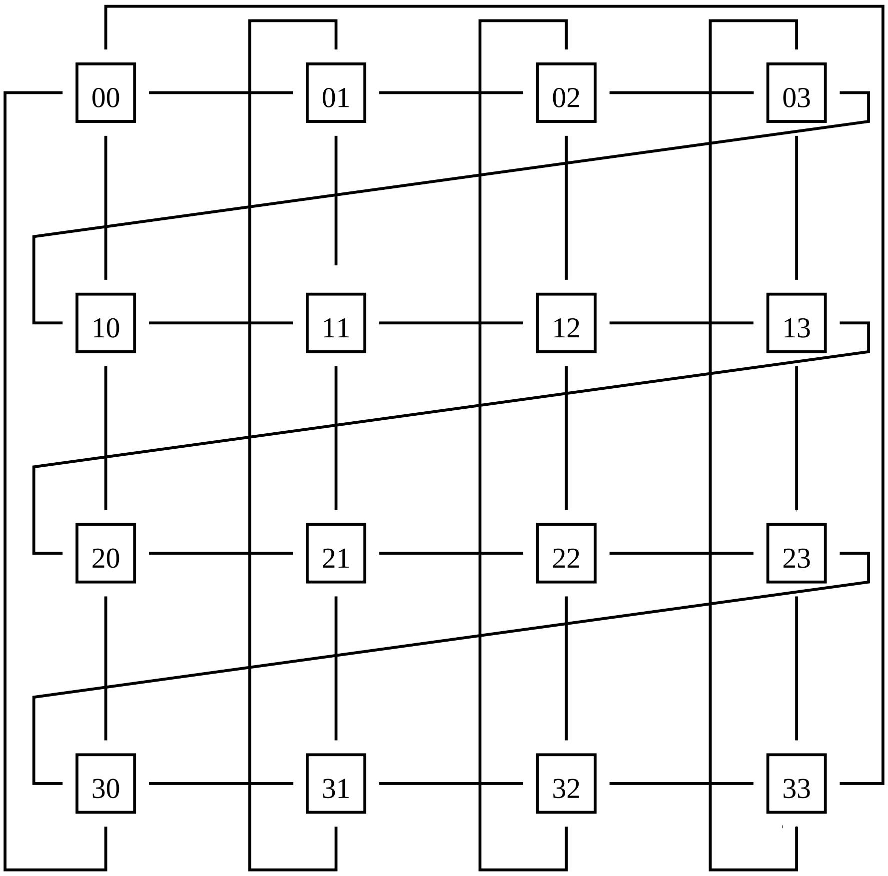
Ejemplo: $q = 2$ y $k = 16$.



- El conjunto de las filas (resp. columnas) determinan un ciclo de orden 16.

Mallas determinadas por otras funciones de interconexión.

Ejemplo: $q = 2$ y $k = 16$.



Es la red utilizada en el ILLIAC-IV.

- Cada procesador puede comunicarse con cualquier otro en $O(\sqrt{k} - 1)$ pasos.

Problema 1: *Dados $n = r^2$ números enteros $a_{11}, a_{12}, \ldots, a_{rr}$, hallar la suma de dichos números.*

Vamos a resolver dicho problema en una malla "ordinaria" **MC**^{2} con $k = n = r^2$ procesadores.

Cada procesador P_{ij} tiene dos variables: s_{ij} (inicializada en a_{ij}) y t_{ij} (inicializada en 0).

