Histogram Equalization Performance and Power Consumption Comparison on Different Platforms

Ercüment Kaya
Emre Dinçer
Burak Topçu

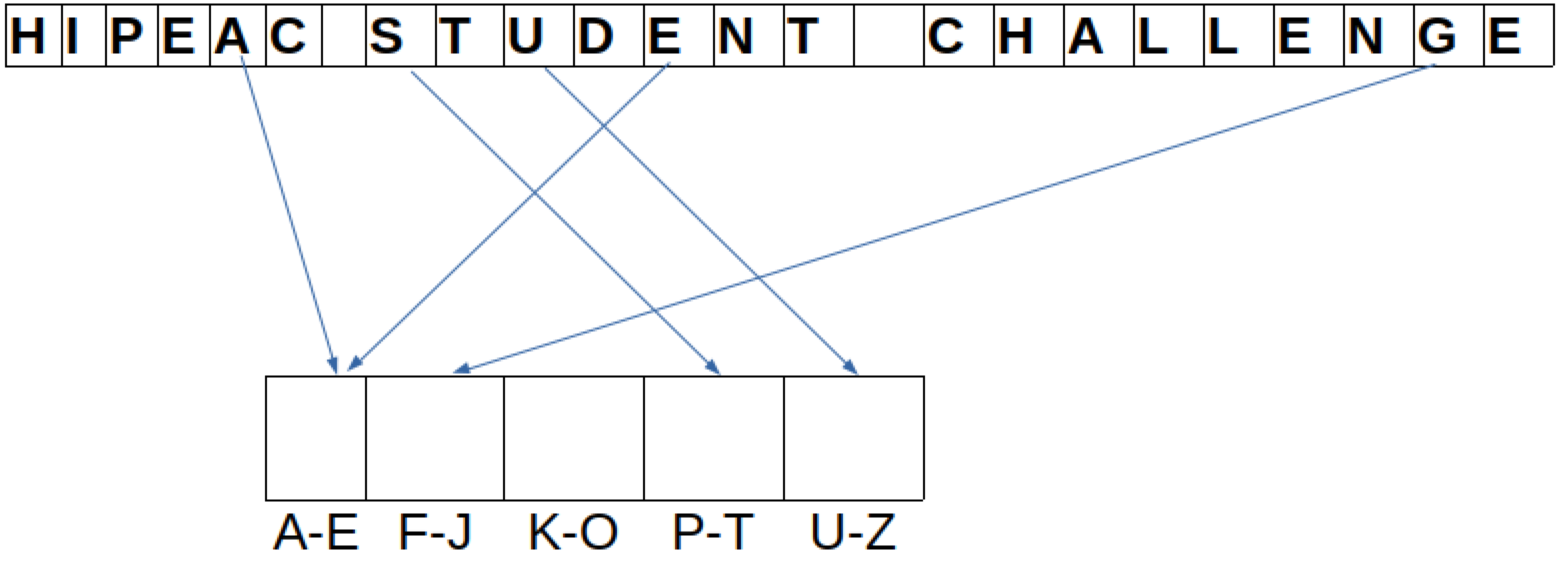
Overview

- General Information
- GPU Implementation
- CPU Implementation
- FPGA Implementation

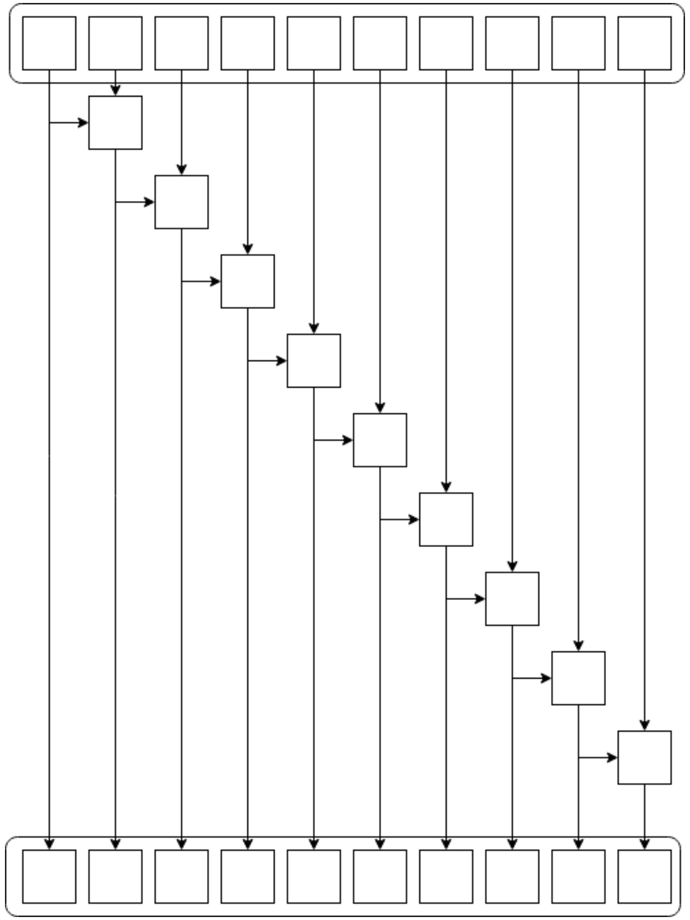
General Information

- Consists of four parts:
  - Calculating Histogram
  - Calculating Cumulative Distribution Function
  - Finding non-zero minimum of CDF
  - Calculating Equalization

