1. HISTOGRAM

slika velikosti WIDTH × HEIGHT

```
scrypka:    for ( y = 0 ... HEIGHT)
                for (x = 0 ... WIDTH)
                    Hist [ img [ y x WIDTH + x ] ] ++;
```

to leni problem so paralelizacijo

IDEJA ZA PARALELIZACIJO:

→ zaženemo toliko niti, kolikor je pixlov

```
VSAKA NIT:    hist [ img [ t.y x WIDTH + t.x ] ]++ ;
```

več niti skuša updatat isti element histograma

PROBLEM: Dve sosednji niti v snopu ne dobijata do dveh sosednjih elementov histograma

⇓

NI ZDRUŽEVANJA (COALESCING) POT. DOSTOPOV

TREBA SE JE LOTIT PARALELNO: ⇒

(+) NITI V ISTEM BLOKU RAČUNATA SVOJ LASTEN HISTOGRAM → šele ko se dostop do nekega elementa histograma tepe le toliko niti, kolikor jih je v bloku

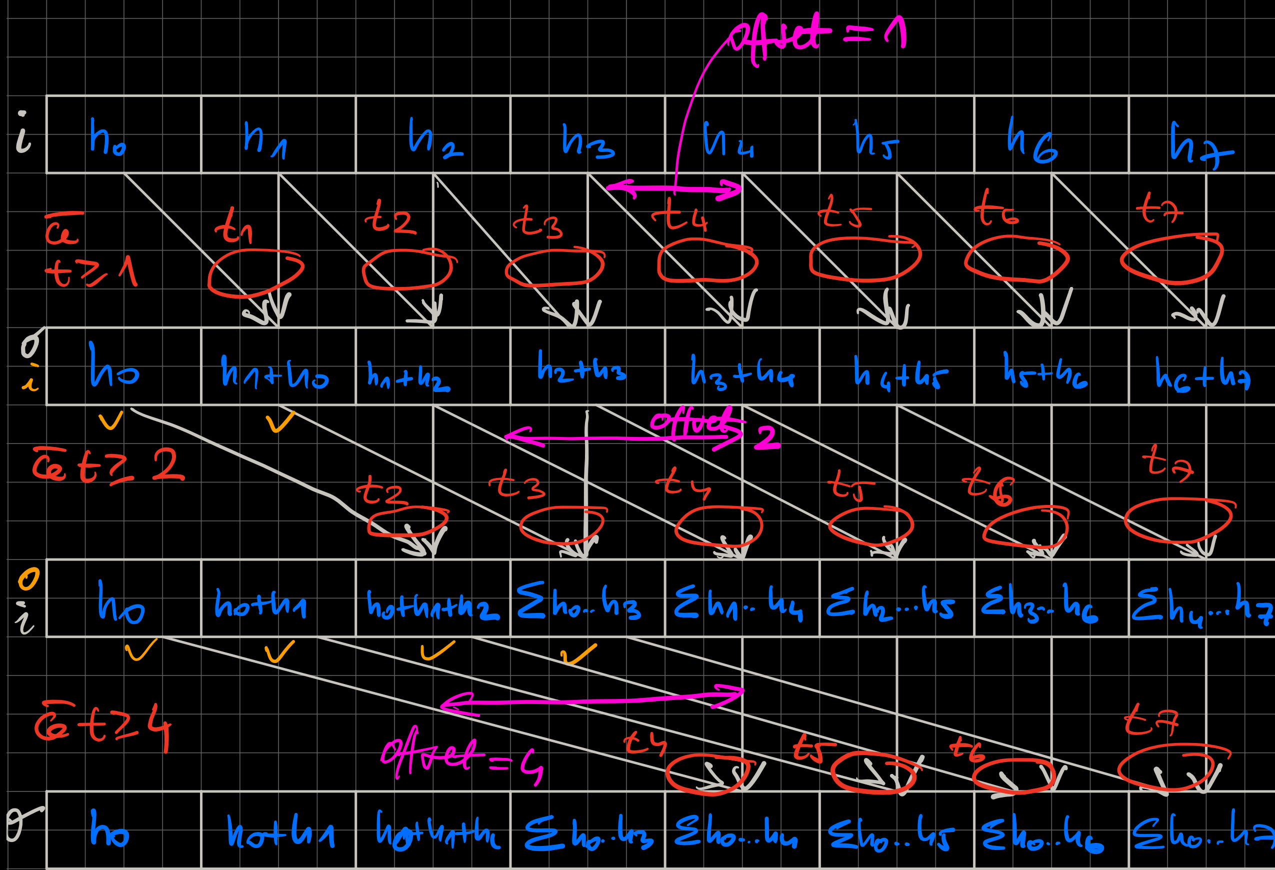
(+) TA HISTOGRAM, KI PRIPADA NEKEMU BLOKU HRANIMO V LOKALNEM POMNILNIKU → veliko hitrejši dostopi