procedimiento $suma_{\mathrm{MC}^2}(a_{11}, \ldots, a_{rr})$

  para $i \leftarrow r-1$ **hasta** 1 **hacer**

    para cada P_{ji} tal que $1 \leq j \leq r$ **hacer**

      $t_{ji} \Longleftarrow s_{j(i+1)}$

      $s_{ji} \leftarrow s_{ji} + t_{ji}$

  para $i \leftarrow r-1$ **hasta** 1 **hacer**

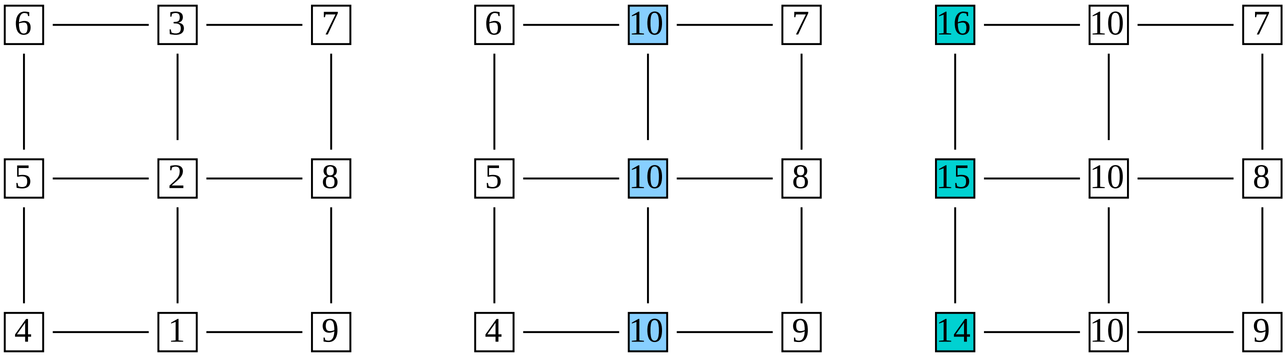
      $t_{i1} \Longleftarrow s_{(i+1)1}$

      $s_{i1} \leftarrow s_{i1} + t_{i1}$

  devolver s_{11}

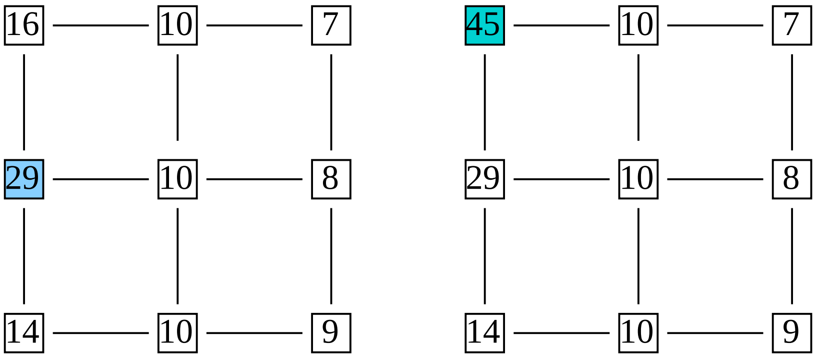
Ilustración: sumar los números de la lista $(6, 3, 7, 5, 2, 8, 4, 1, 9)$.

En este caso $k = n = 9$ y $r = 3$.



Inicio   i = 2   i = 1

Ejecución del segundo bucle principal



i = 2   i = 1

Conclusión: $s_{11} = 45$.

Teorema 1. El algoritmo $suma_{\mathbf{MC}^2}$ es correcto para hallar la suma de n números enteros.

Demostración: Obviamente el algoritmo **para** sobre cualquier dato de entrada.

Para establecer la corrección parcial, inicializamos la variable i en r.

Consideremos la fórmula

$$\varphi(i) \equiv \forall j \ (1 \leq j \leq r \rightarrow s_{ji} = \sum_{k=i}^{r} a_{jk})$$

Entonces, dicha fórmula

- Es verdadera inmediatamente antes de la ejecución del primer bucle principal.

- Es un invariante del primer bucle principal (por inducción acotada descendente sobre i).

Por tanto, tras la ejecución de ese bucle será verdadera $\varphi(1)$.

Consideremos la fórmula

$$\psi(i) \equiv s_{i1} = \sum_{j=i}^{r} \sum_{k=1}^{r} a_{jk}$$

Inicializando nuevamente i en el valor r, dicha fórmula:

- Es verdadera inmediatamente antes de la ejecución del primer bucle principal.

- Es un invariante del segundo bucle principal (por inducción acotada descendente sobre i).

