Gather - Scatter

Tran Giang Son, tran-giang.son@usth.edu.vn

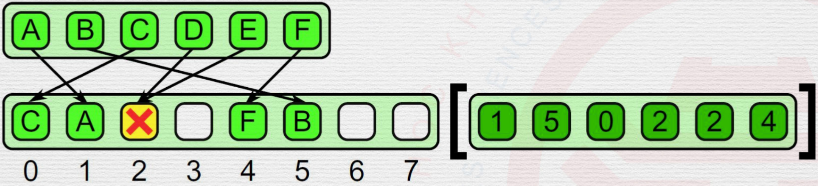
ICT Department, USTH

Scatter

What?

- “Sequential read, scattered random write”
- A relationship *from input* to output
- Smaller set of values is distributed (**written**) into another array
    - [Optional] with an index vector
- There may be gaps in the output array
- Highly memory intensive

What?



Why?

- Efficient memory access
  - Pre-cache contents
- AoS to SoA
- Building block of other algorithms
  - e.g. Radix sort

How?

In serial fashion

```
for (int i = 0; i < N; i++) {
    out[idx[i]] = in[i];
}
```

In parallel

```
    int tid = ...;
    out[idx[threadIdx.x]] = in[tid]; // lookup
    out[tidx * 3] = in[tid];         // strided write
}
```

Examples

- RGB to Gray

```
out[tid].x = out[tid].y = out[tid].z = gray;
```

- RGB to HSL

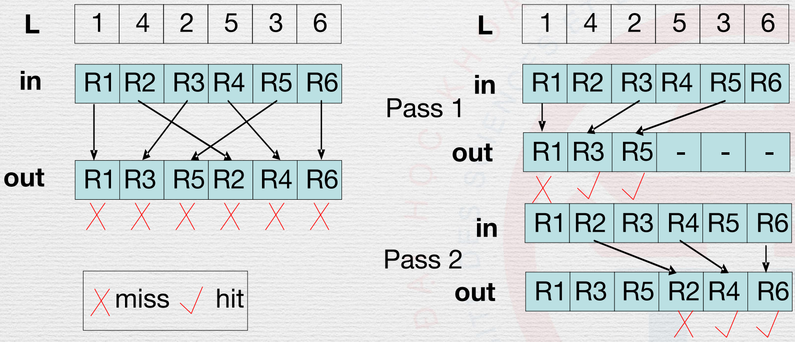
```
outh[tid] = h;
outs[tid] = s;
outl[tid] = l;
```

Optimization

- Exploit cache locality
- Multi-pass scatter [1]
  - n passes
  - Divide output to n areas
  - In i^{th} pass
    - Check destination
    - Write only if in i^{th} area

[1] Efficient Gather and Scatter Operations on Graphics Processors

Optimization



Labwork 8: Scatter

- To prepare for the next “Gather” labwork, we need to convert the input image:
  - From RGB to HSV, from AoS to SoA: to 3 different arrays (`H[]`, `S[]`, `V[]`)
  - Reversely, from HSV to RGB, from SoA to AoS
- Implement the two scatter 2D kernels `RGB2HSV()` and `HSV2RGB()`
- Test the two kernels for a sample image (convert to HSV and convert back to RGB), compare the output with the input image
- Write a report (in LaTeX)
  - Name it « Report.8.scatter.tex »
  - Explain how you implement the labworks
  - Explain and measure speedup, if you have performance optimizations
- Push the report and your code to your forked repository

Extra: HSV

- Hue, Saturation, Value
  - $H \in [0..360]$: “The color”. Red? Yellow? Cyan? Magenta?
  - $S \in [0..1]$: “The colorfulness”. Really cyan? Light yellow?
  - $V \in [0..1]$: “The brightness”. Dark cyan? Crimson?

Extra: RGB to HSV

- Preparation
  - Scale R, G, B to [0..255] to [0..1]
  - Find max and min among $R, G, B \in [0..1]$
  - $\Delta = max - min$
- Conversion

$$H = \begin{cases} 0^o & \Delta = 0 \\ 60^o \times (\frac{G-B}{\Delta} \bmod 6) & max = R \\ 60^o \times (\frac{B-R}{\Delta} + 2) & max = G \\ 60^o \times (\frac{R-G}{\Delta} + 4) & max = B \end{cases}$$

$$S = \begin{cases} 0 & max = 0 \\ \frac{\Delta}{max} & max \neq 0 \end{cases}$$

$$V = max$$

Extra: HSV to RGB

- Preparation

- $d = H/60$
- $hi = (int)d \bmod 6$
- $f = d - hi$
- $l = V \times (1 - S)$
- $m = V \times (1 - f \times S)$
- $n = V \times (1 - (1 - f) \times S)$

