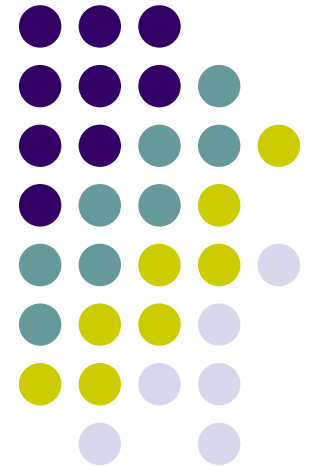


# Ordenación en Paralelo, Computación distribuida

- Ordenación en paralelo mediante fusión.
- Comentarios acerca de p-ram EREW y p-ram CRCW.
  - Cómputos distribuidos.



# Ordenación en Paralelo



- Se mostrará un algoritmo de ordenación paralela (*Cole's Parallel Merge Sort Algorithm*) que se puede ejecutar en una  $p$ -ram CREW, este algoritmo ordena  $n$  elementos en un tiempo que está en  $\Theta(\log n)$ , empleando un número de procesadores que está en  $\Theta(n)$ . Por lo que el trabajo total está en  $\Theta(n \log n)$ . Este algoritmo es óptimo en cuanto a la ordenación por comparación.

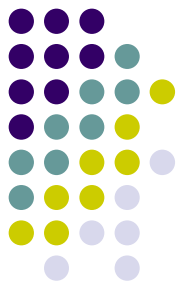
# Ordenación en Paralelo



## Suposiciones:

- Se asume que los  $n$  elementos sean distintos.
- Estos elementos se encuentran distribuidos en las hojas del árbol binario.
- $n$  es potencia de 2.
- El objetivo es ordenarlos de manera ascendente.

# Ordenación en Paralelo



- El algoritmo de Cole es una ordenación por fusión basada en árboles completos con  $n$  hojas. El cálculo progresa ascendiendo por el árbol, nivel por nivel, desde las hojas hasta la raíz. La idea de Cole es suministrar información adicional acerca de dos sucesiones ordenadas para que éstas pudiesen ser fusionadas en tiempo constante, por lo que el algoritmo global se ejecutaría en un tiempo de  $O(\log n)$ , puesto que existen  $\lg n$  niveles de nodos internos en el árbol.



# Ordenación en Paralelo

- A medida que progresa el algoritmo, se almacenan diferentes subconjuntos de estos  $n$  elementos por orden ascendente en vectores ordenados.

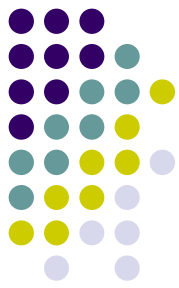
## Conceptos básicos

- Rango.
- Rango cruzado.
- Cubrimiento.

# Ordenación en Paralelo



- La idea clave es fusionar dos vectores en un tiempo  $O(1)$ . Supongamos que tenemos tres vectores ordenados ***J***, ***K***, ***L***; en donde asumimos que ***L*** cubre tanto a ***J*** como a ***K***.



# Ordenación en Paralelo

- Como  $L$  cubre a  $J$ , todo intervalo inducido por los elementos consecutivos de  $L$ , incluyendo los inducidos por los elementos invisibles, contienen como **máximo** tres elementos de  $J$ ; lo mismo se puede decir para  $K$ . Al conocer los rangos cruzados de estos elementos, es fácil determinar que elementos se encuentran en cada intervalo.



# Ordenación en Paralelo

- Así, podemos concluir que si podemos realizar la fusión en tiempo  $O(1)$  puesto que como máximo tendremos 6 elementos que fusionar, siempre y cuando contemos con la cantidad suficiente de procesadores, lo único que resta por hacer es concatenar los resultados de fusionar cada intervalo por separado. A este tipo de fusión se le llama *fusión con ayuda*, puesto que ***J*** y ***K*** se fusionan con ayuda de ***L***.