Por tanto, tras la ejecución del algoritmo será verdadera $\psi(1)$.

Es decir,

$$s_{11} = \sum_{j=1}^{r} \sum_{k=1}^{r} a_{jk}$$

$\blacksquare$

Análisis del algoritmo:

- $t(n) = n$.
- $t(n, n) \in \theta(2(\sqrt{n} - 1)) = \theta(\sqrt{n})$.
- $S(n, n) = \frac{n}{\sqrt{n}}$.
- $C(n, n) = n \cdot \sqrt{n}$.
- $E(n, n) = \frac{n}{n \cdot \sqrt{n}} = \frac{1}{\sqrt{n}}$.

Como $\sqrt{n} >> \lg n$, resulta que este algoritmo **no** es eficiente.

Problema 2: *Dada una lista de números enteros* $a_{11}, a_{12}, \ldots, a_{rr}$ *y un número entero* x, *determinar si* x *está en la lista.*

Resolvemos el problema en una malla **MC**^{2} con $k = n = r^2$ procesadores, P_{ij}. La entrada y salida se hará a través del procesador P_{11}.

Cada procesador P_{ij} contiene cuatro variables:

- a_{ij} con el correspondiente valor de la lista.
- b_{ij} (booleana) que contiene el test de chequear si $x = a_{ij}$
- u_{ij} donde P_{ij} guarda el valor de b_{kl} que le comunica P_{kl}.
- t_{ij} que ubicará el elemento x.

El algoritmo consta de dos fases:

- Fase de **comparación y distribución**: se envía x a todos los procesadores, que lo comparan con a_{ij} y actualizan b_{ij}.

- Fase de **concentración**: se propaga de forma alternativa, y hacia arriba de manera *consistente*, los valores b_{ij} hasta *llegar* a b_{11}.

Implementación de la fase de **comparación y distribución.**

(Inicialmente $t_{11} = x$).

si $t_{11} = a_{11}$ **entonces**

$b_{11} \leftarrow TRUE$

devolver b_{11}

si no

para $i \leftarrow 1$ **hasta** $r - 1$ **hacer**

para cada P_{ji} tal que $1 \leq j \leq i$ **hacer**

$t_{j(i+1)} \Longleftarrow t_{ji}$

si $t_{j(i+1)} = a_{j(i+1)}$ **entonces**

$b_{j(i+1)} \leftarrow TRUE$

si no

$b_{j(i+1)} \leftarrow FALSE$

para cada P_{ij} tal que $1 \leq j \leq i + 1$ **hacer**

$t_{(i+1)j} \Longleftarrow t_{ij}$

si $t_{(i+1)j} = a_{(i+1)j}$ **entonces**

$b_{(i+1)j} \leftarrow TRUE$

si no

$b_{(i+1)j} \leftarrow FALSE$

Implementación de la fase de **concentración.**

para $i \leftarrow r$ **hasta** 2 **hacer**

  para cada P_{ji} tal que $1 \leq j \leq i$ **hacer**

    $u_{j(i-1)} \Longleftarrow b_{ji}$

    si $u_{j(i-1)} = TRUE$ **entonces**

      $b_{j(i-1)} \leftarrow TRUE$

  para cada P_{ij} tal que $1 \leq j \leq i-1$ **hacer**

    $u_{(i-1)j} \Longleftarrow b_{ij}$

    si $u_{(i-1)j} = TRUE$ **entonces**

      $b_{(i-1)j} \leftarrow TRUE$

devolver b_{11}

El tiempo de ejecución del algoritmo es del orden $\theta(r) = \theta(\sqrt{n})$.

El análisis del este algoritmo es idéntico al descrito en la resolución del problema 1.

Hipercubo (Cube Connected, CC)

Un hipercubo, **CC**^{r}, de dimensión r está caracterizado por las siguientes condiciones:

(a) Es un grafo de tamaño 2^r que consta de $r \cdot 2^{r-1}$ aristas.