⊖ NA KONCU IMAMO TOLIKO "DELNIH" HISTOGRAMOV, KOLIKO SE BLOKOV

↓

Easy s sestetjem vseh histogramov!

```
// ta kernel izvede toliko niti koliko je pixlov, in so organizirane v bloke velikosti 256 (16x16)
__global__ void histogram_kernel(unsigned char* image, int width, int height, unsigned int *histogram) {

    // izracunaj svoj globalni x, y:
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    // izracunaj svoj lokalni x, y:
    int lx = threadIdx.x;
    int ly = threadIdx.y;

    // v lokalnem spominu rezervirajmo porsotor za lokalni histogram:
    __shared__ unsigned int localHistogram[GRAYLEVELS];

    // vsaka nit pobriše svoj beam hitograma:
    localHistogram[blockDim.x * ly + lx] = 0;
    __syncthreads();


    if (x < width && y < height) {
        // če sem nbit v sliki, popravi beam v histogramu, ki ga kaže moja barva:
        atomicAdd( &localHistogram[image[y*width + x]], 1);
    }
    __syncthreads();
    // sedaj imamo izračunan lokalni histogram


    // sedaj vsaka nit "nese"svoj beam v glavni pomnilnik. Ker sosednje niti nesejo sosednje bime na sosednje pom. lokacvije v GP,
    // lahko združujemo pomnilniške dostope (= mewmory coalescing):
    atomicAdd( &(histogram[blockDim.x * ly + lx]), localHistogram[blockDim.x * ly + lx] );
}
```



Regularno:

index koraka / zanke: 1, 2, 4, ...

↳ for (i = 1 ... ; i = i*2)

v vsakem koraku:

→ delajo vse niti katerih index ≥ indeksu zanke

→ offset = index zanke

število korakov = $\log_2 8 = 3$

↳ za naš histogram: $\log_2 256 = 8$

Rešitev: delo le 1 bloka z 256 niti !!

1. Vse niti preberejo globalni hist v lokalnega, tako, da bodo delale z lok. spominom

2. Vsaka nit:

```
for (offset = 1 ... 256 ; ofset = offset×2 )
{ if tid ≥ offset
  hist_out[tid] = hist_in[tid] +
                  hist_in[tid - d] ;
}
```