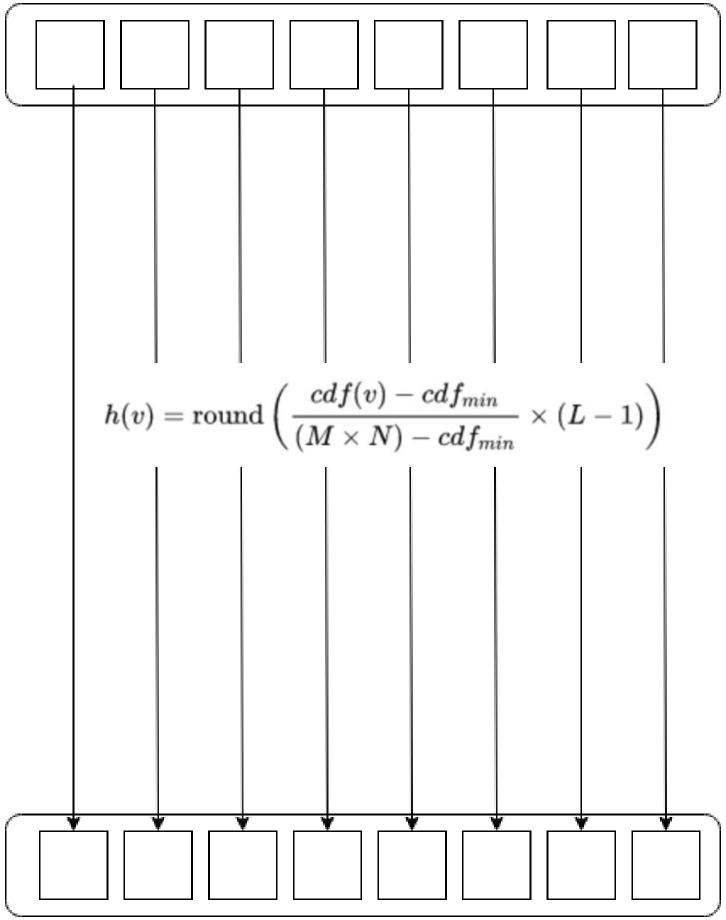
Histogram Calculation



Calculating CDF && Finding Minimum



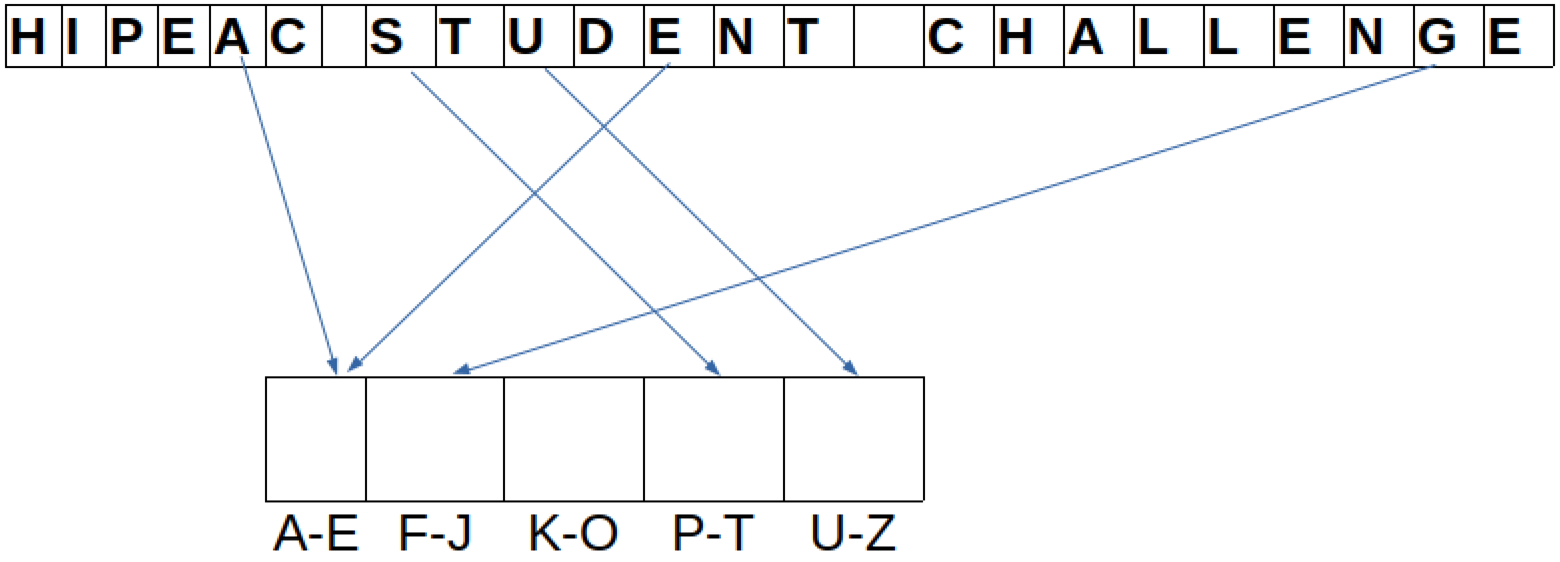
Calculating Equalization



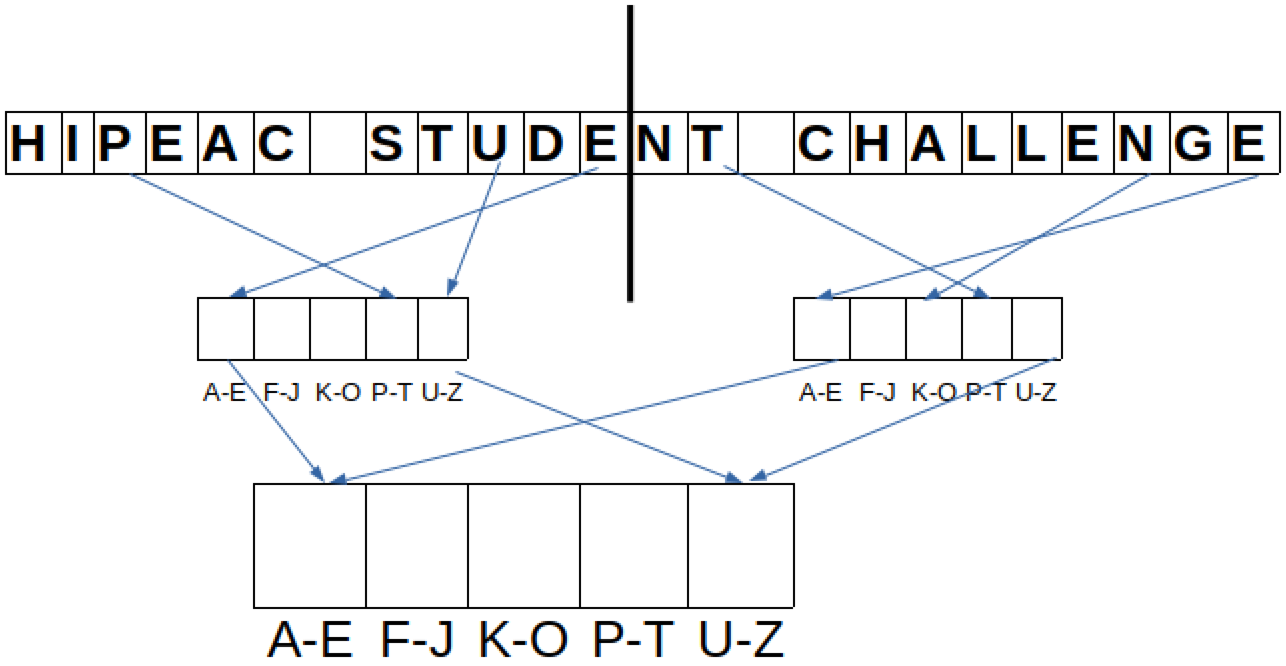
$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L - 1)\right)$$

GPU Implementation

Histogram Calculation



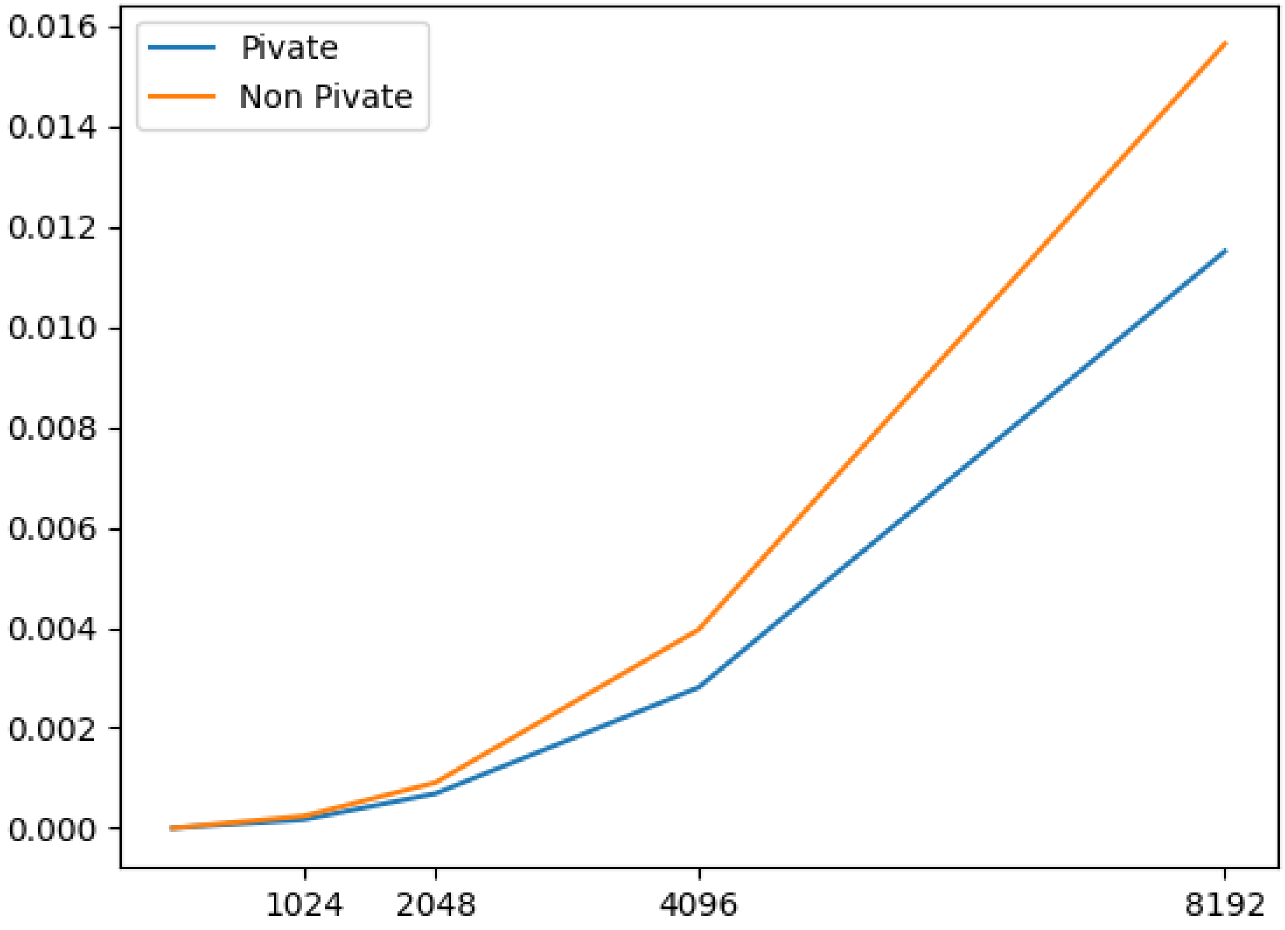
Histogram Calculation



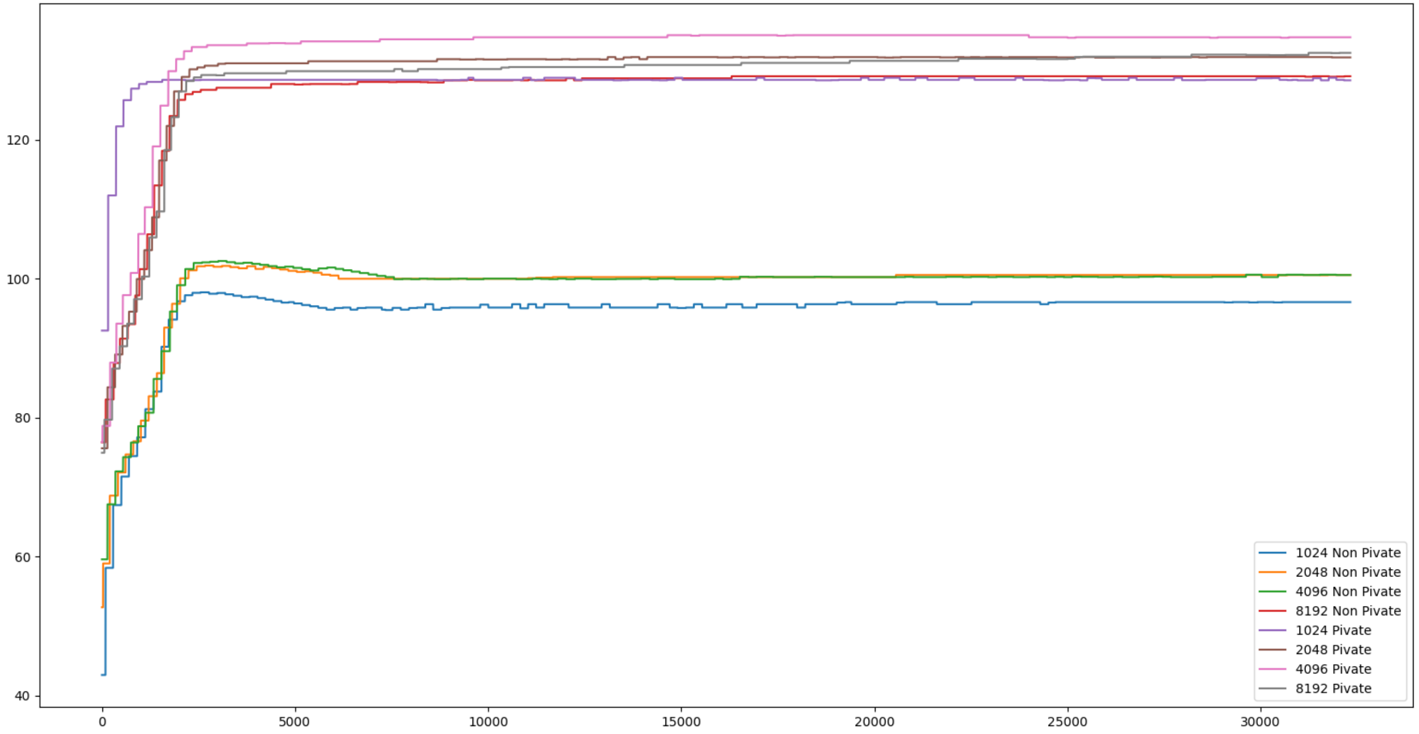
Histogram Calculation

- Privatization increases
  - Atomic Operations
  - # of total memory access
- Yet it decreases
  - # of total global memory access
- Therefore, it's expected that *decreasing* in executing time