(b) Representando los nodos en base 2, para cada i $(0 \leq i \leq r-1)$ el nodo

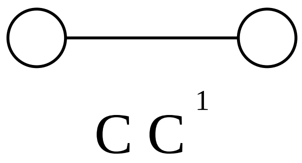
$$a_{r-1}a_{r-2}\ldots a_{i+1}a_i a_{i-1}\ldots a_0$$

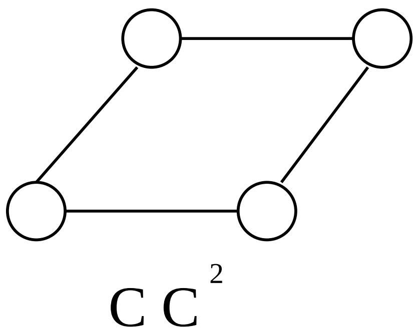
está conectado con

$$a_{r-1}a_{r-2}\ldots a_{i+1}a_i^* a_{i-1}\ldots a_0$$

en donde $a_i^* = 1 - a_i$.

Ejemplos:





Problema: *Dados* $n = 2^r$ *números enteros* $a_0, a_1, \dots, a_{n-1}$, *hallar la suma,* S, *de dichos números*

Vamos a dar un algoritmo **SIMD–CC**^{r} con $k = n = 2^r$ procesadores que lo resuelve.

Cada procesador P_i $(0 \leq i \leq n-1)$ tiene dos variables:

- s_i inicializada en a_i.
- t_i inicializada en 0.

procedimiento $suma_{\mathbf{CC}}(a_0, a_1, \dots, a_{n-1})$

    para $i \leftarrow$ lg$n - 1$ **hasta** 0 **hacer**

        $d \leftarrow 2^i$

        para cada P_j tal que $0 \leq j < d$ **hacer**

            $t_j \Longleftarrow s_{j+d}$

            $s_j \leftarrow s_j + t_j$

    devolver s_1

Como $d = 2^i$ resulta que P_{j+d} y P_j están conectados.

Teorema: *El algoritmo suma*$_{\mathrm{CC}}$ *es correcto.*

Demostración: Por metainducción sobre la estructura del algoritmo.

Consideremos la fórmula

$$\varphi(i) \equiv S = \sum_{k=0}^{2^i-1} a_k$$

para cada i tal que $0 \leq i \leq r-1$.

1. Añadiendo la instrucción $i \leftarrow r$ al comienzo del algoritmo, φ es verdadera inmediatamente antes del comienzo de la ejecución del bucle principal.

2. φ es un invariante del bucle principal (por inducción débil descendente acotada sobre i).

3. φ es verdadera tras ejecutar el conjunto F de instrucciones.

4. Al finalizar la ejecución del algoritmo, la fórmula $\varphi(0)$ será verdadera.

Análisis del algoritmo:

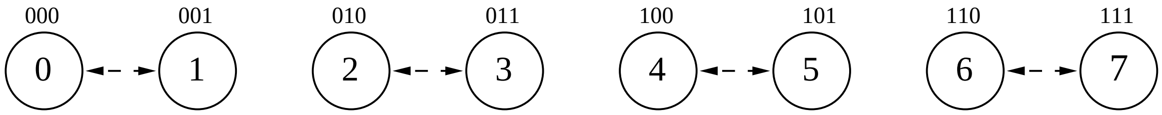
- $t(n) = n$.
- $t(n, n) = \text{lg} n$.
- $S(n, n) = \frac{n}{\text{lg} n}$.
- $C(n, n) = n \cdot \text{lg} n$.
- $E(n, n) = \frac{n}{n \cdot \text{lg} n} = \frac{1}{\text{lg} n}$.

Por tanto, el algoritmo $suma_{\mathbf{CC}}$ es eficiente.

