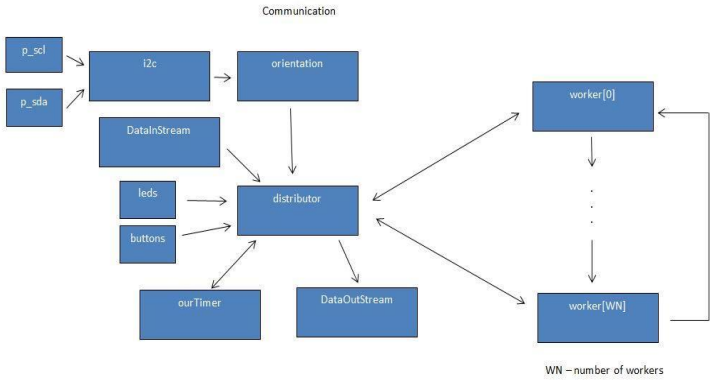
Andrei Alexandru : aa17956@my.bristol.ac.uk
Erich Reinholtz : er17660@my.bristol.ac.uk

Functionality and Design

To considerably increase the possible size of images to be processed, we implemented packing and unpacking functions, where we store 8 bits, which reduced our memory usage by eightfold.

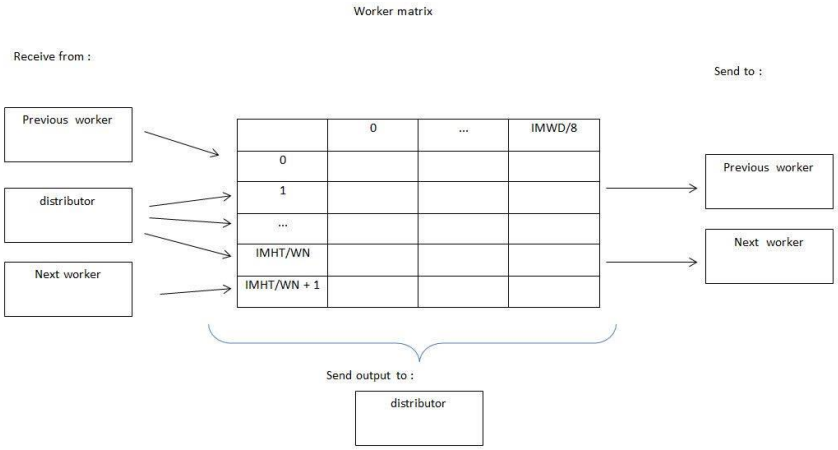
In the beginning, when the packing occurs, the generator allows us to choose whether we wish to load the data from the input picture or, alternatively, to create random binary values to be stored. The latter option reduces the overall loading time, which was used for bigger images.



The timer thread is used to calculate the elapsed time after one processed round. It can also be used to determine how long it takes to read or to output the image.

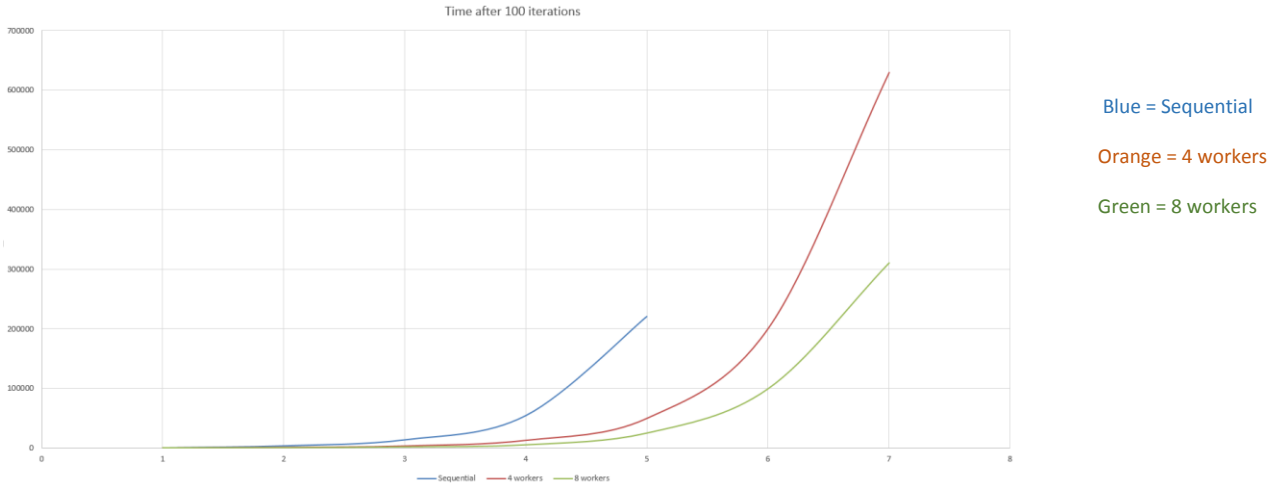
The distributor thread dictates the actions of all the other threads. It receives all the data from DataInStream and then shares it between the workers, after it receives an input from the SW1 button. The distributers also tells the timer when to start, pause and resume and it checks for inputs from the accelerator and the SW2 button and gives the workers the corresponding commands, to either keep working(0), send the data for output(1) or print the status report(2).