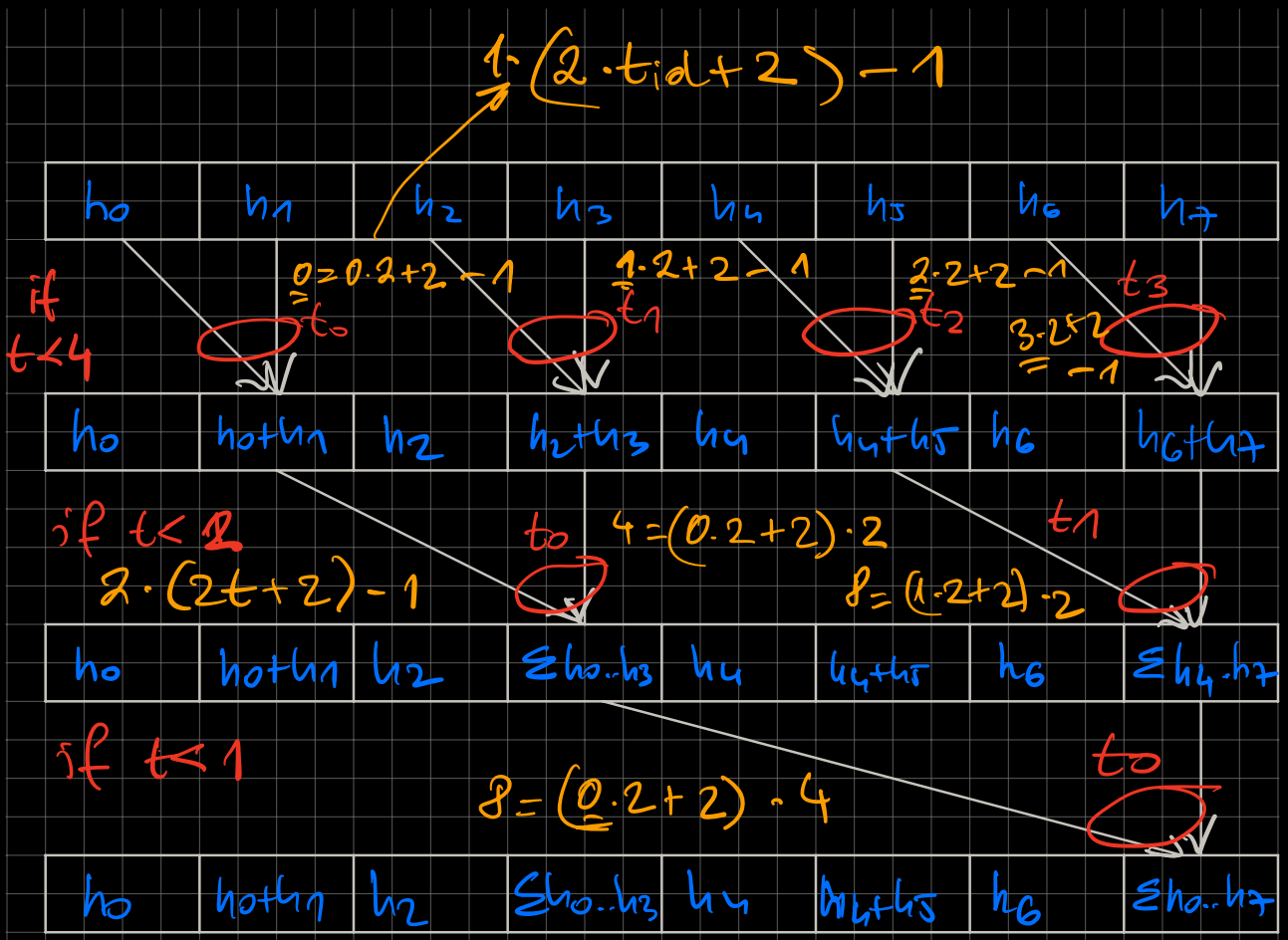
3. Vse prepiši v glob. pomnilnik

Koliko seštevanj?

off = 1	255	$(n-1)+(n-2)+(n-4)+\dots+(n-128)$
off = 2:	254	$(n-2^0)+(n-2^1)+(n-2^2)+\dots+(n-2^7)$
off = 4:	252	$(2^8-2^0)+(2^8-2^1)+(2^8-2^2)+\dots+(2^8-2^7)$
= 8:	248	$8\cdot 2^8-(2^0+2^1+\dots+2^7)$
= 16:	240	$= n(\log n)+1$
= 32:	224	$O(n \log n)$
= 64:	192	
= 128:	128	
	Σ 1794	

```cpp
/*
    Kernel za računanje CDF. Izvede ga le en blok z 256 nitmi.
    Potrebnih seštevanj = 1794
    Zahtevnost: O(n log n)
    Kernel izvede le en blok niti
*/
__global__ void CDF_kernel(unsigned int* histogram, unsigned int* cdf)
{
    __shared__ unsigned int temp[GRAYLEVELS*2]; // potrebujem dvojni buffer, ker si niti v vsakem koraku prepisujejo elemente, ki jih berejo
    int tid = threadIdx.x;

    // Spremenljivki za preklapljanje med dvojnima bufferjema:
    int pout = 0, pin = 1;

    // Naložim histogram iz globalnega pomnilnika v spodnjo polovico lokalnega dvojnega bufferja
    temp[tid] = histogram[tid];

    __syncthreads();

    for( int offset = 1; offset < GRAYLEVELS; offset <<= 1 )
    {
        pout = 1 - pout; // swap double buffer indices
        pin = 1 - pout;
        if (tid >= offset)
            temp[pout*GRAYLEVELS+tid] = temp[pin*GRAYLEVELS+tid]+temp[pin*GRAYLEVELS+tid - offset];
        else
            temp[pout*GRAYLEVELS+tid] = temp[pin*GRAYLEVELS+tid];
        __syncthreads();
    }


    cdf[tid] = temp[pout*GRAYLEVELS+tid]; // prepišem izhodni del dvojnega bufferja v globalni pomnilnik
}
```

ta kernel izvede 1 blok z 256 niti !!