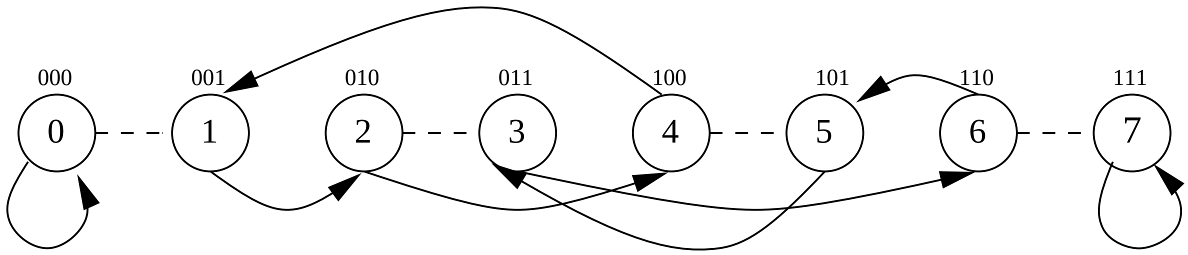
Red de intercambio–mezcla (PS: Perfect shuffle network)

Un modelo **SIMD** con red de intercambio–mezcla consta de $k = 2^r$ procesadores enumerados de 0 a $k-1$ conectados entre sí de dos posibles maneras:

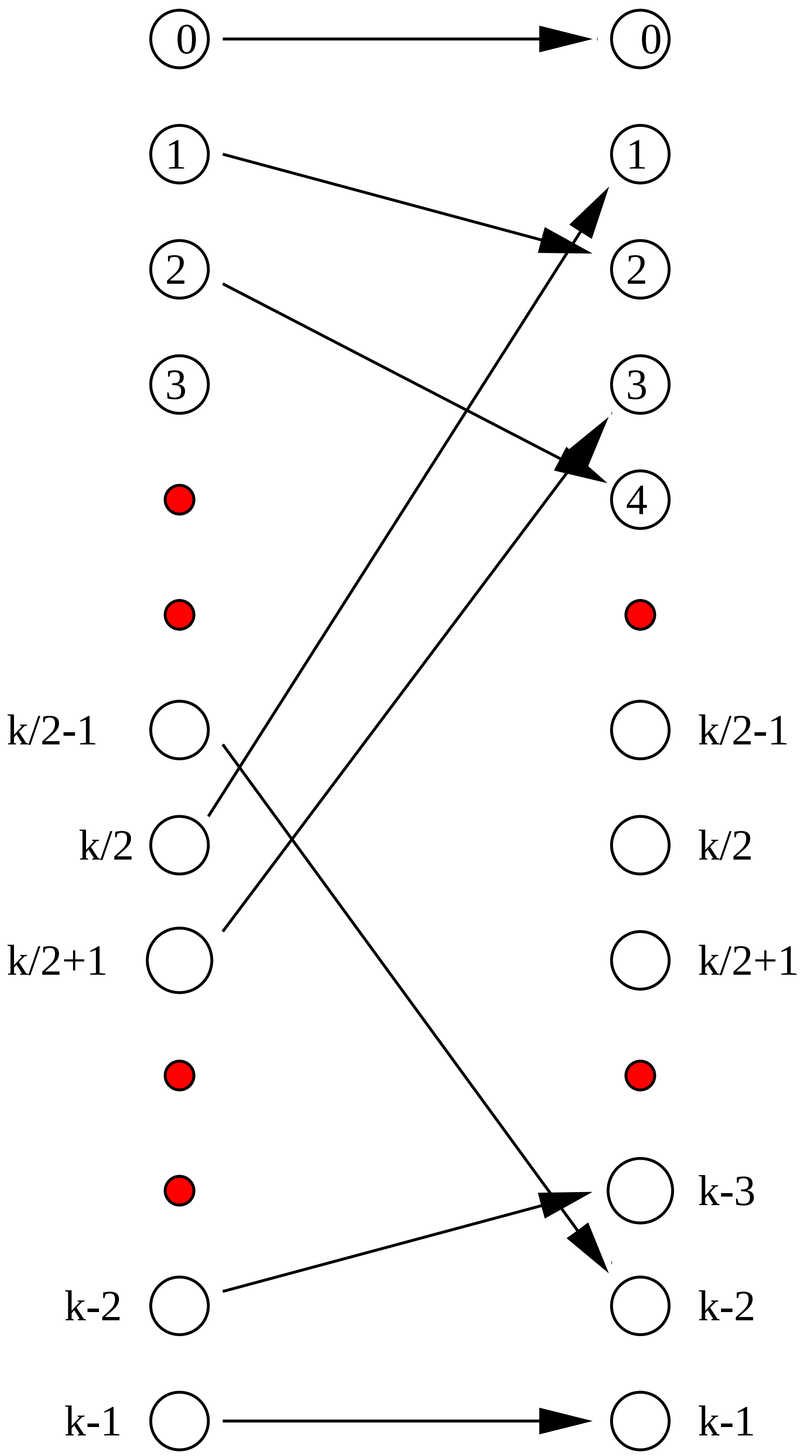
- Conexión de **intercambio**: P_i está conectado con P_j sii los números i, j en binario (con r dígitos) difieren sólo en las unidades.



