Università di Pisa

Distributed systems: paradigms and models

Median Filter – Multi-core Implementation

Antonio Mallia – 423458

me@antoniomallia.it

Abstract

This software is the parallel implementation of a median filter over images. Realized with two different skeletons and two implementations for each one. Automatic tests have been realized to obtain performance parameters. Results in terms of performance are in line with the expectation. Future improvements are possible.

1 Introduction

This report summarizes the work done to realize a software with the main purpose of reduction salt-and-pepper noise from images. It will be presented the phases of the algorithm identification, the parallel design choices, the implementation and the performance evaluation. Finally a conclusion with possible future enhancements is proposed.

In the appendix it is possible to examine a small documentation of the software usage to replicate the tests conducted.

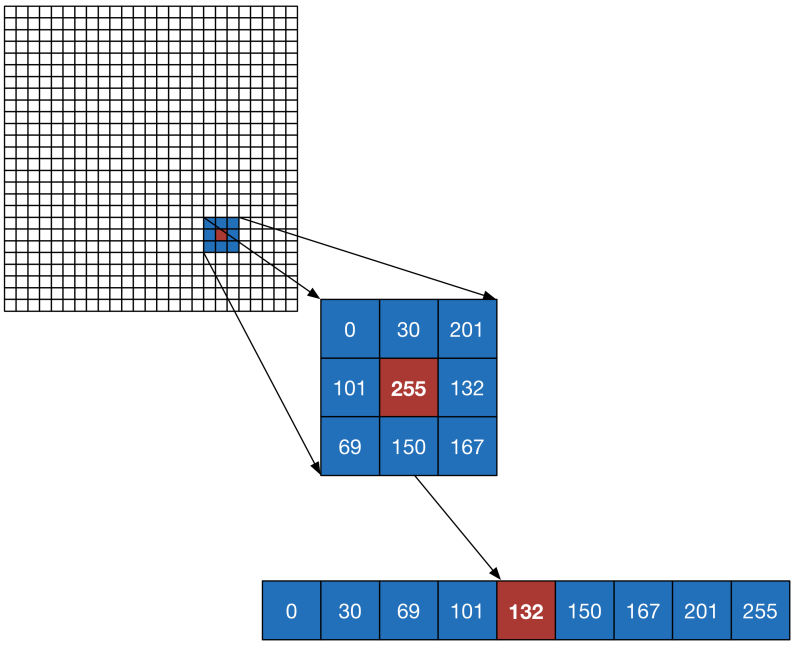
2 Design

The very first design stage has been the identification of the algorithm in charge of the execution of the median filter, with the aim of removing salt-and-pepper noise. The algorithm, in the first attempt, was engineered in a sequential form, to work on the whole matrix.

2.1 Algorithm

The image, previously converted to a two dimensional matrix of integers, is scanned element-by-element, so pixel-by-pixel. For each element, some neighbours are collected. This stencil used is typically a square (also referred as window) where the analysed pixel represents the central one. The window can have several sizes; usually the side is an odd number. For this project a fixed window has been chosen of size 3x3.

The pixel values of the window are inserted in an array, which is then sorted and the median value is taken as a substitute for the analysed pixel. A new matrix is constituted with the new values to avoid interferences.



```java
/**
	 * This method applies the median filter algorithm to the current matrix
	 */
	public void medianFilter() {
		int[][] denoisedMatrix = new int[getHeight()][getWidth()];
		// The window side length (square window)
		int windowSize = 3;
		// window area
		int window = windowSize * windowSize;
		// padding from the center
		int edge = windowSize / 2;

		Color[] pixel = new Color[window];
		// three arrays used for RGB colors
		int[] r = new int[window];
		int[] g = new int[window];
		int[] b = new int[window];

		// for each pixel in the matrix (border excluded)
		for (int i = edge; i < this.getHeight()  - edge; i++) {
			for (int j = edge; j < this.getWidth()- edge; j++) {
				// get all pixel values in the window
				for (int k = 0; k < windowSize; k++) {
					for (int l = 0; l < windowSize; l++) {
						pixel[k * windowSize + l] = new Color(matrix[i + k
								- edge][j + l - edge]);
					}
				}
				// Decompose the colors in RGB
				for (int k = 0; k < window; k++) {
					r[k] = pixel[k].getRed();
					g[k] = pixel[k].getGreen();
					b[k] = pixel[k].getBlue();
				}
				// Sort
				Arrays.sort(r);
				Arrays.sort(g);
				Arrays.sort(b);
				// Get the median value and generate the color
				denoisedMatrix[i][j] = new Color(r[window / 2], g[window / 2],
						b[window / 2]).getRGB();
			}
		}
		matrix = denoisedMatrix;
	}
```