Upotovim:

korak 1: 2^0 nit sestejé elemente:

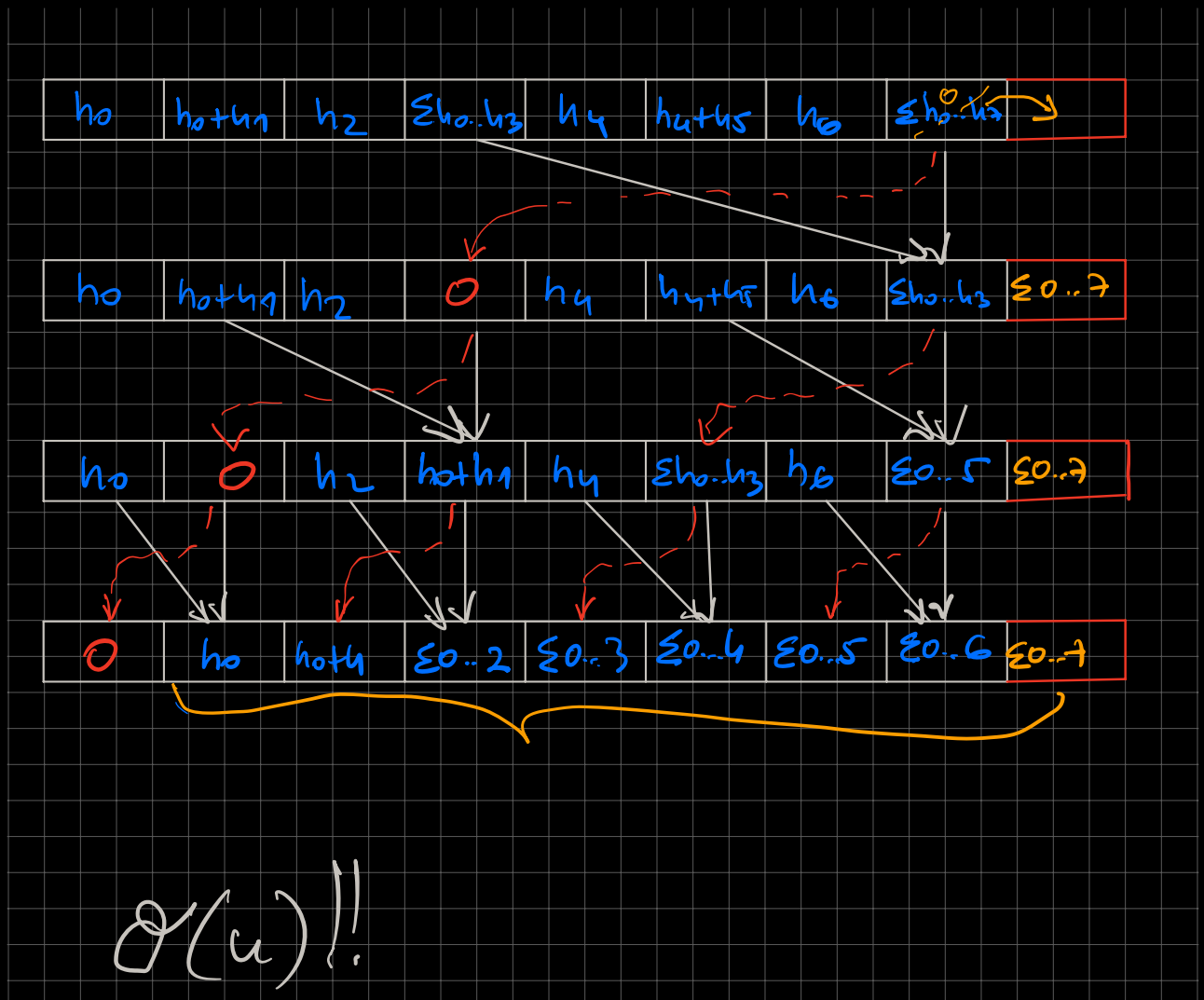
$$\underset{=1}{1}\cdot(t\cdot 2+2)-1 \;+$$
$$\underset{=1}{1}\cdot(t\cdot 2+1)-1$$