The worker thread is used to process the image. Each worker has a matrix containing a row received from the previous worker, the rows received from the distributor and a row received from the next worker. The first processed row is stored in “prossedRow”. The following processed rows are then each stored in the matrix’s first row, which is no longer needed, the previous row is updated with “processedRow” and, then, “processedRow” is updated with the first row.



Andrei Alexandru : aa17956@my.bristol.ac.uk
Erich Reinholtz : er17660@my.bristol.ac.uk

Tests and Experiments



Time efficiency comparison between different approaches, 100 iterations.

Sequential approach:

	Time after one iteration	Time after 100 iterations
16x16	4.64	464
64x64	37	3700
128x128	138	13800
256x256	545	54500
512x512	2210	221000
1024x1024	N/A	N/A
1776x1776	N/A	N/A

4 Workers:

	Time after one iteration	Time after 100 iterations
16x16	0.32	32
64x64	8	800
128x128	30	3000
256x256	125	12500
512x512	495	49500
1024x1024	2000	200000
1776x1776	6300	630000

8 Workers:

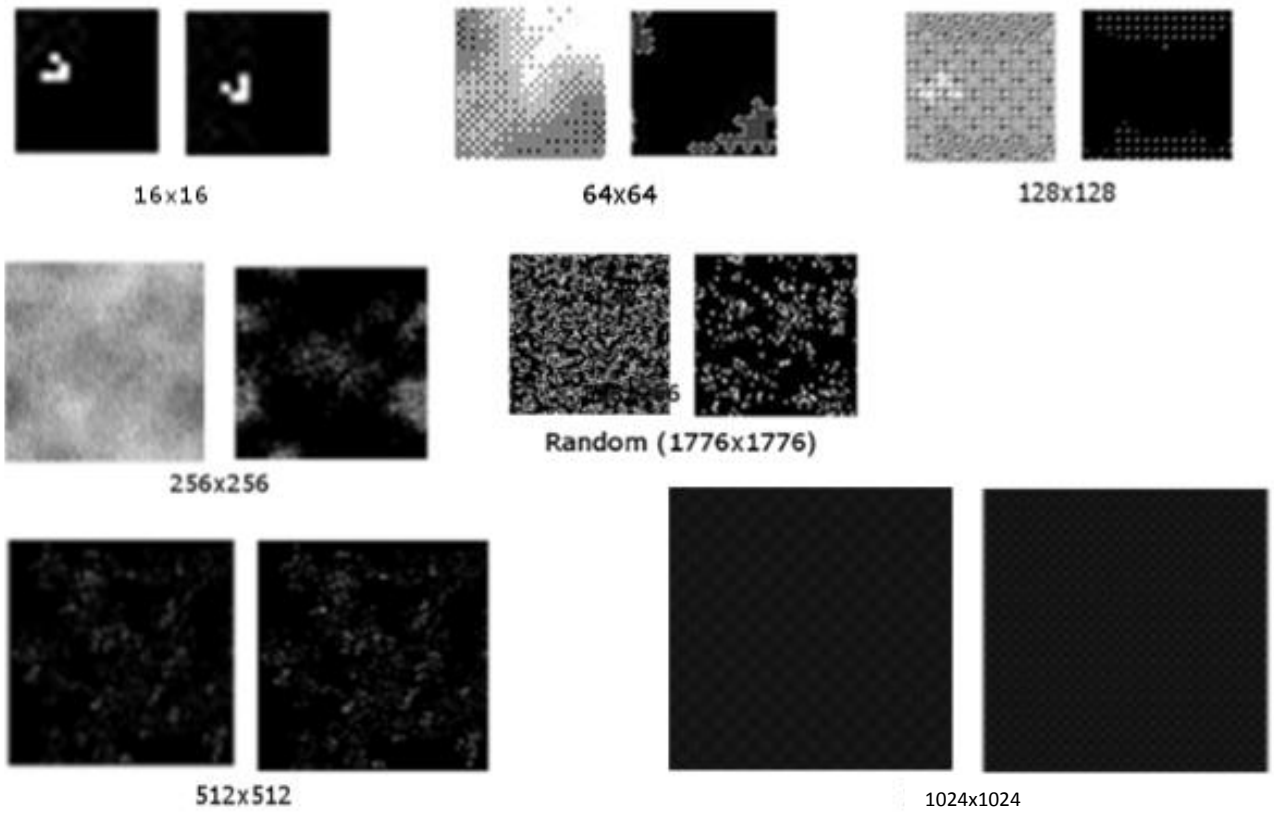
	Time after one iteration	Time after 100 iterations
16x16	0.25	25
64x64	3.78	378
128x128	15	1500
256x256	50	5000
512x512	247	24700
1024x1024	990	99000
1776x1776	3100	310000