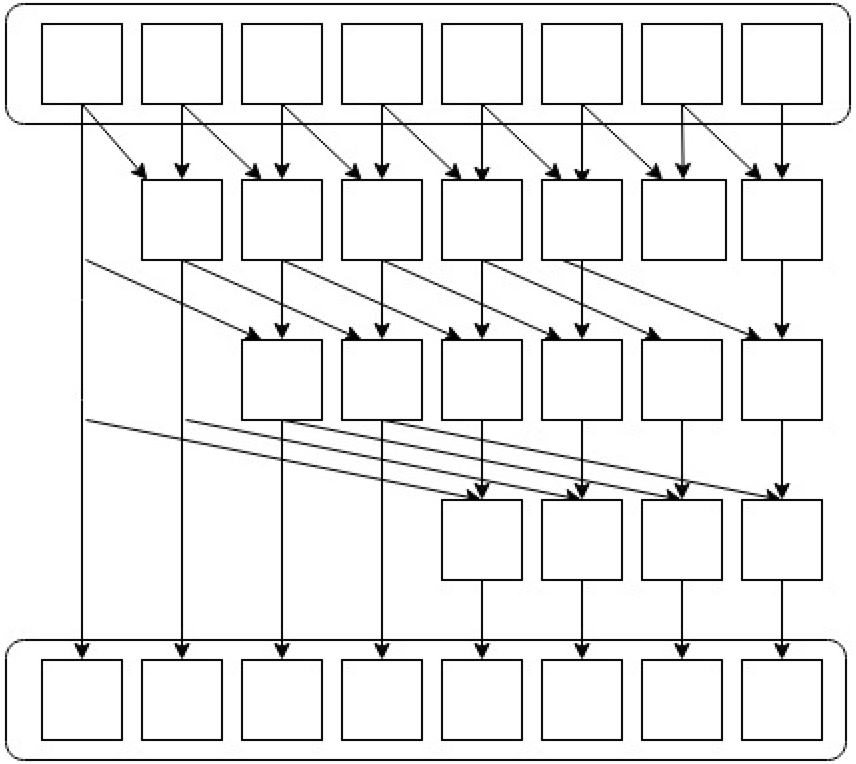
Calculating Histogram – Experiments



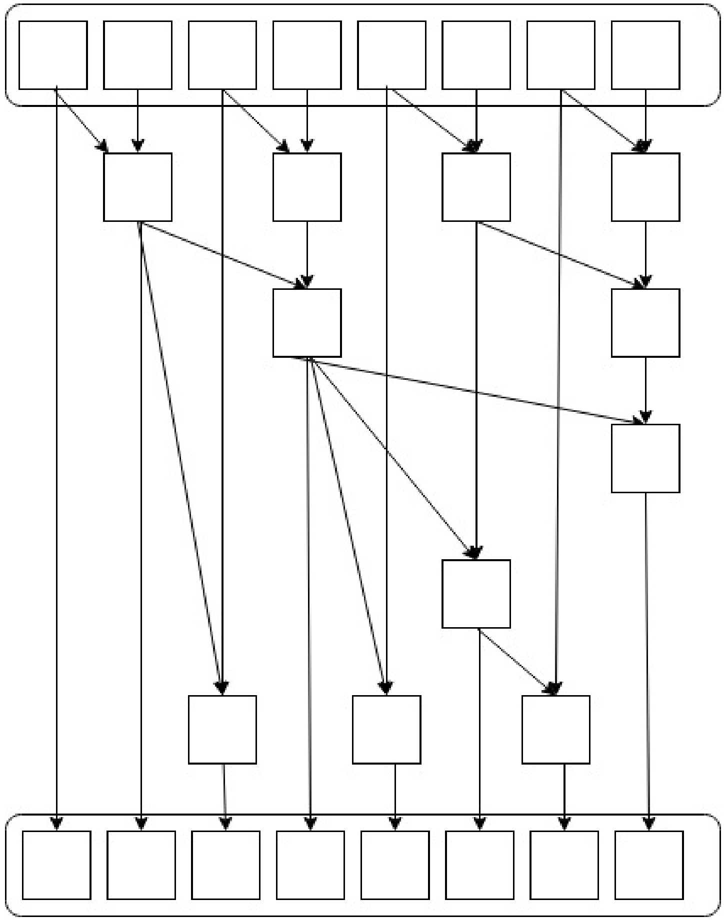
Calculating Histogram – Experiments



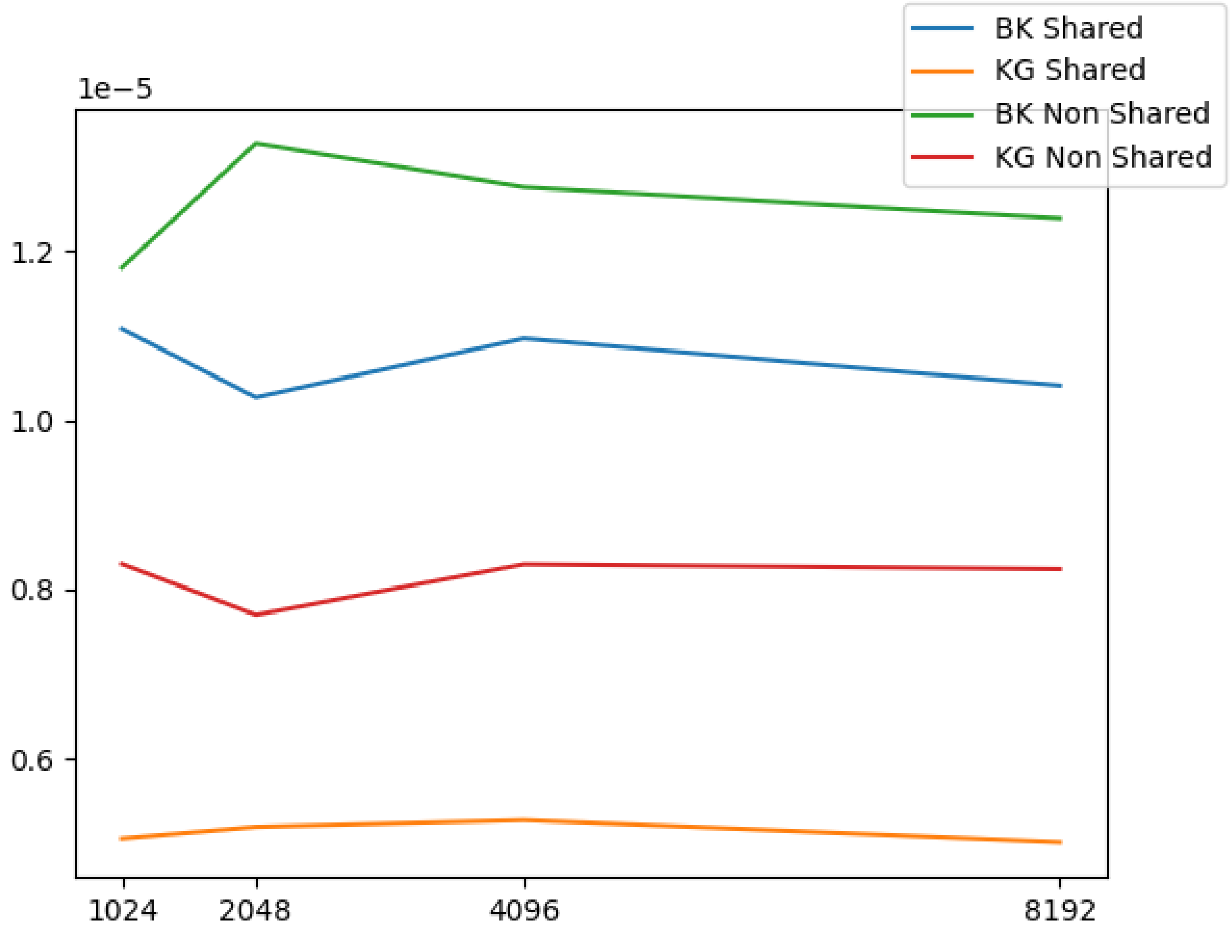
Calculating CDF && Finding Minimum



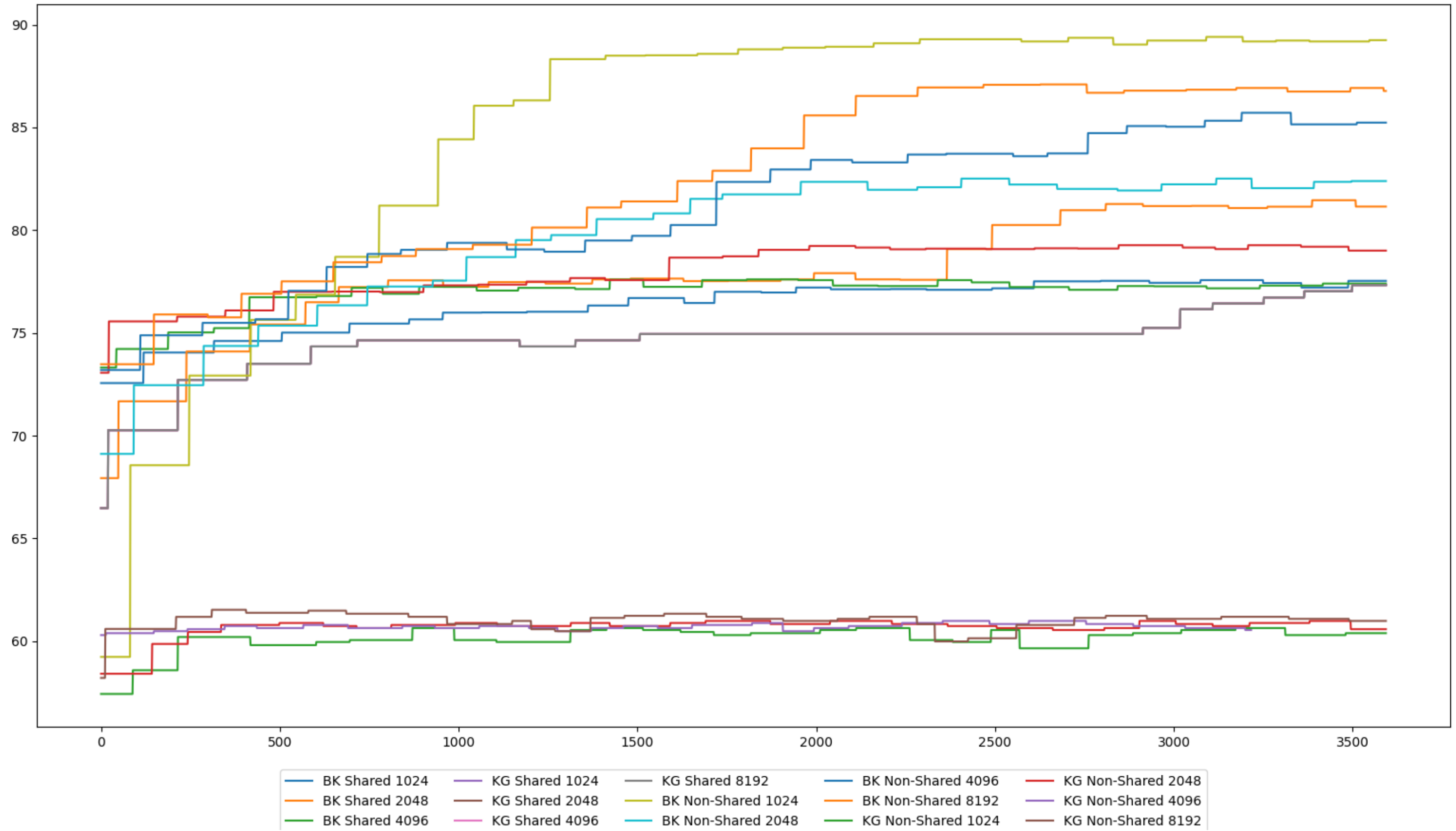
Calculating CDF && Finding Minimum



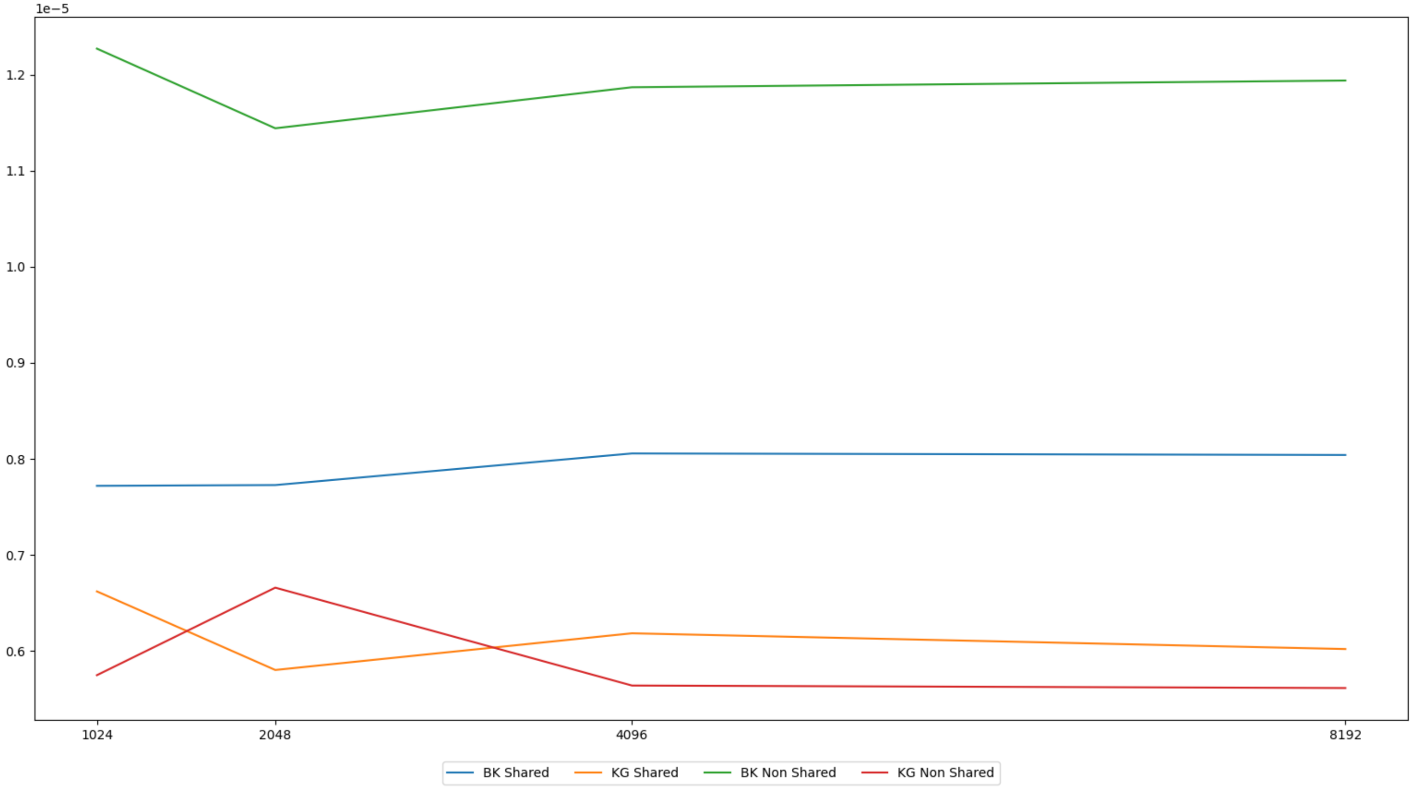
Calculating CDF - Experiments



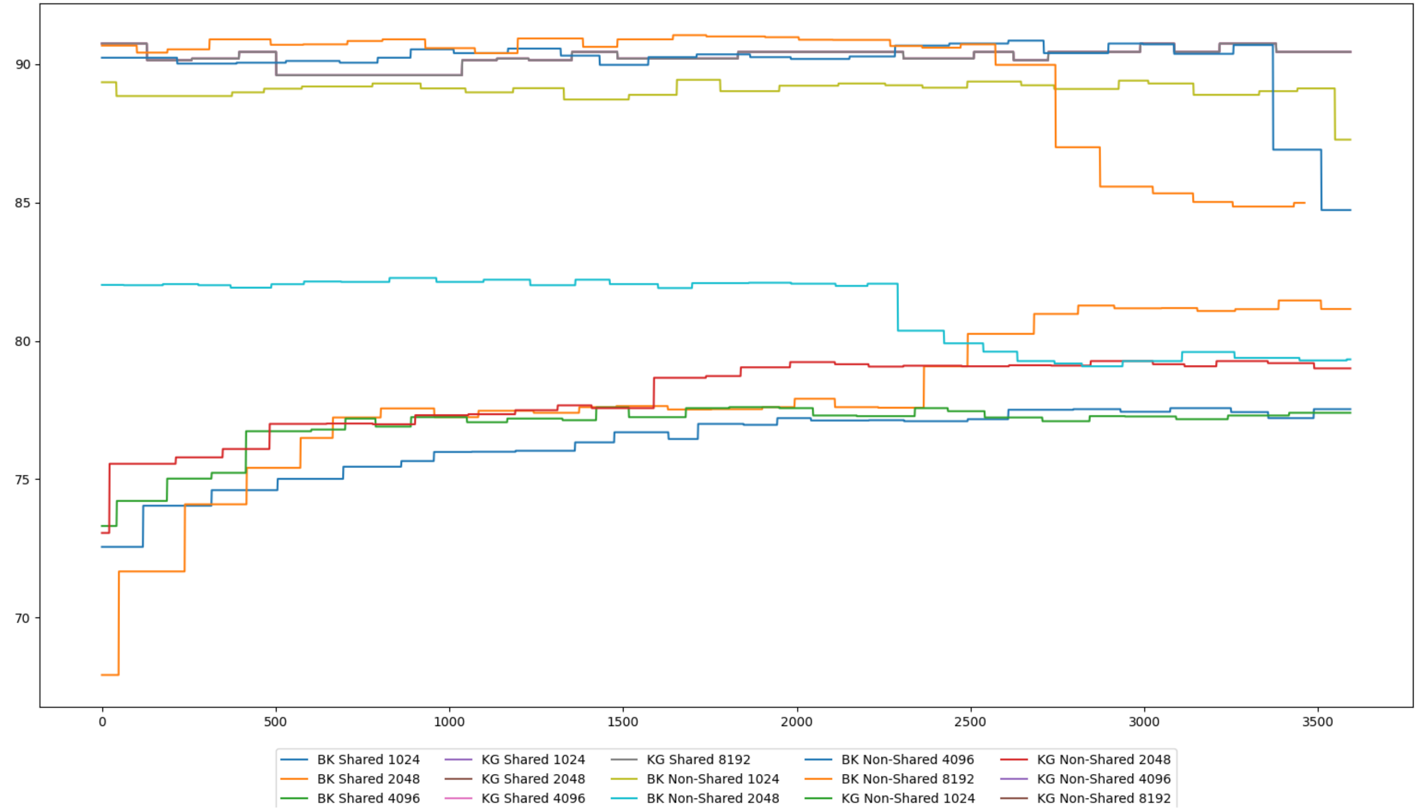
Calculating CDF - Experiments



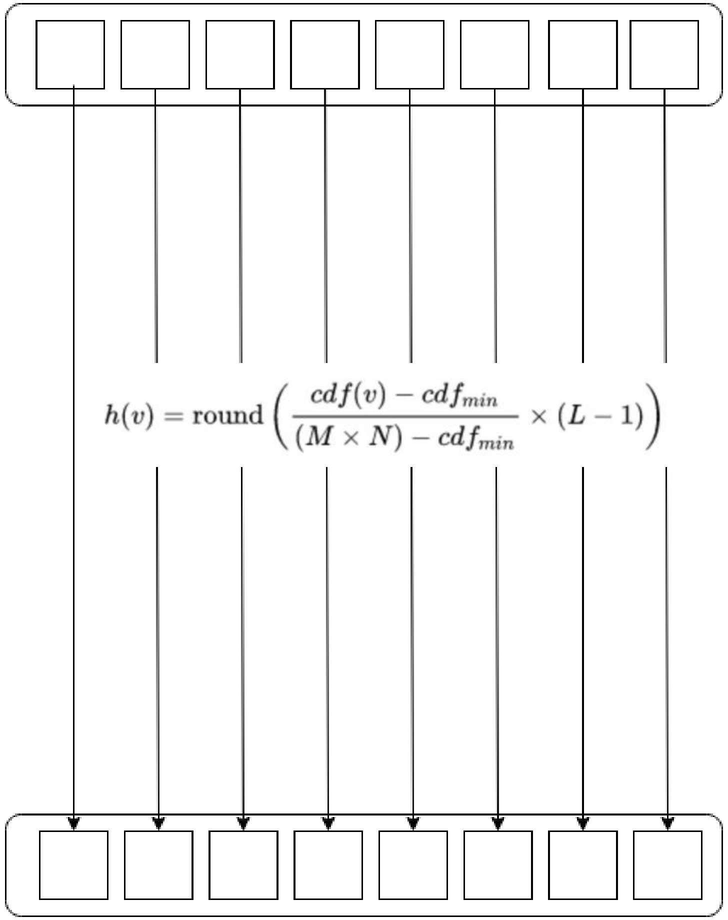
Finding Minimum - Experiments



Finding Minimum - Experiments

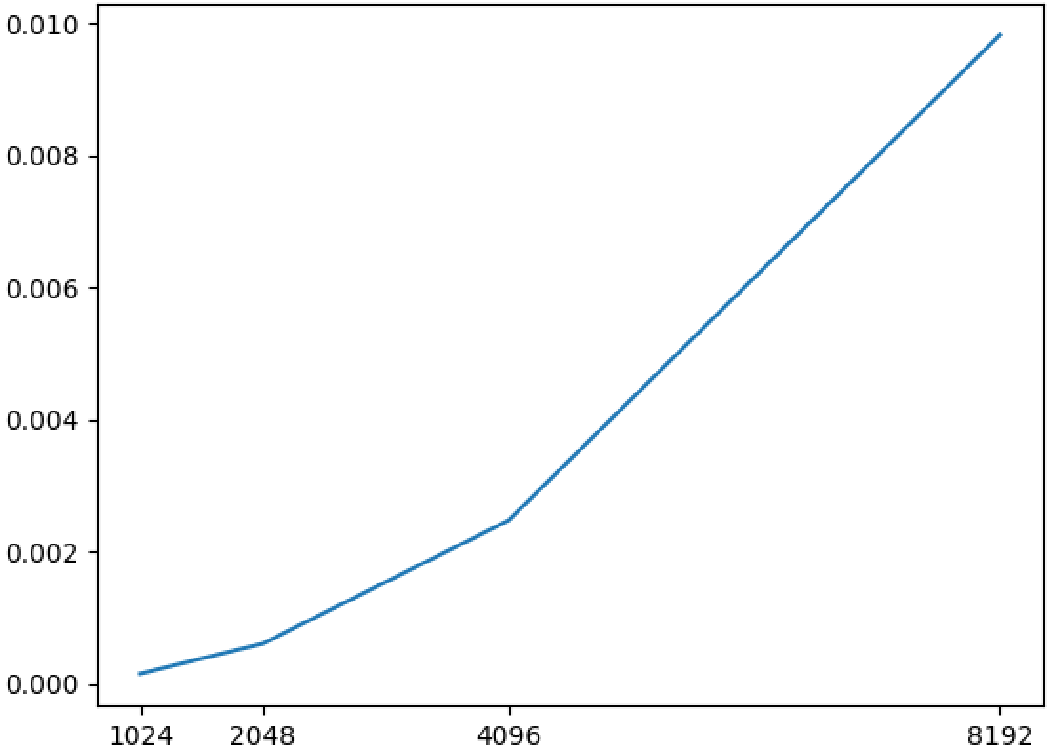


