

Microprocessor Architectures. SIMD lab 3 report. Raymond Lochner and Aldar Saranov.

Resource sharing

The main shared resource in the project is the image matrix. In order to implement a parallel image processing, one must be able to access it simultaneously by the means of several threads. The pixel processing is basically done in the same way as in the first two labs except that they operate within separate areas.

Threshold

Since in threshold processing every pixel is independent one can simply update the pixel values to the thresholded ones. This is of course only possible if each thread does not overlap the processing areas of the other threads. For example an image with 2000 pixels can be sliced by 2 threads into 2 areas of 1000 pixels each.

Minmax

As a conceptual difference from the threshold processing, minmax processes each pixel depending on its neighbors. Therefore one must store the result matrix as a separate matrix.

Partition form

One can use several trivial partition forms:

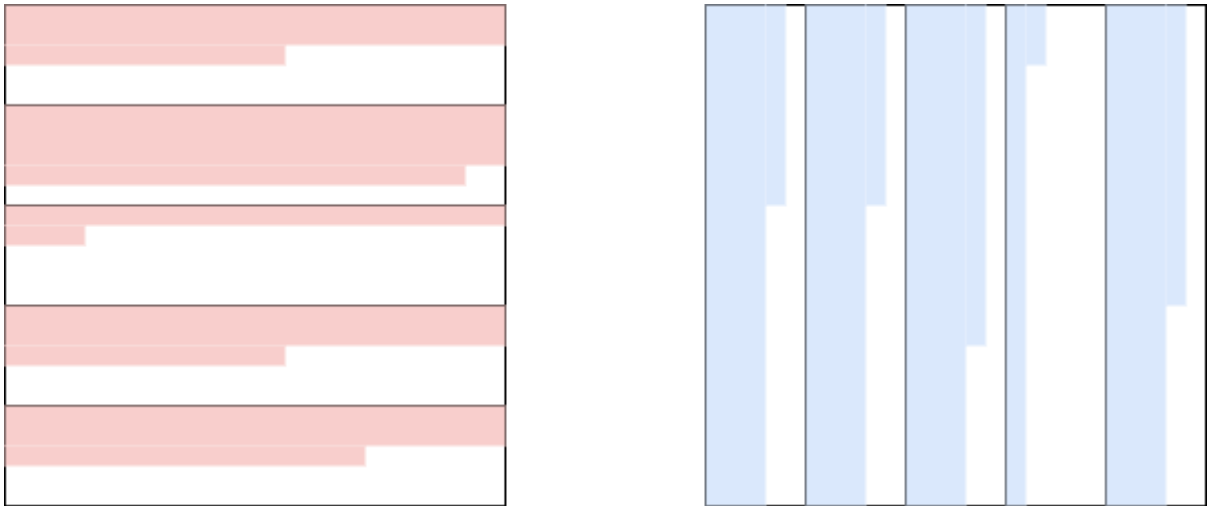


Figure 1. Horizontal and vertical matrix partitions.

Both horizontal and vertical partitions gain equivalent areas of the matrix and therefore the partition can be considered “fair”. The advantage of the left approach is that whilst the matrix can be represented as a one-dimensional array (row by row) this partition will allow to treat these areas as sub-arrays of this array and forget its matrix-type nature.

Thread management

In order to ensure that every thread finished its execution we use an array of mutexes where each mutex corresponds to one thread and stands for its termination. The threads are initialized using `CreateThread()` function which creates a vector of thread handles at one invocation. Right after the thread initializing the main thread converts to awaiting state of the child thread completion which is done by `CloseHandle()` function.

Miscellaneous

The project was implemented in Visual Studio 13.