We tried different experiments with different number of workers and different images sizes. We concluded that our fastest version is with 8 workers, which is twice the speed when using only 4 workers. Another observation is that the use of workers on different tiles offers the possibility to use images bigger than 512x512 and, comparing to a sequantial system, our system is more than 10 times faster.

One of the first things we tried to use were global variables. We tried to store the image on two different global variables and have the workers process them. We soon found out that it was not possible to use them in an concurrent system. This forced us to do more research and find better methods to create a true concurent system and to create efficient communication between threads. One of the biggest challanges was creating a stable communication system between the workers. In the end, we connected each worker with two channels, one for the next and one for the previous worker.

Another important change we made was the method for storing the matrix. Initially, we stored the image in an [2][IMHT/WN][IMWD/8] matrix. After experimenting with different implementations, we finished the current design, where we only use a matrix of [IMHT/WN][IMWD/8], which increased our memory capacity by 50% and our speed by 12%.

Andrei Alexandru : aa17956@my.bristol.ac.uk
Erich Reinholtz : er17660@my.bristol.ac.uk



In the above images, there are presented the original input images(on the left) and the output image after 2 rounds(on the right). The only exception is "Random (1776x1776)". It presents a random image created with generator after 2 rounds(on the left) and 100 rounds (on the right).

To find our memory capacity, we kept generating random images with different sizes until we found the limit. After a series of different tests, we concluded that our maximum input image size accepted is 1776x1776.

Andrei Alexandru : aa17956@my.bristol.ac.uk
Erich Reinholtz : er17660@my.bristol.ac.uk

Critical Analysis

The maximum input accepted is an image of 1776x1776, which can be processed at a speed of 3100 milliseconds per iteration, with 8 workers. Although we are proud of this achievement, we are aware that there is a lot of room for optimisation.

Regarding the loading and output time, these could be improved if they were distributed to the workers in some way. The processing time could be improved, especially in larger images, if we could check is a pack is full of dead cells. That way, we could just store the pack as a 0 and save time and space.

We also experimented with different bit pack sizes, to try to reduce the memory usage, but we concluded there is no real difference. Furthermore, it's possible to reduce the memory usage if we can reduce the size of the worker's arrays.

Because of lack of time and research, we weren't able to implement streaming channels. Using them could greatly improve the communication efficiency and speed between the distributor and the workers. In our current system, the distributor and the workers have to wait for the other channel's end to receive the data before communicating with the next thread. Streaming channels would free both ends of the channels from waiting and would increase the speed considerably.

Another method for optimising communication would be interfaces. An interface specifies a set of transaction types, which connect two tasks, a client and a server. This could offer better communication between threads and, possibly, better time.