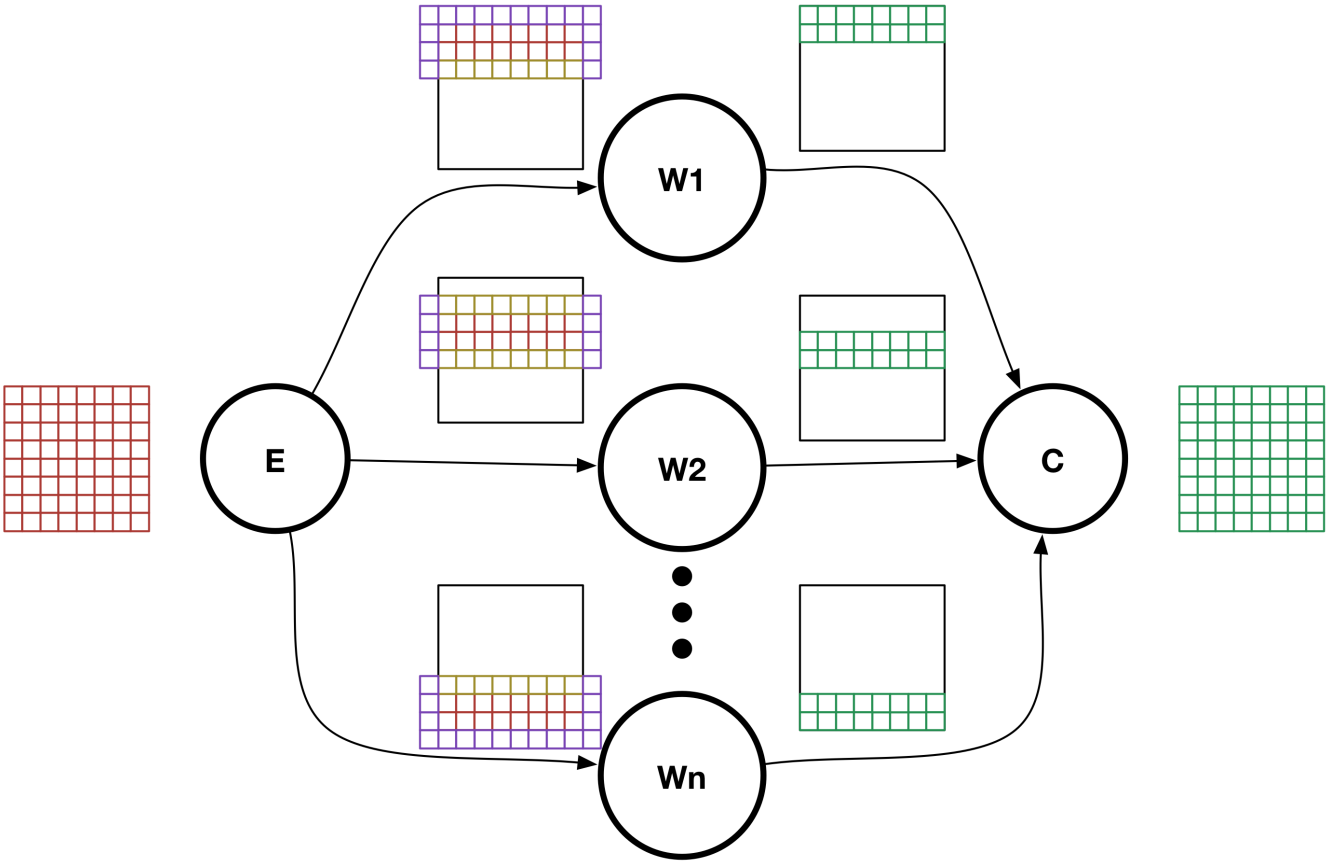
2.2 Parallel Model

Starting from the sequential algorithm, two different parallel models have been individuated: the Map and the Farm models.