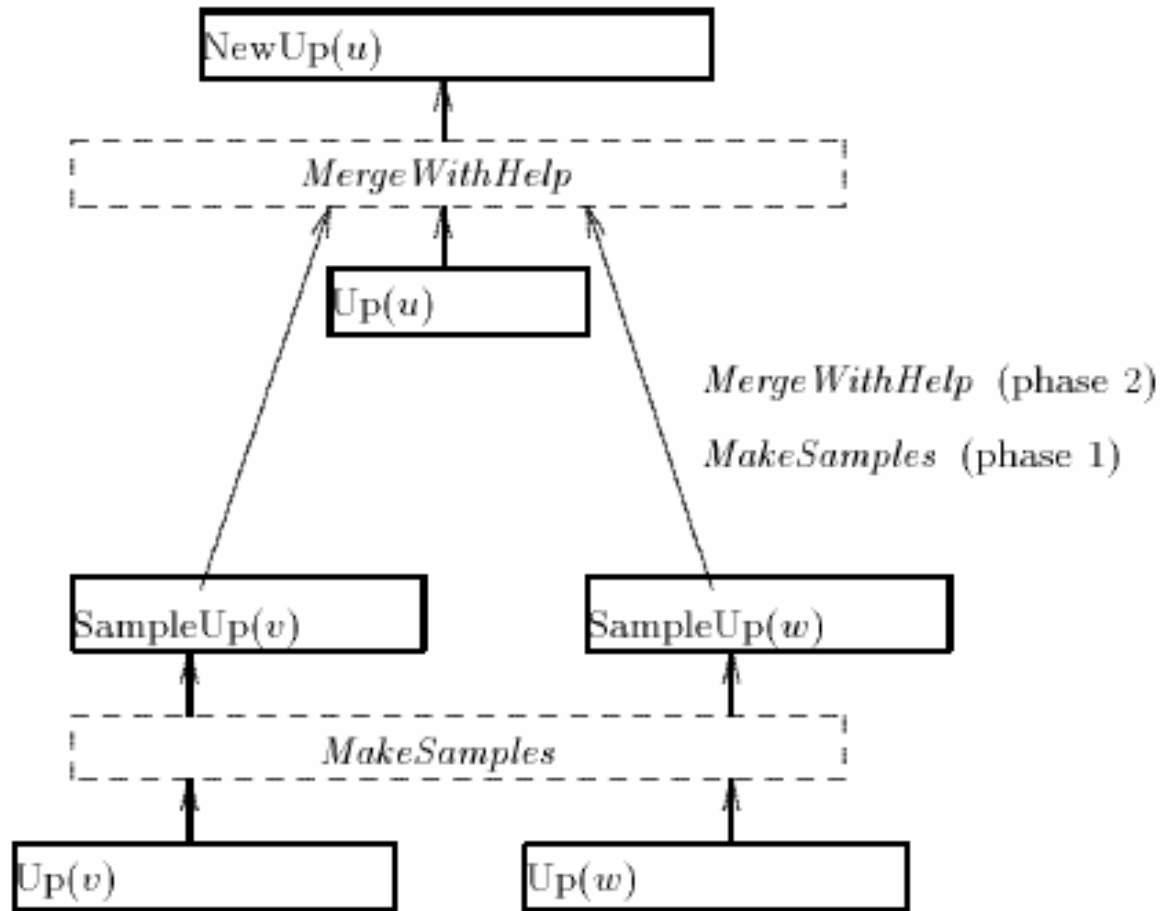
# Ordenación en Paralelo

Cole hace dos observaciones importantes:

- Fusión en tiempo constante  $O(1)$ .
- La fusión de los diferentes niveles del árbol pueden hacerse por tuberías (pipelines), esto es posible puesto que las fusiones parciales hechas en el nivel  $I$  del árbol pueden utilizarse para proporcionar muestras de tamaño apropiado para combinarse en el nivel siguiente sobre  $I$  sin perder la propiedad de tiempo de fusión constante.