Calculating Equalization

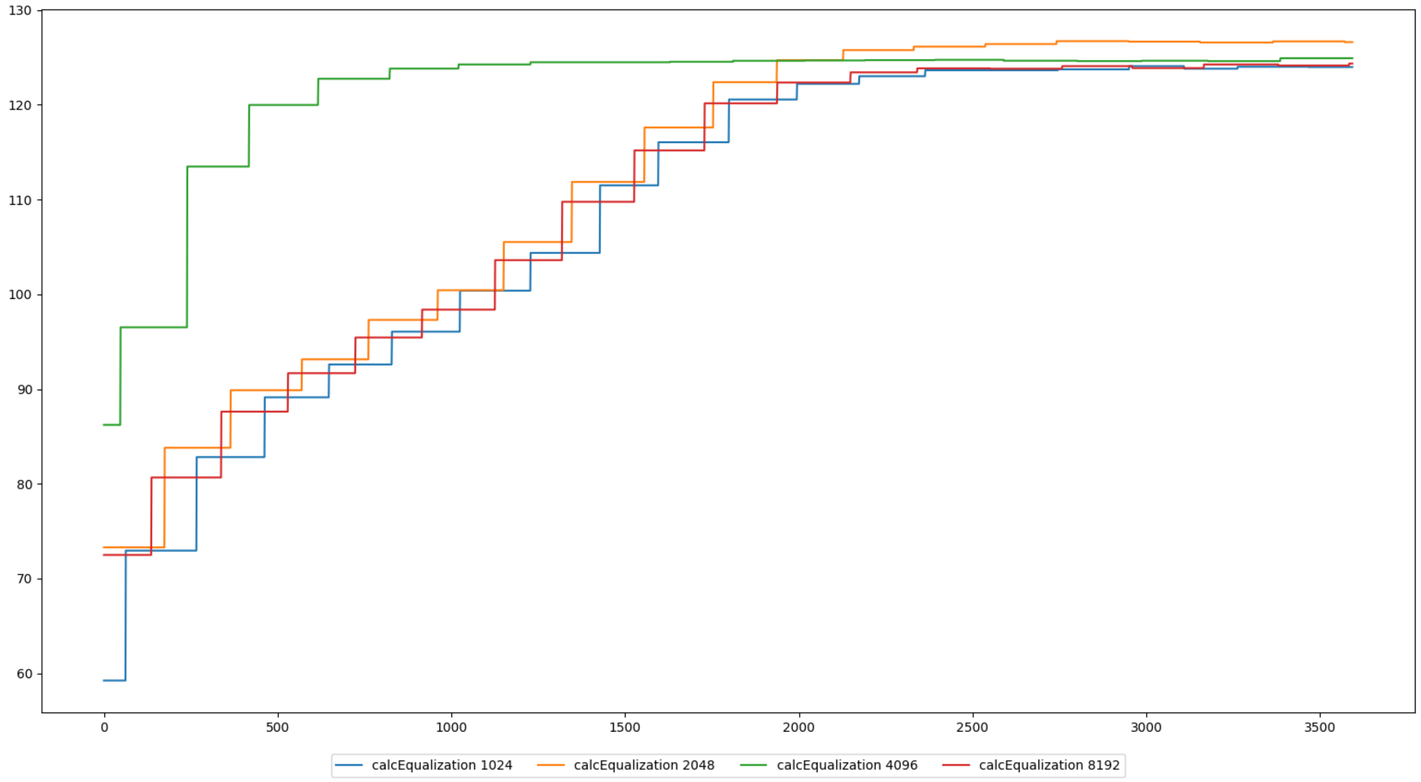


$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Calculating Equalization - Experiments



Calculating Equalization - Experiments



CPU Implementation

Parallel Programming Patterns

Step	Pattern
Histogram Creation	Parallel For
CDF Calculation	Parallel Prefix Sum
Equalization	Parallel Map
Remapping	Parallel Gather

Environment, Libraries & The Language

- C++17
- Clang 13 with gcc-9 libs
- LLVM OMP API version: 5.0
- Linux ubuntu 5.11.0-38-generic

Test System Properties

- Intel(R) Xeon(R) Gold6148 CPU @ 2.40GHz
  - 20 Cores, 40 Threads