- Conversion

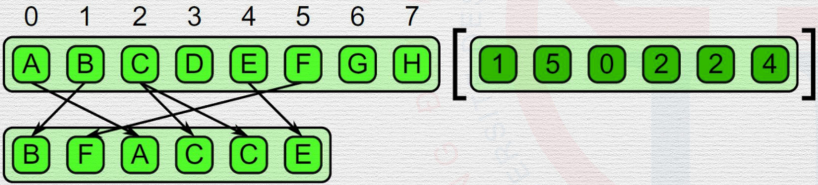
$$(R, G, B) = \begin{cases} (V, n, l) & 0^o \leq H < 60^o \\ (m, V, l) & 60^o \leq H < 120^o \\ (l, V, n) & 120^o \leq H < 180^o \\ (l, m, V) & 180^o \leq H < 240^o \\ (n, l, V) & 240^o \leq H < 300^o \\ (V, l, m) & 300^o \leq H < 360^o \end{cases}$$

- Scale R, G, B from $[0..1]$ to $[0..255]$

Gather

What?

- “Gathered random read, sequential write”
- A relationship *from output* to input
- Large set of values are **read** from an array
  - [Optional] with an index vector
- No gap in the output array
- Highly memory intensive



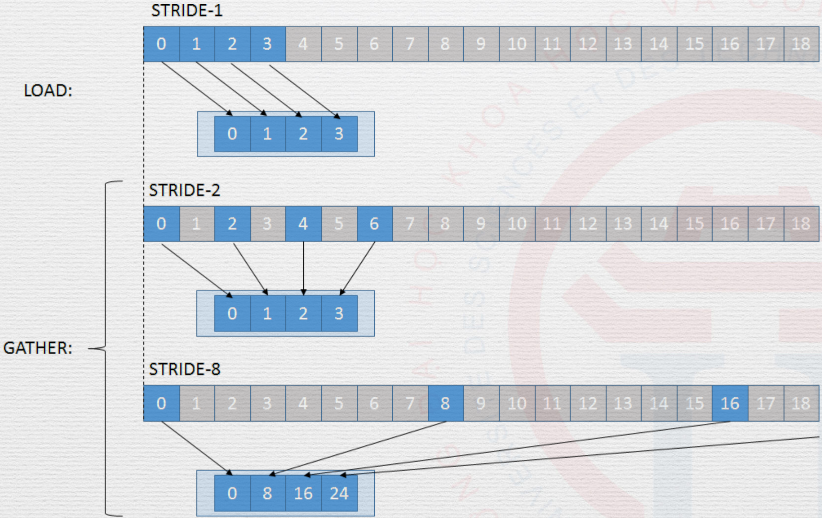
Why?

- SoA to AoS
  - Efficient memory access
- Building block of other algorithms
  - e.g. Radix sort

Why?

- Strided memory access
  - Memory fields accessed are equally distant
  - Called a stride
- Gathered $\Rightarrow$ contiguous access

Why?


STRIDE-1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
LOAD:
0 1 2 3
STRIDE-2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
0 1 2 3
GATHER:
STRIDE-8
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
0 8 16 24

How?

In serial fashion

```
for (int i = 0; i < N; i++) {
    out[i] = in[idx[i]];
}
```

In parallel

```
    int tid = ...;
    out[tid] = in[idx[threadIdx.x]]; // <-- lookup
    out[tid] = in[tid * 3];          // strided read
}
```

Example

- RGB to gray

```
char gray = (in[tid].x + in[tid].y + in[tid.z]) / 3;
```

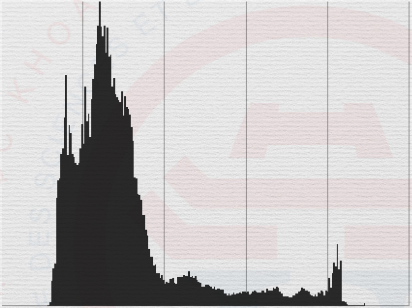
Labwork 9: Histogram Equalization

- Implement labwork 9a: Histogram
    - Calculate histogram of input **grayscale** image
- Implement labwork 9b: Histogram Equalization for **grayscale** image
    - Equalize the histogram for that input image
- Write a report (in LaTeX)
    - Name it « Report.9.gather.tex »
    - Explain how you implement the labworks
    - Explain and measure speedup, if you have performance optimizations
- Push the report and your code to your forked repository

Extra 9a: Histogram

- Graphical representation of the value distribution in a digital image
  - Divide range of values to certain ranges (“bins”)
  - Count number of value occurences for each “bin”
- For image: an array, each element i in the array counts the number of pixels having gray level i

Extra 9a: Histogram



Extra 9a: Histogram

In serial

```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        histo[input[y][x]]++;

    }
}
```

Extra 9a: Histogram

In parallel: 2 ways

- Atomic operations
  - Race condition due to parallelization
  - Not discussed this week. Next week with Prof. Lillian Aveneau.
- Local histogram
  - Each thread calculates a local histogram `lhisto[]` of a region in image (GATHER)
  - A sum is then combined for all regions (REDUCTION)

