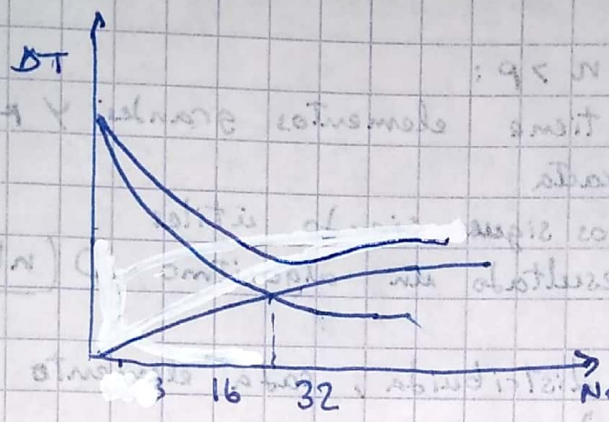
Examen Parcial

① Los puntos de cruce representan el número óptimo de nodos por cada simulación. Viendo la gráfica, aproximadamente 6 nodos para 8K cuerpos y 64 para un millón de cuerpos.
Para N = 2500K el punto de cruce se mueve hacia abajo y a la izquierda con respecto a la segunda figura, pero a la derecha y hacia arriba respecto al de 8K.



Esbozando el gráfico, la cantidad de nodos óptimos estaría en un rango [16, 32] aprox..

② Los FLOPS teóricos serían

$$S \approx \frac{500 \cdot N^{2.31}}{10^{-9} \cdot N^{2.31}/5000 + \log 5000} \quad \text{..(α)}$$

Se puede construir con esta ecuación una gráfica FLOPS vs np y también se puede comparar con la gráfica (S) speedup vs np.
La gráfica S vs np es mucho más fácil para deducir la eficiencia $E = S/np$ y la escalabilidad.

Si $np \ll S$, en $E = \frac{S}{np}$, decimos que escala en n, y es un escalamiento débil, conocido como weak scaling.

analizando (α) vemos que S es mucho mayor que que np. ⇒ hay weak scaling

③ * Mirando el nivel de recursividad:
- 2 broadcast
- 1 suma de prefijos

$\rightarrow O(T_{start} \log p)$

* Profundidad de recursividad esperada: $O(\log p)$

Tiempo total esperado: $O(T_{start} \log^2 p)$

Ahora analizando para $n > p$:
- Cada PE genalmente tiene elementos grandes y pequeños
- La división no es exacta
- La suma de prefijos siguen siendo útiles
- PRAM da como resultado un algoritmo $O\left(\frac{n \log n}{p} + \log^2 p\right)$
- En caso de memoria distribuida, cada elemento se envía $O(\log p)$ veces

$\Rightarrow O\left(\frac{n}{p}(\log n + T_{byte} \log p) + T_{start} \log^2 p\right)$

(9)

```
init = Time()
for i = a to n-1 pardo:
        indexRandom = Rand()
        Enviar Mensaje (Pi , P indexRandom)   ...  (1)
        Enviar Mensaje (P indexRandom , Pi)   ...  (2)

        index Random = Rand()
        Enviar Mensaje (Pi , Pindex Random)
end = Time()

Determinar tiempo (& P0 , end, init)
```

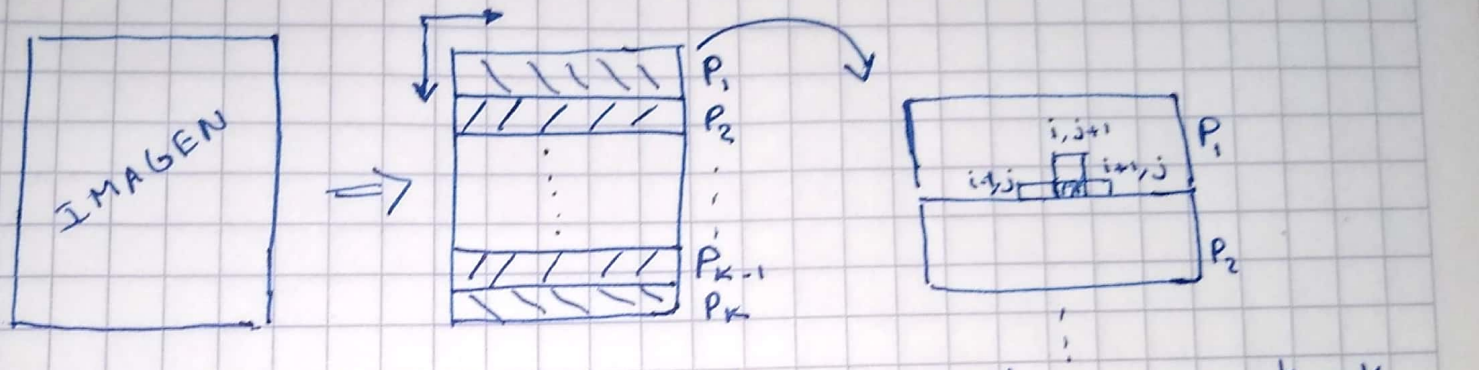
La función 'Determinar tiempo' solo resta end - init en el proceso P_i. Note que la línea (1) y (2) envían 2np mensajes (tomando en cuenta el bucle for).

Determinar tiempo (& P_0, end, init)

La función 'Determinar tiempo' solo resta end - init en el proceso P_i. Note que la línea (1) y (2) envían 2np mensajes (tomando en cuenta el bucle for).

(5)

Como cada proceso recibe n/p pixeles, supongamos que la imagen se divide en K procesos:



Con n = # pixeles = ancho x alto = a x h con $\frac{h}{p} = K$

Un código aprox. podria ser el siguiente.

-> inicialización U } w_1

-> for i = 1 to h/p hacer

for j = 1 to a do

U(i,j) = (U(i-1, j) + U(i, j+1) + U(i+1, j))/3 } w_2

// comunicación al proceso vecino P_{i+1} de los nuevos
// valores de U cercanos al proceso P_{i+1} de parte del
// proceso P_i

$$\Rightarrow W = w_1 + w_2 = O\left(\frac{h}{p} \times a \times \frac{1}{p}\right) = O\left(\frac{n}{p^2}\right)$$