- 192 GB RAM
- Centos 7 Linux

*Istanbul Technical University National Center for High Performance Computing

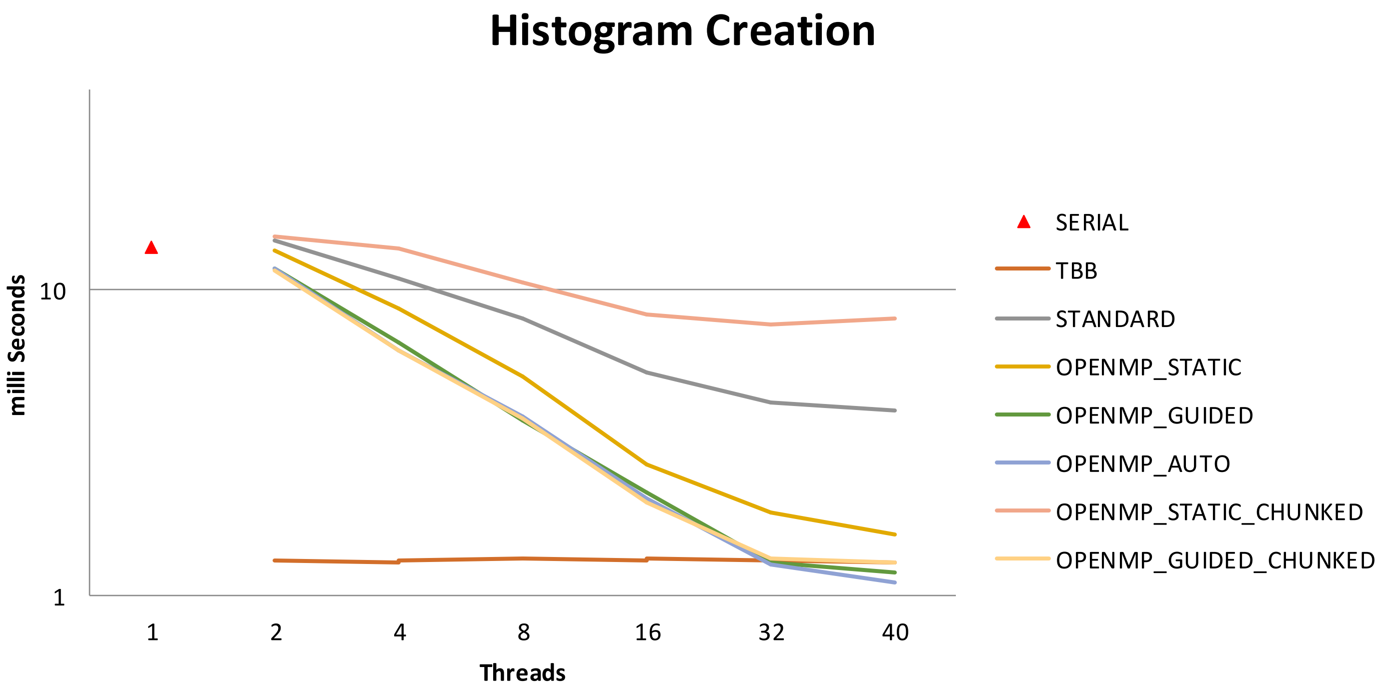
Comparison Categories

- Thread Size (Lyon 4096x4096)
- Image Size (Lena, with 32 threads)

Loop Schedulers

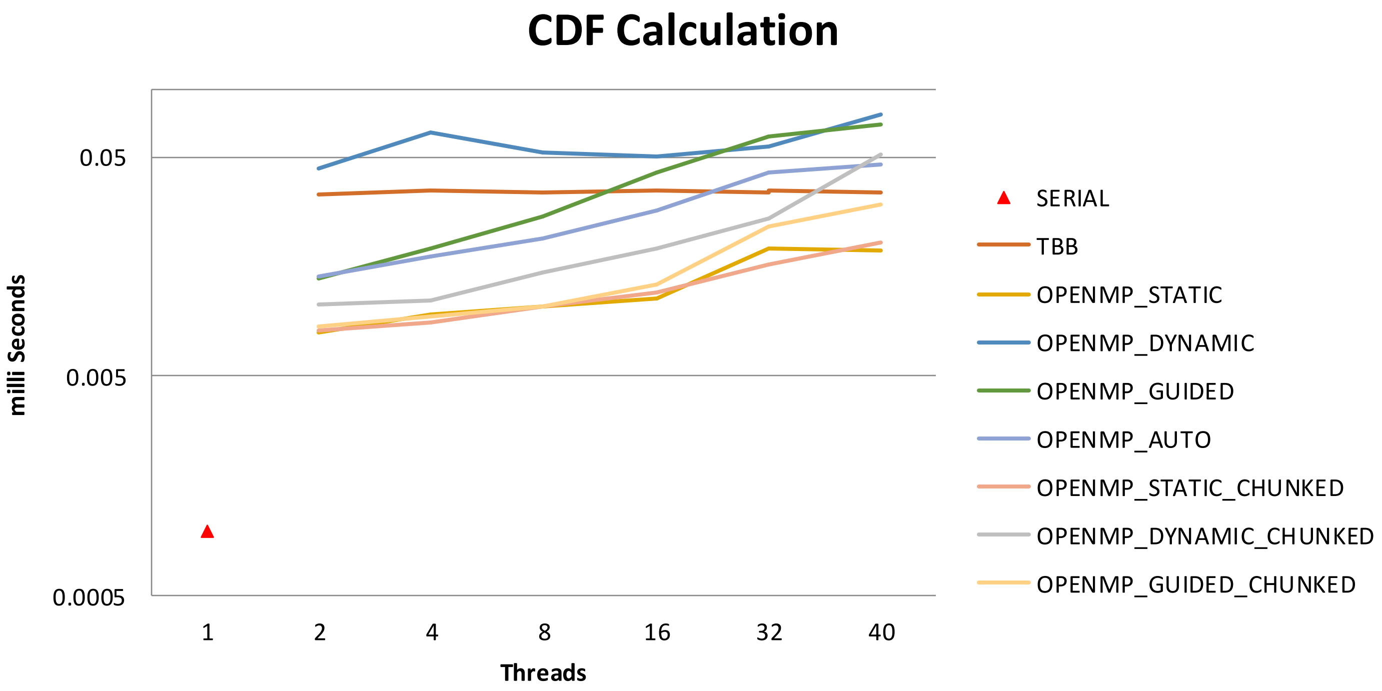
- OPENMP_STATIC
  - divides loop into chunks, distribute in cycle order
- OPENMP_DYNAMIC
  - no order in chunk distribution
- OPENMP_GUIDED
  - the size of the chunks decreases
- OPENMP_AUTO
- OPENMP_STATIC_CHUNKED
- OPENMP_DYNAMIC_CHUNKED
- OPENMP_GUIDED_CHUNKED

Thread Size (Lyon 4096x4096)