- Conexión de **mezcla**: P_i $(i \neq n-1)$ está conectado con P_j (de i a j) sii $j \equiv 2i \pmod{n-1}$. P_{n-1} está conectado sólo consigo mismo.



En general, la conexión mezcla se describe gráficamente así:



Función mezcla:

$$S(a_{r-1}a_{r-2}a_{r-3}\ldots a_1a_0) = a_{r-2}a_{r-3}\ldots a_1a_0a_{r-1}$$

Función intercambio:

$$E(a_{r-1}a_{r-2}\ldots a_1a_0) = a_{r-1}a_{r-2}\ldots a_1a_0^*$$

con $a_0^* = 1 - a_0$.

Un *collar* es un ciclo respecto de la conexión mezcla.

Proposición: *En una máquina* **SIMD–PS** *con* $k = 2^r$ *procesadores, la longitud de un collar es menor o igual que* r*.*

Se dice que un *collar* es *corto* si su longitud es $< r$. Caso contrario, se dice *largo*.

En una **SIMD–PS** con $k = 2^3$ procesadores sólo los collares triviales son cortos.

Problema: *Dados* $n = 2^r$ *números enteros* $a_0, a_1, \ldots, a_{n-1}$, *hallar la suma,* S, *de dichos números*

Vamos a dar un algoritmo **SIMD–PS** con $k = n = 2^r$ procesadores que lo resuelve.

Cada procesador P_i $(0 \leq i \leq n-1)$ tiene dos variables:

- s_i inicializada en a_i.
- t_i inicializada en 0.

procedimiento $suma_{\mathbf{PS}}(a_0, a_1, \ldots, a_{n-1})$

    para $i \leftarrow 1$ **hasta** lg$n$ **hacer**

        para cada P_j tal que $0 \leq j \leq n-1$ **hacer**

            mezcla (s_j)

            $t_j \leftarrow s_j$

            intercambio (t_j)

            $s_j \leftarrow s_j + t_j$

    devolver s_0

En donde

- **mezcla** (s_j) significa que si $P_l \overset{mezcla}{\rightarrow} P_j$, entonces P_l comunica a P_j el valor s_l que P_j guarda en s_j.

- **intercambio** (t_j) significa que si $P_l \overset{interc}{\leftrightarrow} P_j$, entonces los procesadores permutan los correspondientes valores de t_l y t_j.

Análisis del algoritmo:

- $t(n) = n$.

- $t(n,n) = \lg n$.

- $S(n,n) = \frac{n}{\lg n}$.

- $C(n,n) = n \cdot \lg n$.

- $E(n,n) = \frac{n}{n \cdot \lg n} = \frac{1}{\lg n}$.

