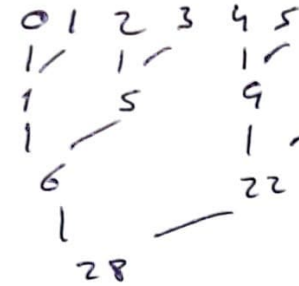
Se quiere implementar en CUDA una función de reducción para sumar N números.

Suponiendo N = 8192 elementos



• Max 512 t/B • Max 8 B • 16 SM → 24 warps

@ Indique una planificación de bloques y threads para realizar esta reducción, ¿cuántos warps se van a utilizar? ¿cuántos warps por SM se están planificando?

24 warps

$512\ T/B \cdot \frac{1W}{32T} = 16\ W/B$ máx $\quad 16\ W/B \cdot 8\ B_{max} = 128$

$\frac{24\ warps}{1\ SM} \cdot \frac{32\ T}{1\ warp} = 768\ T/SM \quad \frac{8192\ T}{16\ SM} = 512\ T/SM$

$\frac{512\ T}{8\ B} = 64\ T/B \rightarrow 2\ W/B$

Se emplean 16 warps, 8 bloques, 64 T/B

⑥ Se pretende realizar cambios optimizando la planificación de warps, ¿qué cambios realizaría para conseguirlo? ¿cuál es el menor número de SM para esa planificación?

Podríamos dinámicamente unir los threads ejecutando instrucciones en los warps para reducir la divergencia en ejecución

$768\ T/SM \quad \frac{8192\ T}{768\ T/SM} = 10'6 \rightarrow 11\ SM$ mínimo

⑦ Explique el esquema de la reducción propuesta y comportamiento en eficiencia

```
__shared__ float pSum[];
unsigned int t = threadIdx.x
for (unsigned int stride = 1; stride < blockDim.x; stride *= 2)
    __syncthreads();
    if (t % (2*stride) == 0)
        pSum[t] += pSum[t + stride];
```

Stride es la distancia entre elementos a sumar, primero 1, luego 2, 4, 8... Cuando te encuentras en un elemento divisible entre el doble de distancia → están en el 1º de los elementos a sumar aunque pSum de la par del elemento le sumará el de la par + stride y así hasta que el stride supere blockDim (han sumado todos los elementos del bloque)

1 procesador escalar → n sumas $\quad T(n) = \log(n) \quad E = \frac{T(1)}{T(n)} = \frac{n}{\log(n)}$

Proponga un esquema alternativo de reducción

```
__shared__ float pSum[];
t = threadIdx.x
for (stride = blockDim.x/2; stride > 1; stride >>= 1) {
    __syncthreads()
    if (t < stride)
        pSum[t] += pSum[t+stride]
}
```

Diferencia del anterior, en este los threads activos siempre están juntos lo que disminuye la divergencia en ejecución mejorando la eficiencia del programa

De esta manera la mitad continuan a la siguiente iteración como parabas antes pero esta vez todos juntos reparando activos de inactivos