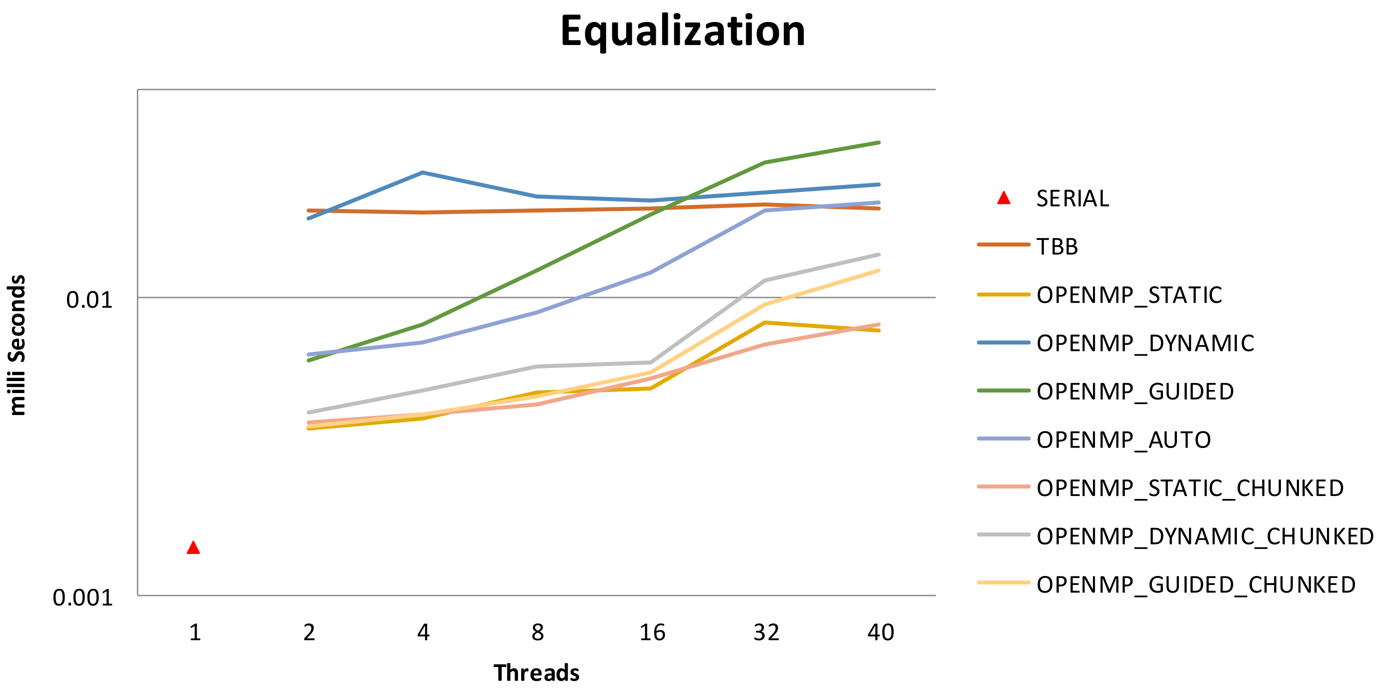
https://unsplash.com/photos/dC0cCASdRck

Thread Size (Lyon 4096x4096)



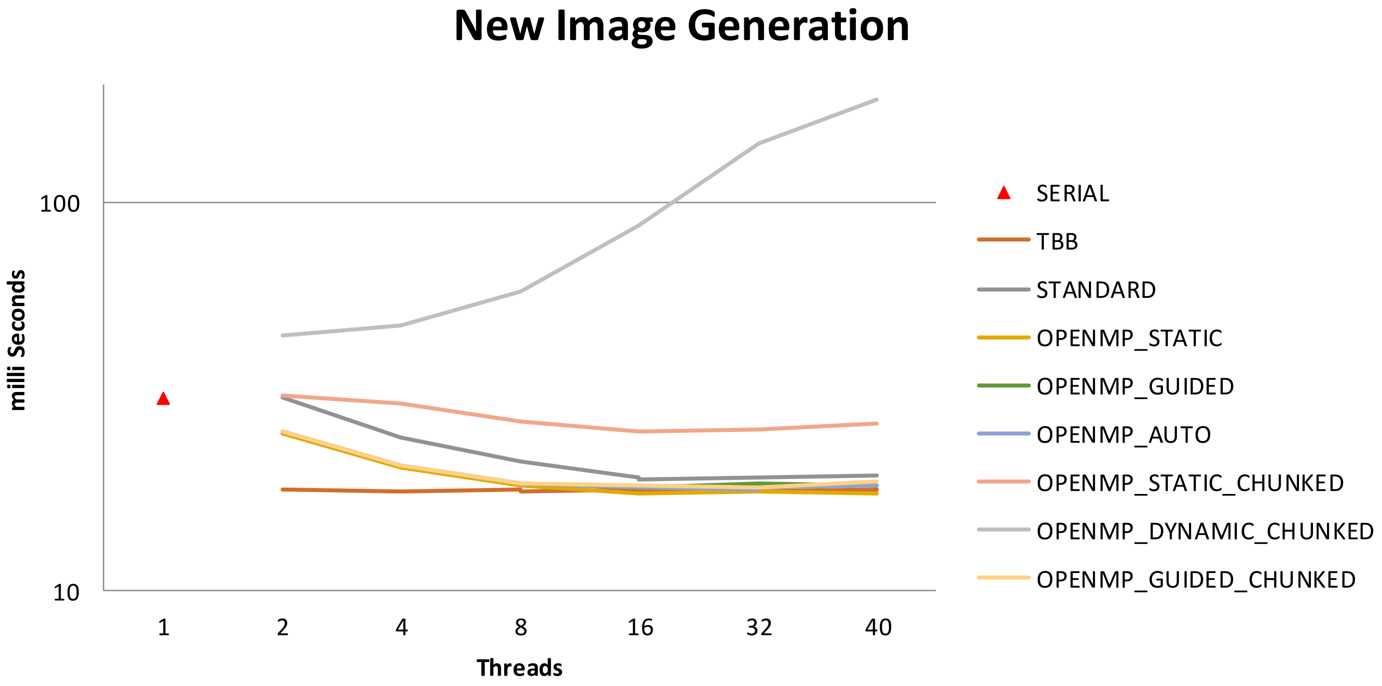
https://unsplash.com/photos/dC0cCASdRck

Thread Size (Lyon 4096x4096)



https://unsplash.com/photos/dC0cCASdRck

Thread Size (Lyon 4096x4096)



https://unsplash.com/photos/dC0cCASdRck

Image Size (Lena, with 32 threads)

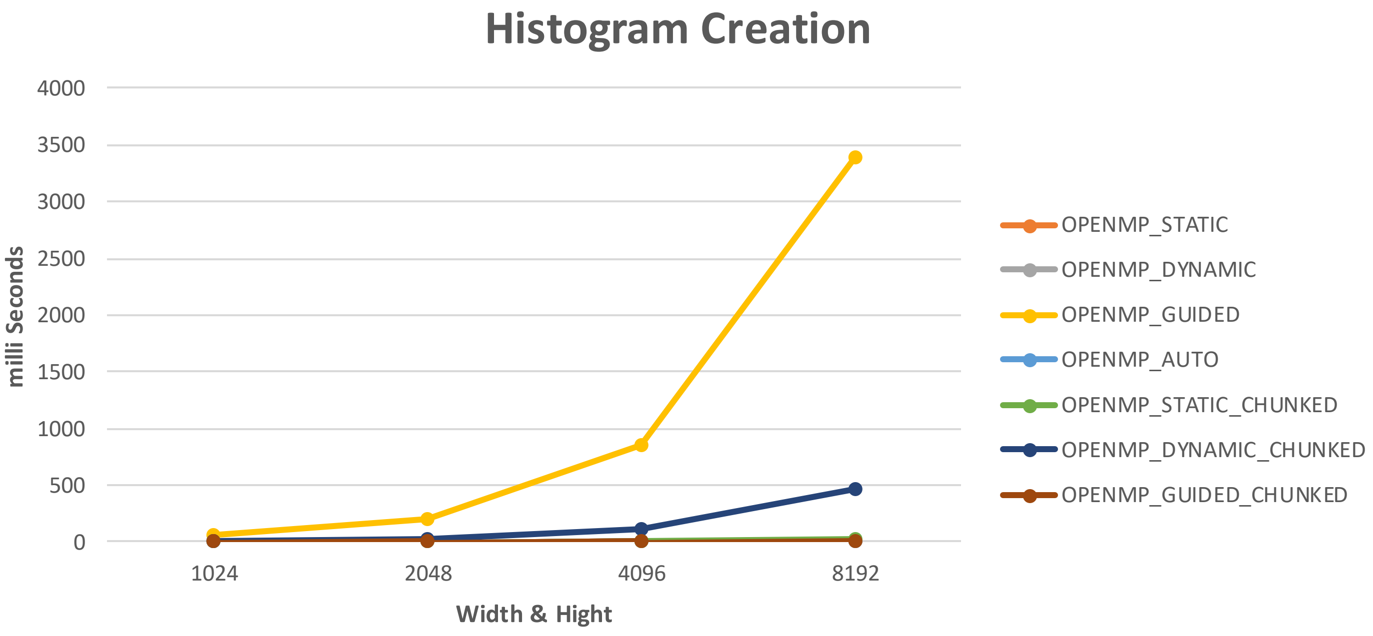


Image Size (Lena, with 32 threads)

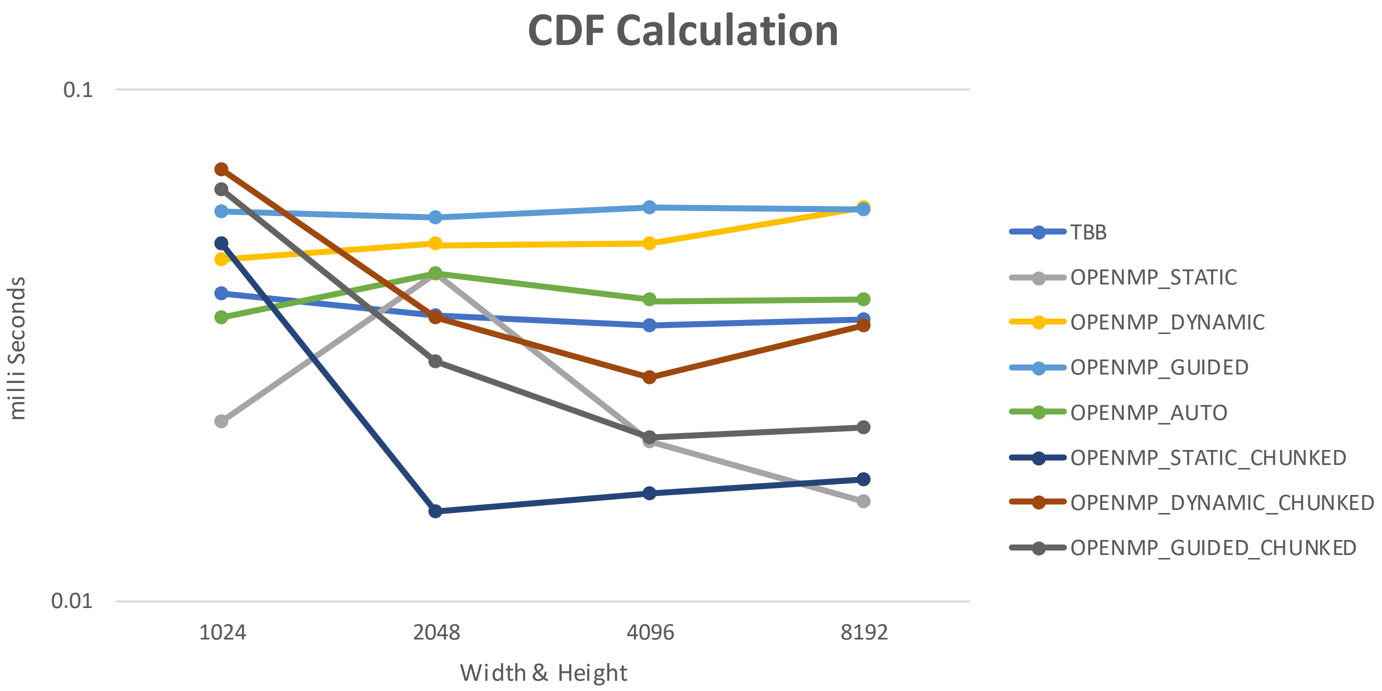


Image Size (Lena, with 32 threads)