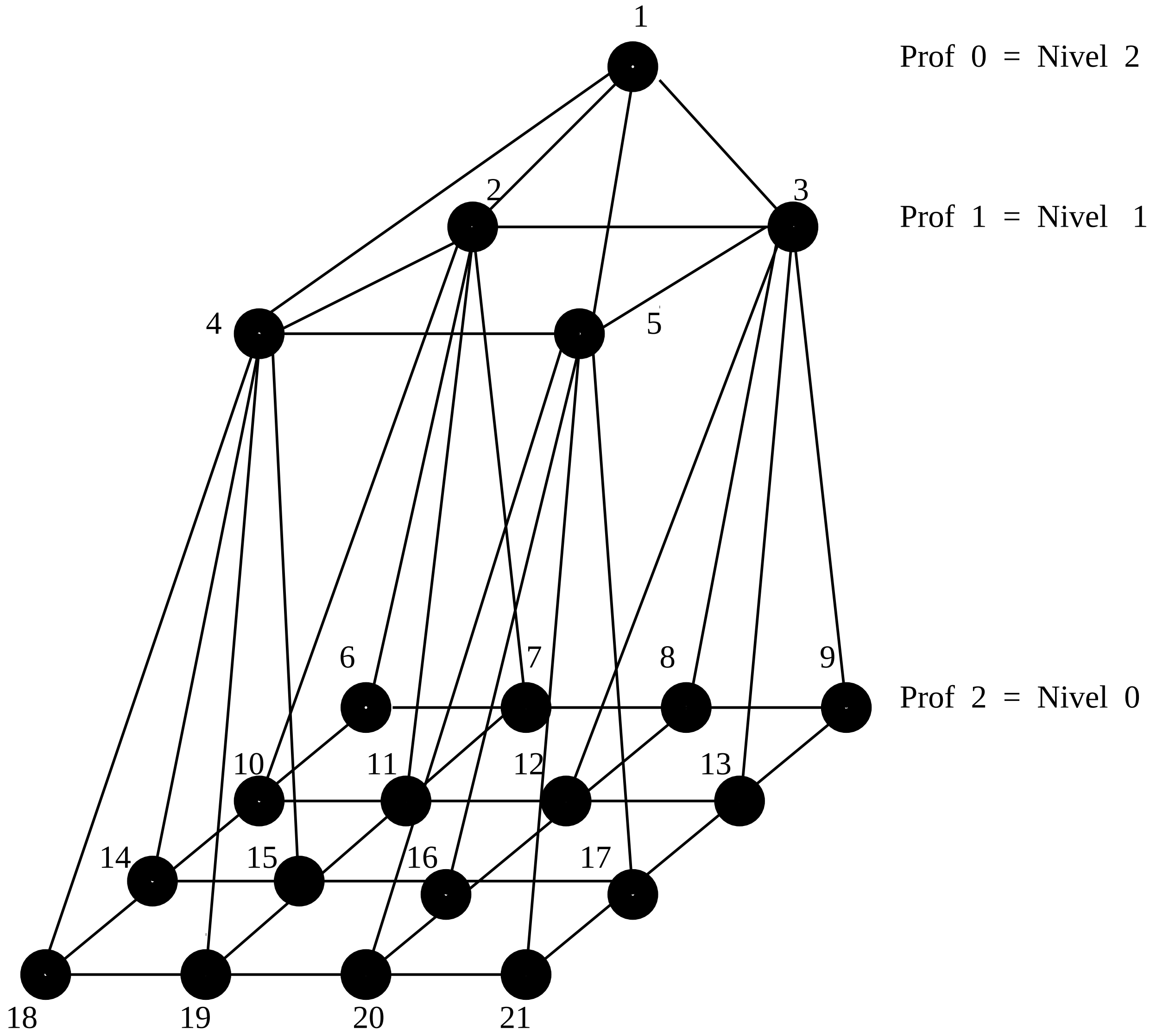
Por tanto, el algoritmo $suma_{\mathbf{PS}}$ es eficiente.

Corrección del algoritmo: analizar la ruta que sigue cada dato, a_j, y ratificar que se suma en s_0 al final del proceso.

Pirámide (P)

Pirámide de tamaño $p = 4^r$: modelo **SIMD** con una red de comunicaciones en forma de árbol 4–ario completo tal que cada nivel forma una malla **MC**^{2}.

