Parallel random-access machine (Todos los procesadores trabajan sincronizados con un mismo reloj):

EREW : Exclusive Reading Exclusing Writing

CREW : Concurrent Reading Exclusing Writing

CRCW : Concurrent Reading Concurrent Writing

Lema de Brent (SOLO PARA EREW):

> Si existe un algoritmo EREW con *T*(*n*,*p*)=O(*t*), tal que *W*(*n*)=*s* (trabajo total), entonces existe un algoritmo EREW con *T*(*n*,*s*/*t*)=O(*t*)

T(n) : Tiempo óptimo del algoritmo con un procesador y un input de tamaño n .

W(n) : Trabajo que realiza el algoritmo.

T(n,p) : Tiempo esperado del algoritmo con p procesadores y un input de tamaño n.

S(n,p) : Que tan mejor se hace con p procesadores comparado con 1 procesador. $S(n,p) = \frac{T(n,1)}{T(n,p)}$

E(n,p) : Tasa de uso de los procesadores $E(n,p) = \frac{S(n,p)}{p} = \frac{T(n,1)}{p \cdot T(n,p)}$

Speedup óptimo => $p = \frac{T(n,1)}{T(n,p)}$

Algoritmos para encontrar el máximo:

EREW : Haciendo un torneo con n/logn procesadores -> t() = 0(logn), E=O(1)

CRCW : T= O(log log n) E= O(1/log log n) Se dividen los números en grupos y se encuentra en dos mas el máximo de los grupos.

parallel prefix T(n,n) = log n.

Se toma una operación binaria asociativa.

T(n,1) = O(n) -> hacerlo secuencial

CREW -> $T(n,n) = O(\log n),\ E(n,n) = O\left(\frac{1}{\log n}\right)$

EREW -> $T\left(n, \frac{n}{\log n}\right) = O(\log n),\ E\left(n, \frac{n}{\log n}\right) = O(1)\quad W(n) = W\left(\frac{n}{2}\right) + n - 1, W(2) = 1 \Rightarrow W(n) = O(n)$

Odd-even sort

$E(n,n) = \frac{n \log n}{n \cdot n} = O\left(\frac{\log n}{n}\right)\quad T(n,n) = O(n),\ W(n) = O(n^2)$

Merge-Sort paralelo

$T(n) = T(n/2) + \log n = O(\log^2 n)$

Cantidad de procesadores : $P(n) = 2P(n/2) + n \log n = O(n \log^2 n)$ $E(n) = \frac{n \log n}{n \log^2 n \cdot \log^2 n} = O\left(\frac{1}{\log^3 n}\right)$

***rank*(*B*:*A*)**: toma tiempo *O*(log log *m*)

Algoritmos Aleatorizados:

Algoritmo Montecarlo: Tiene una cierta probabilidad de equivocarse, pero se ejecuta en tiempo polinomial.

Algoritmo Las Vegas : Entregan la respuesta correcta pero su tiempo de ejecución no está garantizado.

Hashing universal y Hashing perfecto

Hashing universal:

Escoger función de hash aleatoriamente, independiente de las llaves. Con alta probabilidad tendremos un buen tiempo promedio.

H colección finita de funciones de hash, decimos que es universal si la cantidad de funciones que cumple que tienen igual imagen para un mismo input es a lo mas |H|/m. P(colisión)<=1/m

Se puede garantizar que cada secuencia de operaciones se puede procesar con un buen tiempo esperado de ejecución

$h_{a,b} = ((ak + b) \mod p) \mod m$

Hashing perefcto:

Para conjunto de llaves estatico. Hashing universal pero para las colisiones hay una función de hash especial .

Buena cota para el peor caso.

Algoritmos aproximados:

Encontrar soluciones casi optimas.

$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$ Max: $0 < C \leq C^*$, razón C^*/C Min: $0 < C^* \leq C$, razón C/C^*

Subset-sum:

Encontrar un subconjunto S cuya una sea lo mayor posible pero no mayor que t.

```
Trim(L,δ)
1. m <- |L| // L = [y1, y2, …, ym]
2. L' <- [y1]
3. last <- y1
4. for i <- 2 to m
5.     if yi > last*(1+δ)
6.         agregar yi al final de L'
7.         last <- yi
8. return L'
```

Tiempo teta(m).