# Ordenación en Paralelo

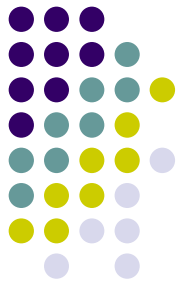




# Ordenación en Paralelo

- Durante el progreso del algoritmo, cada nodo  $u$  es almacenado en el vector  $Up(u)$ . La meta de cada nodo  $u$  es hacer de  $Up(u)$  una lista ordenada que contiene los elementos que inicialmente se encontraban en las hojas del subárbol cuya raíz es  $u$ . Cada etapa del procedimiento de fusión de Cole consiste en dos fases. En la fase 1, el vector  $Up(u)$  se muestrea de manera sistemática para producir el vector  $SampleUp(u)$ . En la fase 2, las dos muestras  $SampleUp(v)$  y  $SampleUp(w)$  (desde los dos hijos del nodo  $u$ ) se fusionan, originando una nueva secuencia  $NewUp(u)$  con la ayuda del vector  $Up(u)$ .

# Comentarios acerca de las p-ram EREW y CRCW



Antes de continuar definamos ciertos términos:

- **p-ram**: Máquina paralela de acceso aleatorio, en donde se supone que existen procesadores secuenciales que comparten una memoria global.
- **EREW**: Lectura Exclusiva, Escritura Exclusiva.
- **CREW**: Lectura Concurrente, Escritura Exclusiva.
- **CRCW**: Lectura Concurrente, Escritura Concurrente.

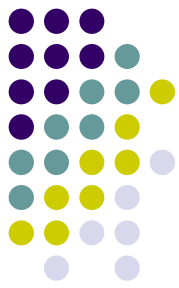
# Comparativa entre: p-ram CREW y p-ram EREW



Considere el siguiente problema:

- Un árbol binario contiene  $n$  nodos, y todo nodo  $i$  salvo la raíz está enlazado con su antecesor mediante un puntero  $p[i]$ . En la raíz, el puntero  $p$  tiene el valor especial **nil**. Todo nodo  $i$  posee además un segundo puntero  $r[i]$ . Se desea fijar el valor de  $r$  en todos los nodos de tal manera que apunte a la raíz de árbol.

# Comparativa entre: p-ram CREW y p-ram EREW



03/06/2005

buscarRaiz( Arbol T )

pre: { T es un árbol binario con n nodos }

**(( r[i] = Si( p[j] = nil ) entonces j  
fsi ) j = 1,n,1 ) i = 1,n,1**

**i,j** forman el par ordenado de nodos del árbol.  
**r[i]** = puntero a la raíz del árbol del nodo **i**.  
**p[j]** = puntero al antecesor en el árbol del nodo **i**.

**Tiempos de ejecución:**

**p-ram CREW:**  $O(1)$  empleando  $n^2$  procesadores.

**p-ram EREW:**  $\Omega(\log n)$ .

# Comparativa entre: p-ram CREW y p-ram CRCW



Considere el siguiente problema:

- Hallar el máximo de  $n$  números que están almacenados en un vector  $\mathbf{A}[1..n]$ .

# Comparativa entre: p-ram CREW y p-ram CRCW



03/06/2005

hallarMax( Entero A[1..n] ):Entero

pos: { Regresa el valor máximo del vector A[1..n]

$( M[i] = verdadero ) i = 1, n, 1$   
 $(( M[i] = \mathbf{Si}( A[i] < A[j] ) \mathbf{entonces falso} ) j = 1, n, 1 ) i = 1, n, 1$   
 $( max = \mathbf{Si}( M[i] ) \mathbf{entonces} A[i] ) i = 1, n, 1$   
**devolver** max

**M[1..n]** es un vector de valores booleanos.  
**max** es la variable que contendrá el valor máximo encontrado en el vector **A[1..n]**

## Tiempos de ejecución:

**p-ram CRCW:**  $O(1)$  haciendo uso de  $n^2$  procesadores

**p-ram CREW:**  $\Omega(\log n)$

# Algunas consideraciones sobre los p-ram CRCW



- Posibilidad de escritura simultánea en una misma localidad.
- Posibilidad de lectura de una localidad que está siendo escrita en ese mismo instante.