Ejemplo: **SIMD–P** de tamaño $p = 16$.



En una **SIMD–P** de tamaño $p = 4^r$:

- Para cada h $(0 \leq h \leq r)$, el nº de nodos de profundidad h es 4^h.

- Existen $\frac{4p-1}{3}$ nodos.

- Cada nodo "interior" está conectado con 9 nodos.

- Se pueden enumerar los nodos: por ejemplo, los de profundidad h serían $\frac{4^h-1}{3}+j$ $(1 \leq j \leq 4^h)$.

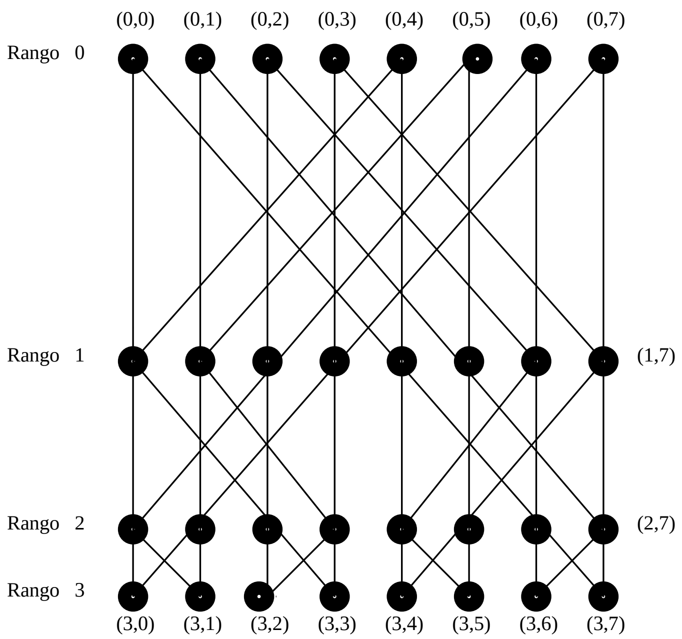
Ejercicio: Diseñar un algoritmo **SIMD–P**, eficiente, para hallar la suma de $n = 4^r$ números enteros.

Mariposa (B)

Mariposa de rango $r+1$: modelo **SIMD** tal que

- Consta de $(r+1) \cdot 2^r$ nodos organizados en $r+1$ filas (*rangos*) etiquetadas de 0 a r.
- Cada nodo tiene asociado un índice (i,j) con $0 \leq i \leq r$ y $0 \leq j < 2^r$.
- El nodo (i,j), $i > 0$, está conectado con $(i-1,j)$ e $(i-1,m)$, siendo m el número obtenido permutando 0 por 1 en el i–ésimo dígito de la representación binaria de j (con r dígitos)

Ejemplo: **SIMD–B** de rango 4.



Consideraciones:

- Un hipercubo de dimensión r se puede obtener a partir de una mariposa de rango $r+1$, colapsando sus columnas a un único nodo.

- Un *hipercubo de ciclos* de dimensión r es una mariposa de rango $r+1$ en donde los nodos de rango 0 están identificados con los de rango r. Es decir:

  ⋆ Consta de $r \cdot 2^r$ nodos, P_{ij} con $1 \leq i \leq r$ y $0 \leq j \leq 2^r - 1$.

  ⋆ P_{ij} está conectado con P_{lm} sii $(i \neq l \wedge j = m) \vee (i = l \wedge m$ resulta al permutar 0 por 1 en el i–ésimo dígito de la representación binaria de j, usando r dígitos).

Arbol (TCC)

Arbol de profundidad r: Modelo **SIMD** en donde la red de comunicaciones es un árbol binario completo con $2^r - 1$ nodos, enumerados de la manera usual.

Cada nodo representa un procesador.

La raíz y las hojas son los únicos procesadores con comunicación al exterior.

Ejemplo: **SIMD–TCC** de profundidad 4 (con $2^4 - 1$ procesadores).