korak 2: 2^1 nit sestje elemente:

$$\underset{=2}{2}\cdot(t\cdot 2+2)-1 \;+$$
$$\underset{=2}{2}\cdot(t\cdot 2+1)-1$$

korak 3: 2^2 $\underset{=4}{4}\cdot(t\cdot 2+2)-1 + 4\cdot(t\cdot 2+1)-1$


```
/*
1. Up-sweep (reduction) phase:
   delajo niti: 0,1,2,...,127 | 0,1,2,...63 | ... | 0,1,2,3 | 0,1 | 0
   indeksiranje:
                2tid+2 so sodi elementi, 2tid+1 so lihi elementi polja
                offset * (2tid+2) in offset*(2tid+1) potem v vsakem koraku d izbere vsak drug element polja glede na prejsnji korak d
*/


for (size_t d = GRAYLEVELS/2; d > 0; d >>= 1)
{
    if (tid < d) {
        temp[offset*(2*tid+2)-1] += temp[offset*(2*tid+1)-1];
    }
    offset *= 2;
    __syncthreads();
}

// Shranimo zadnji element in vanj vpišemo 0
// down-sweep naredi exclusive scan, zato moram shraniti zadnji element
if (tid == 0) {
    temp[GRAYLEVELS] = temp[GRAYLEVELS-1]; // shranimo vsoto vseh elementov
    temp[GRAYLEVELS-1] = 0; // v zadnjega vpisemo 0
}
__syncthreads();
```



$\theta(n)$!!

```
// Shranimo zadnji element in vanj vpišemo 0
// down-sweep naredi exclusive scan, zato moram shraniti zadnji element
if (tid == 0) {
    temp[GRAYLEVELS] = temp[GRAYLEVELS-1]; // shranimo vsoto vseh elementov
    temp[GRAYLEVELS-1] = 0; // v zadnjega vpisemo 0
}
__syncthreads();

/*
2. Down-sweep (reduction) phase:
Delajo niti: 0 | 0,1 | 0,1,2,3 | ... | 0,1,2,3...63 | 0,1,2...127 | -> zato d narašča
indeksiranje:
                2tid+2 so sodi elementi, 2tid+1 so lihi elementi polja
                offset * (2tid+2) in offset*(2tid+1) potem v vsakem koraku d izbere vsak drug element polja glede na prejsnji korak d
*/

offset = GRAYLEVELS/2;
for (size_t d = 1; d <= GRAYLEVELS/2; d <<= 1)
{
    if (tid < d) {
        unsigned int t = temp[offset*(2*tid+1)-1];             // shrani levi element v temp
        temp[offset*(2*tid+1)-1] = temp[offset*(2*tid+2)-1];  // prenesi desnega v levi element
        temp[offset*(2*tid+2)-1] += t;                         // levemu prištej desnega
    }
    offset >>= 1;
    __syncthreads();
}
```

Nekdo me je prosil, naj objavim kodo za izpis vmesnih rezultatov:

```
/*****************************************************************
Izpis rezultatov in slik
*****************************************************************/


// Izračun histograma nove slike:
CalculateHistogram(imageOutGPU, width, height, new_histogram);

printf("Beam       Hist_CPU     Hist_GPU     CDF_CPU      CDF_GPU      newHist\n");
for (size_t i = 0; i < GRAYLEVELS; i++)
{
    printf("%3d      %6d      %6d       %6d       %6d         %6d \n", i, histogram_CPU[i], histogram[i], CDF_CPU[i], CDF[i], new_histogram[i]);
}

printf("\n\nSlika velikosti %d x %d\n", width, height);
printf("GPU time: %.4f ms\n", gpu_elapsedTime );
printf("CPU time: %.4f ms\n", cpu_elapsedTime);


stbi_write_jpg("slikaCPU.jpg", width, height, DESIRED_NCHANNELS, imageOut, 100);
stbi_write_jpg("slikaGPU.jpg", width, height, DESIRED_NCHANNELS, imageOutGPU, 100);
```