Extra 9b: Histogram Equalization

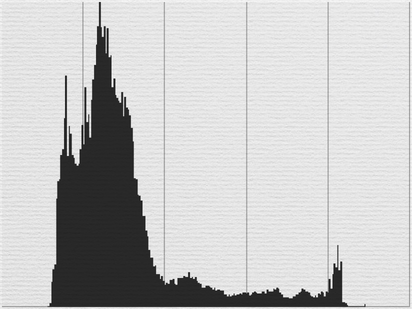
- Previously, grayscale stretch (LW 6)
  - Increases global contrast
  - Linearly calculates intensity of each pixel from [$min..max$] to [0..255]
- Histogram equalization
  - Increases global contrast
  - Recalculates intensity using normalization of cumulative distribution function

Extra 9b: Histogram Equalization

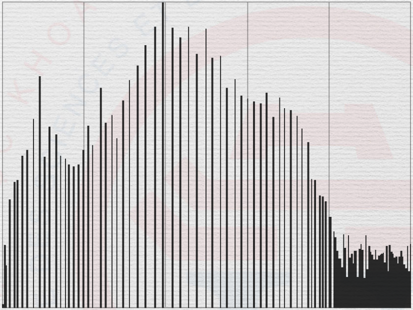
Original

Equalized

Extra 9b: Histogram Equalization



Original



Equalized

Extra 9b: Histogram Equalization

- Calculate histogram `histo[]` (LW8a, GATHER + REDUCE)
- Let n be number of total pixels in the image
- Calculate probability of given intensity j (MAP)

$$p_j = \frac{histo_j}{n}, \forall j \in [0..255]$$

Extra 9b: Histogram Equalization

- Cumulative distribution function (CDF) c is calculated as [2]

$$c_i = \sum_{j=0}^{i} p_j$$

- Linearly scale $c_i \in [0..1]$ to $h_i \in [0..255], \forall i \in [0..255]$ (MAP)
- Original intensity i is transformed to h_i (MAP)
- We should have histogram equalized as now.

[2] c_i in this case should be calculated using parallel SCAN, but you haven't learnt about it yet. Let's do this step in sequential fashion.

Final Labwork: Fine-art transformation

- Implement Kuwahara filter
- Write a report (in LaTeX)
  - Name it « Report.10.kuwahara.tex »
  - Explain how you implement the labworks
  - Explain and measure speedup, if you have performance optimizations
- Push the report and your code to your forked repository

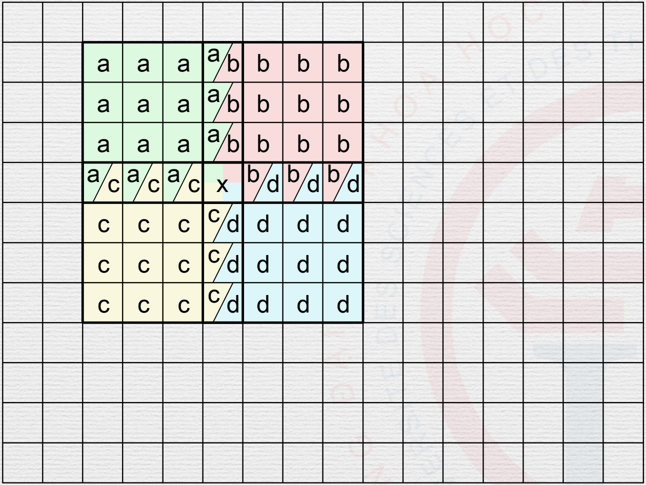
Final Labwork: Fine-art transformation

- Kuwahara filter [wikipedia]
  - Reduces noise
  - Keeps edge
  - Also produces oil effect
  - Requires a lot of computation!

Final Labwork: Fine-art transformation

- Parameter ω as window size
- Convert RGB to HSV (SCATTER)
- For each pixel $\Phi(i,j)$
  - Define use 4 windows $W^k, k \in [1..4]$ of size $(\omega+1) \times (\omega+1)$
  - $W_x^1 \in [i-\omega, i], W_y^1 \in [j-\omega, j]$
  - $W_x^2 \in [i, i+\omega], W_y^2 \in [j-\omega, j]$
  - $W_x^3 \in [i-\omega, i], W_y^3 \in [j, j+\omega]$
  - $W_x^4 \in [i, i+\omega], W_y^4 \in [j, j+\omega]$

Final Labwork: Fine-art transformation



Example $\omega = 3$

Final Labwork: Fine-art transformation

- Find $W_l, l \in [1..4]$ having lowest standard deviation of brightness.
  - Use V in HSV color space to calculate SD.
- Assign mean (R, G, B) value of this window $\mid W_l \mid_{RGB}$ as new color (REDUCE, MAP)

$$\Phi(i,j)_{RGB} = \mid W_l \mid_{RGB}$$

Good luck & Have fun