# Computación Distribuida



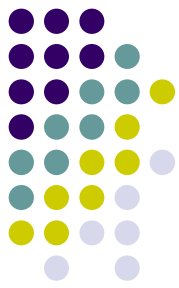
- Anteriormente se habían explicado modelos de *instrucción única y múltiples flujos de datos* (**SIMD**), en donde todos los procesadores ejecutan el mismo programa, ya que comparten una memoria global, en este modelo cabe la posibilidad de que los procesadores manejen distintos datos.

# Computación Distribuida



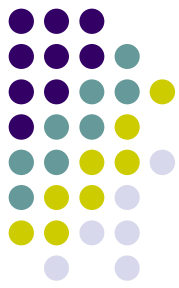
- En el modelo de *instrucción única y múltiples flujos de datos* (**SIMD**) se supone que el acceso a memoria (ya sea para *leer* o *escribir*) se puede hacer en un tiempo constante, independientemente del número de procesadores que se hayan utilizado, lo anterior en la práctica no es cierto, dado que no es factible proporcionar accesos directos (en *hardware*) desde todos los procesadores hasta todas las posiciones de memoria, el tiempo medio necesario para efectuar un acceso a memoria en un sistema real crece a medida que asciende la cantidad de procesadores utilizados.

# Computación Distribuida



- La computación distribuida se basa en el modelo *múltiples instrucciones y múltiples flujos de datos* (**MIMD**), esto significa que cada procesador puede operar sus propios datos, bajo su capacidad o velocidad y empleando su propio programa. Cuando un procesador obtenga valores cruciales procederá a enviar mensajes a los demás procesadores involucrados en el proceso, de igual manera, este procesador debe ser capaz de recibir mensajes de otros procesadores.

# Computación Distribuida



- En el modelo **MIMD** (al contrario que el modelo **SIMD**) no es realmente necesario que todos los procesadores involucrados en la ejecución del algoritmo paralelo se encuentren en un mismo lugar o en un mismo dispositivo físico. De hecho, si los mensajes se intercambian con frecuencia relativamente escasa, los procesadores pueden ubicarse en cualquier parte del mundo, los mensajes cruciales pueden ser enviados por correo electrónico.

# Computación Distribuida

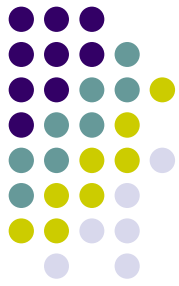


- Como ejemplo de aplicación de la **Computación Distribuida** podemos hacer mención del algoritmo de **Las Vegas** para factorizar números muy grandes, este algoritmo depende esencialmente en hallar una descomposición única de  $n$  como un producto de factores primos, en este caso  $n$  debe ser compuesto, ya que si  $n$  es primo hemos terminado, estos candidatos se seleccionan de manera aleatoria.

# Computación Distribuida



- El mayor número compuesto que se ha factorizado (hasta el momento) es de **129** dígitos, se necesitaron ocho (**8**) meses de cálculo en más de **600** máquinas de todo el mundo. Se estima que habrían sido necesarios **5.000** años de cálculo ininterrumpido si se hubiese empleado una única estación de trabajo capaz de efectuar **un millón de instrucciones por segundo**. Es de hacer notar, que cuando se planteo el desafío (en 1.977), se estimó que la computadora más rápida disponible en aquel entonces, ejecutando el mejor algoritmo para esa época, habría finalizado el cálculo al cabo de ¡dos millones de veces la edad del Universo!.



# ¡Muchas Gracias!