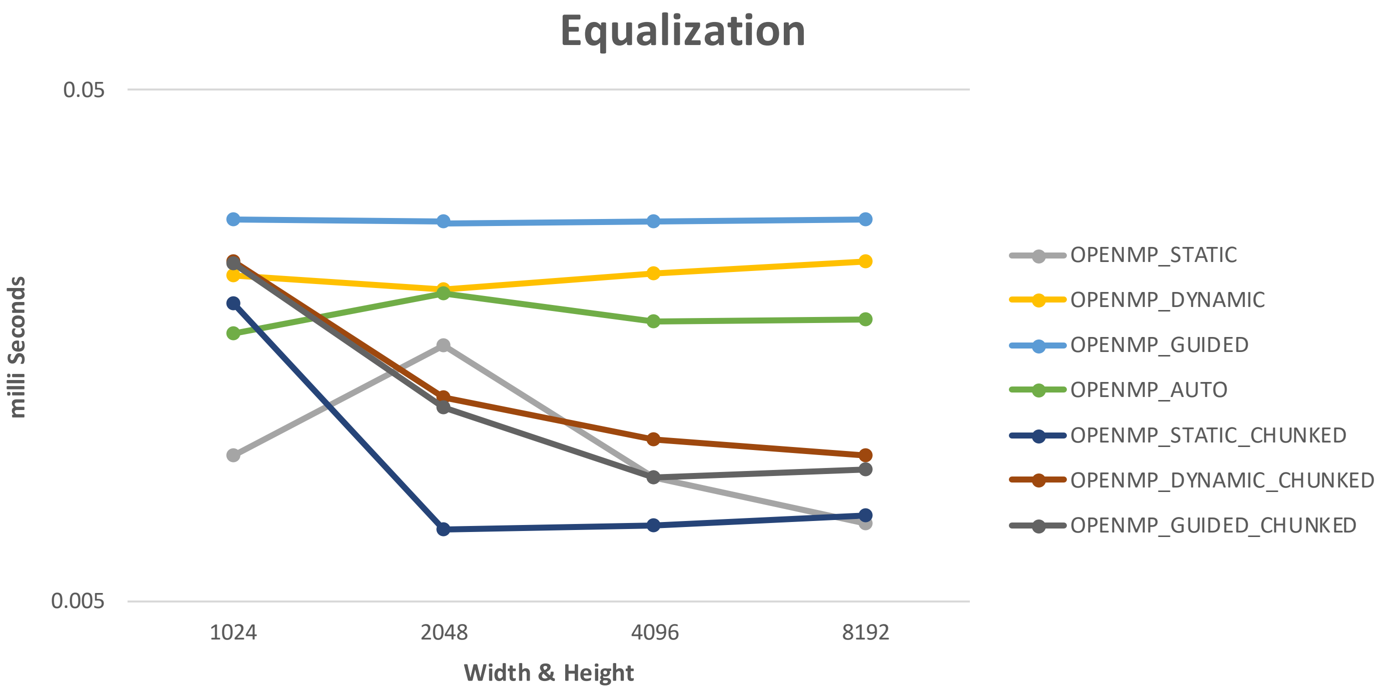
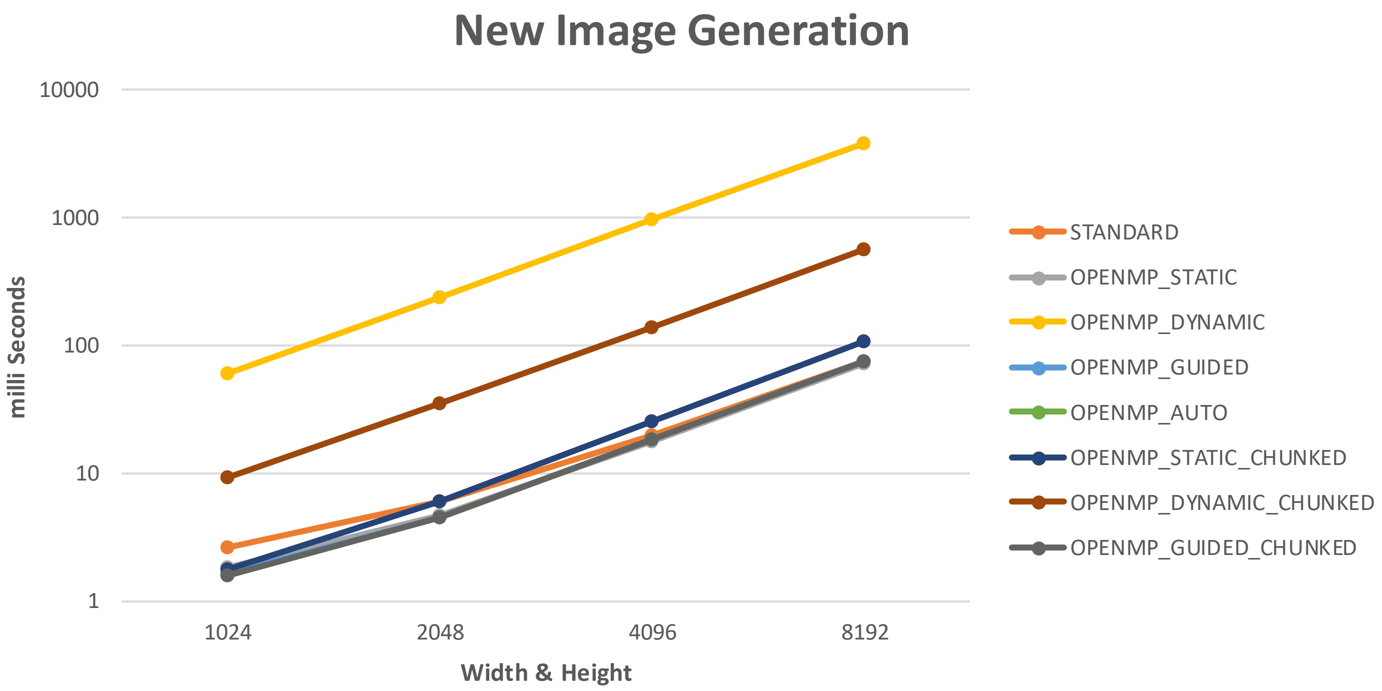


Image Size (Lena, with 32 threads)



Histogram Equalization Implementation on FPGA

Histogram Calculation and Histogram Equalization in FPGA

$$hist\Big[image[y][x]\Big] = hist\big[image[y][x]\big] + 1;$$

$$hist\Big[image[y][x+1]\Big] = hist\Big[image[y][x+1]\Big] + 1;$$

$$hist\Big[image[y][x+2]\Big] = hist\Big[image[y][x+2]\Big] + 1;$$

$$hist\Big[image[y][x+3]\Big] = hist\Big[image[y][x+3]\Big] + 1;$$

Calculation of histogram values

- In each clock, this implementation reads 4 pixels from the image.
- Histogram values are increased for each existing pixel values.

Width = 512, Height = 512

$$image[y][x] = \frac{CDF\Big[image[y][x]\Big] - CDF_{min}}{width * height - 1} * (256-1)$$

$$image[y][x+1] = \frac{CDF\Big[image[y][x+1]\Big] - CDF_{min}}{width * height - 1} * (256-1)$$

$$\cdots$$

$$\cdots$$

$$image[y][x+15] = \frac{CDF\Big[image[y][x+15]\Big] - CDF_{min}}{width * height - 1} * (256-1)$$

Histogram Equalization Step

- It shows writing 16 pixels in each clock as parallel.
- This requires comparably bigger circuitry to calculate the equalized results because of the division and multiplication operations

Calculating CDFs for Each Pixel Value

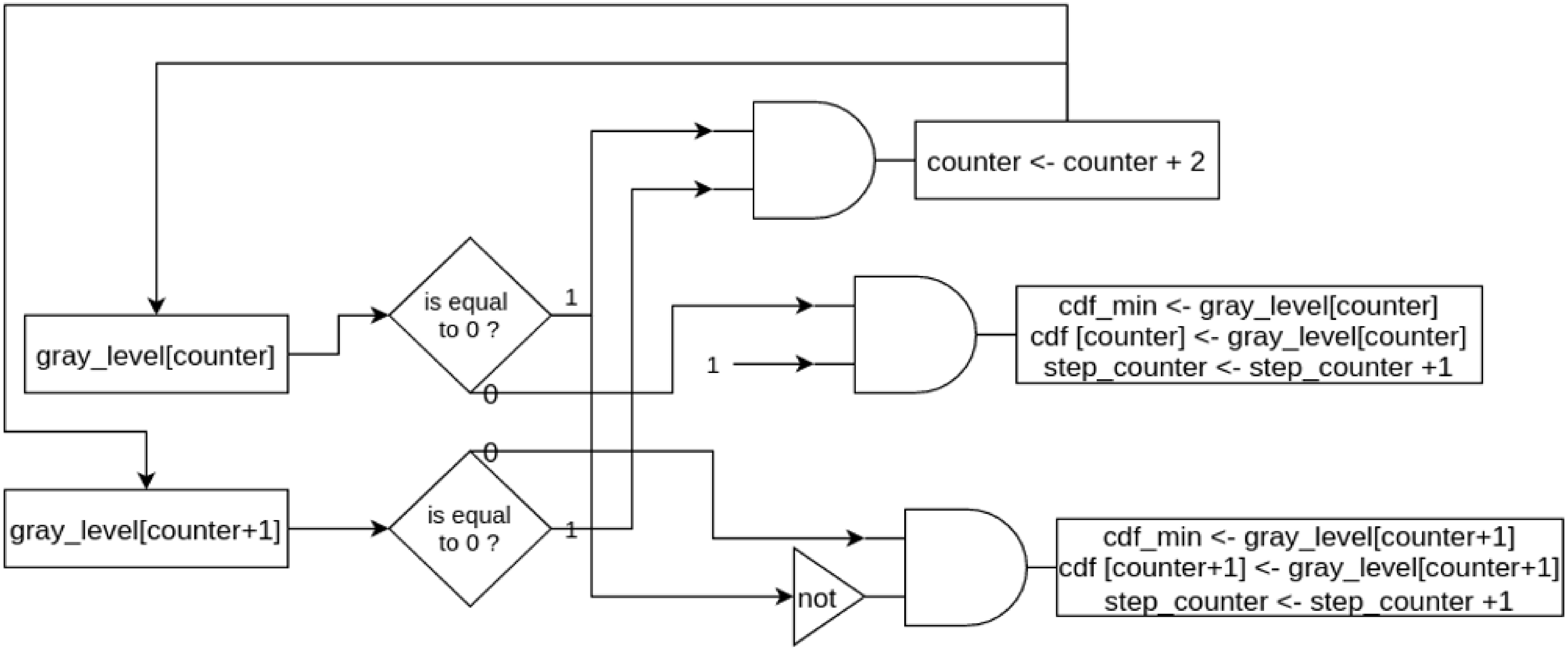
- CDFs for each pixel value is calculated as in the following way:

$$\mathrm{CDF}\left[\mathrm{counter}\right] = \mathrm{CDF}\left[\mathrm{counter} - 1\right] + \mathrm{hist}\left[\mathrm{counter}\right]$$

- This step of the algorithm is implemented in a sequential way.
- Since we have at most 256 calculations to complete this step and the following step, it does not take too much time to complete.

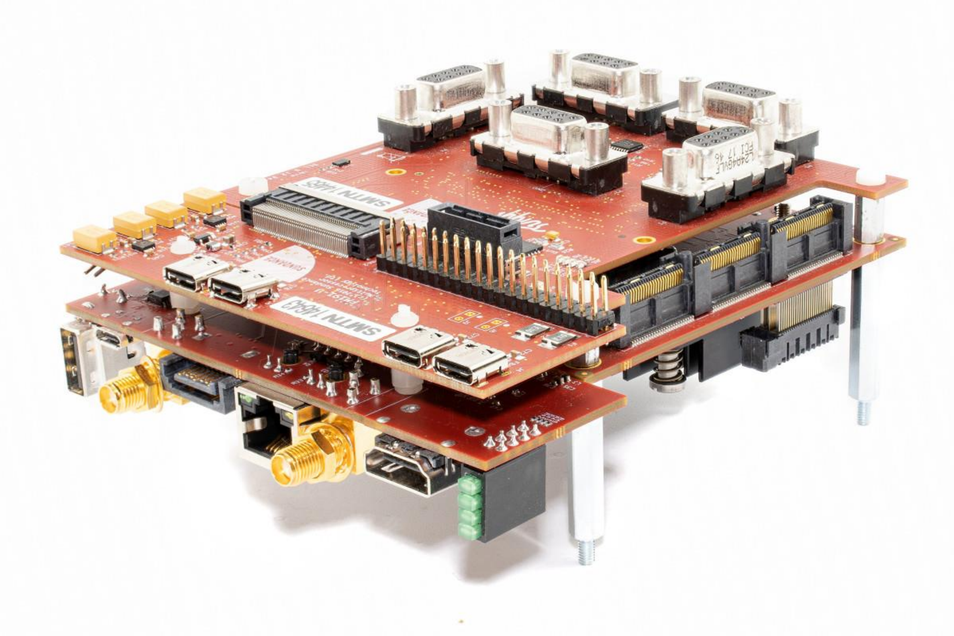
Finding Non-Zero CDF

- To find the first non-zero CDF, we implement a sequential approach as in CDF calculation.
- We can also implement more parallelized versions for this step as described below.
- However, the performance increase of parallelizing this step can be ignored since we will just scan 256 values at maximum, I implement this step in a sequential way.



Design Specifications

- In the experiments, VCS-1 board provided by Sundance Technology is used.
- In this board, there is a Zynq UltraScale+ MPSoC circuitry. There are also multiple peripherals to be used with additional sensors for robotic applications.
- Clock rate is configured as 50MHz.
- Lenna's image which has 512*512 pixels is trained in this experiment.
- There are 2 * 512 MB DDR4 SDRAM memories in Zynq US+ MPSoC which operates in range [800,1600] MHz.

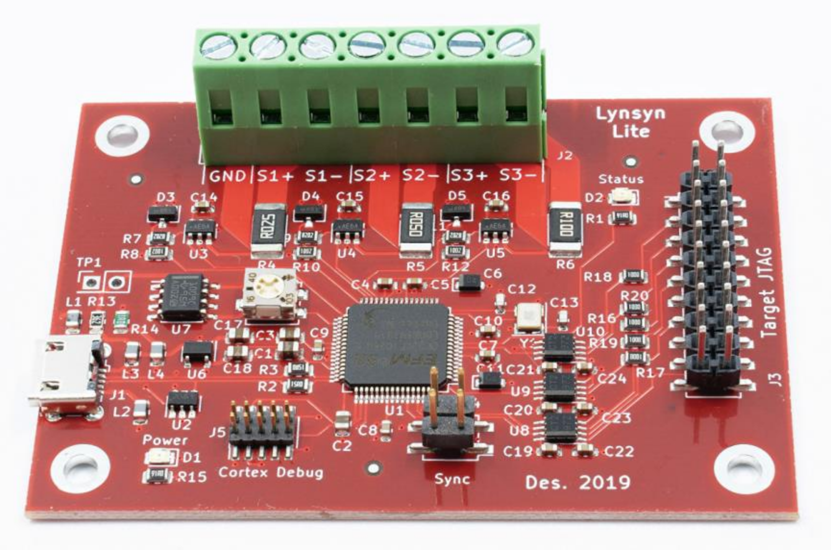


PERFORMANCE RESULTS

HARDWARE DESIGN SPECIFICATIONS	PERFORMANCE(msec)
• **Histogram calculation -> 4 pixels in each clock** • **CDF_min -> sequential (Calculating 1 pixel value in each clock)** • **CDF calculations -> sequential** • **Histogram equalization -> 4 pixels in each clock**	**10.49**
• Histogram calculation -> 8 pixels in each clock • CDF_min -> sequential (Calculating 1 pixel value in each clock) • CDF calculations -> sequential • Histogram equalization -> 8 pixels in each clock	5.36
• Histogram calculation -> 16 pixels in each clock • CDF_min -> sequential (Calculating 1 pixel value in each clock) • CDF calculations -> sequential • Histogram equalization -> 16 pixels in each clock	2.72
• Histogram calculation -> 32 pixels in each clock • CDF_min -> sequential (Calculating 1 pixel value in each clock) • CDF calculations -> sequential • Histogram equalization -> 32 pixels in each clock	1.39

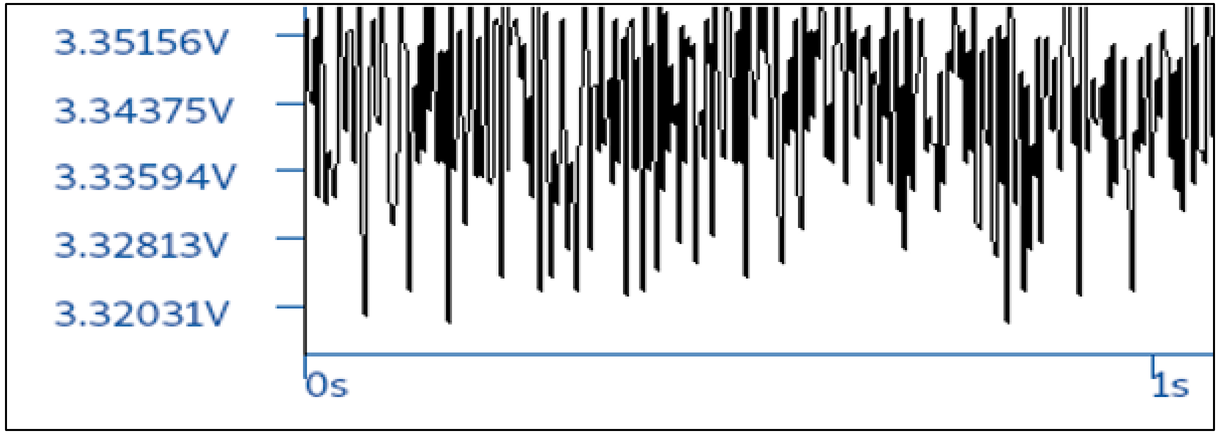
Power Measurement

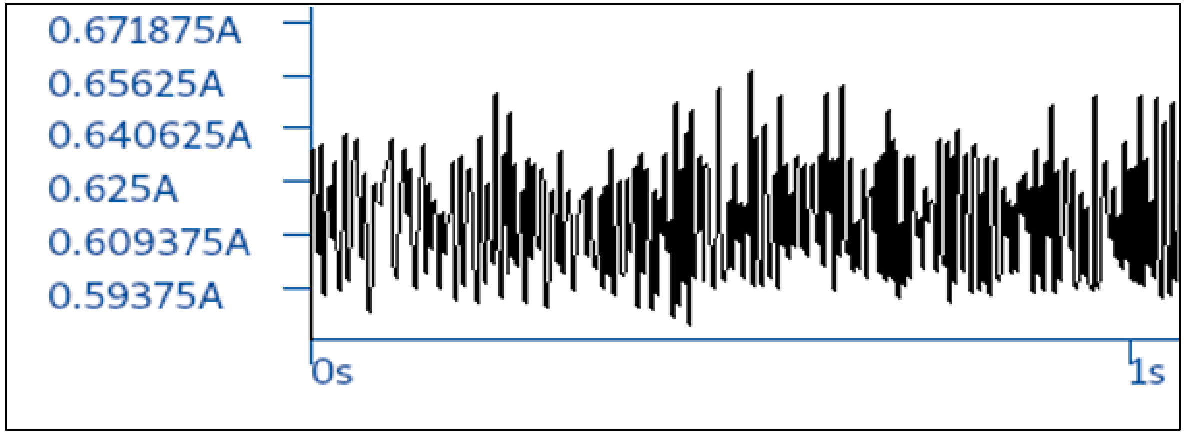
- LynSyn Lite is a device used to measure the power consumption of the embedded application running on the host machine with the help of either JTAG or USB-Micro cables.
- One can profile any application by specifying profiling criteria such as adding breakpoints to the code and, see the results via the viewing GUI shared in the github repository.
- Sundance Multiprocessor is also one of the vendors of this device integrated with the VCS-1 integrated device.

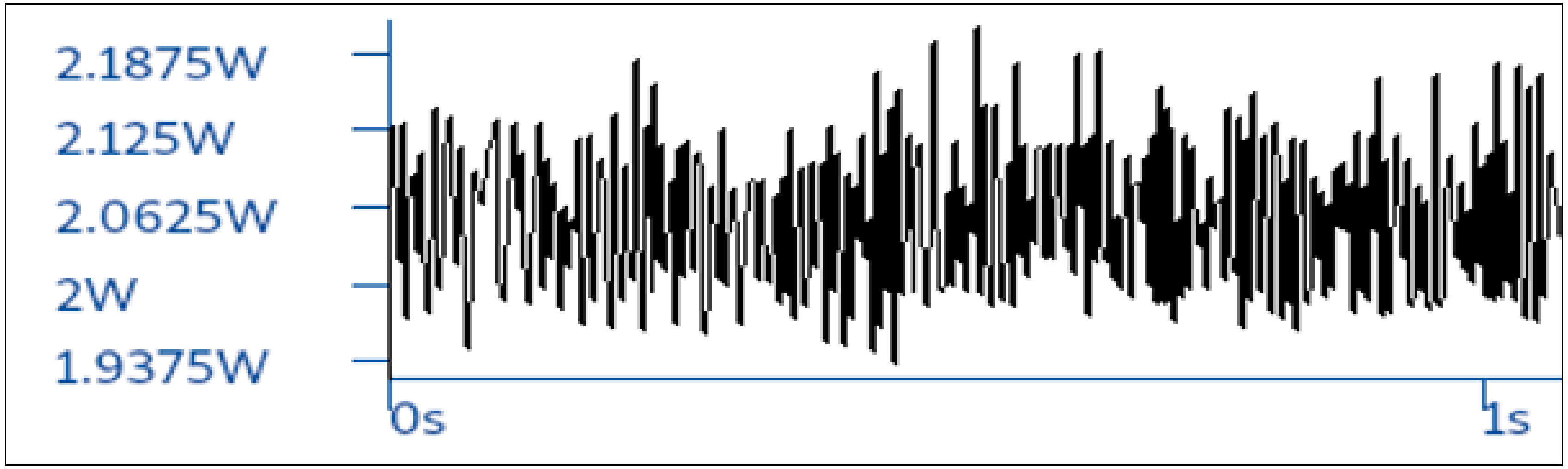


https://github.com/EECS-NTNU/lynsyn-host-software/wiki/Lynsyn-Lite

Power Measurement Results







The algorithm consumes **2.047 watt/sec** power at average.

Performance and Power Consumption Results on FPGA

- As a result, Histogram Equalization algorithm with
  - Histogram calculation of **4 pixels in each clock**,
  - Finding Non-zero CDF in a **sequential way**,
  - Calculating CDF values for each pixel values in a **sequential way**,
  - Calculating equalized histogram values as **4 pixels in each clock**

  takes **10.49 milliseconds without including reading/writing from memory**.
- Also, the algorithm consumes **2.047 watt/sec power** which is equal to **2.047 joule/sec** energy consumption**.**

Thank you for your kind attention.