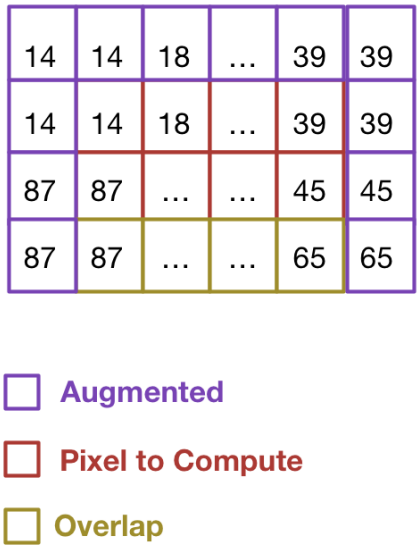
2.2.1 Map model

The map model contemplates the split of the noisy image matrix in a number of chunks (sub-matrices). The split work is row based. This number depends on the available workers. Each of them is in charge of just one sub-matrix, all the workers operates in parallel.

When the workers terminate the execution of the median filter, the fractions of the matrix have to be merged back, producing the denoised version of the original image.



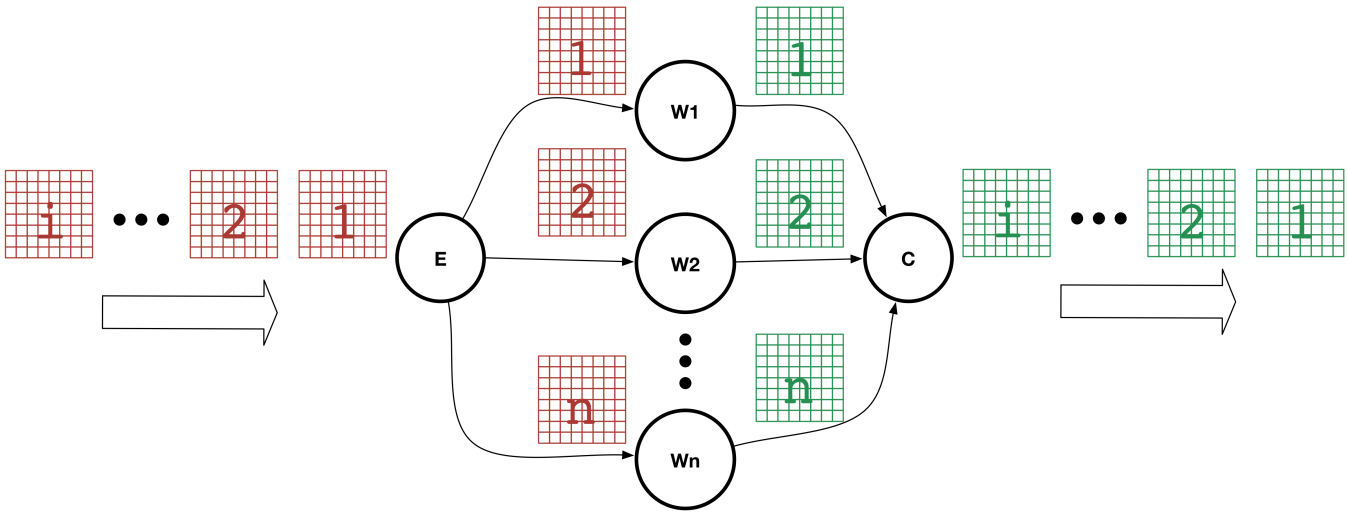
It is important to notice that the split of the matrix has to handle an overlap between chunks and generate an augmented matrix with dummy borders of replicated values. This lets the algorithm be able to compute also the edge pixels, obtaining same results of the sequential algorithm. In the merge phase only the computed pixels are actually merged and the augmented and overlapping ones are discarded.



An alternative could be to split in squared sub-matrices, each with side length of the original matrix size divided by the number of workers, obtaining more than one sub-matrix for each worker. This could lead to a better balancing between the workers in case of not homogeneous noise, but for sure an higher overall delay and a larger overlapping between the sub-matrices.

2.2.2 Farm model

The Farm model, on the other side, is a coarse grain computation respect to the Map model. Indeed every worker is in charge of the computation for the whole matrix, but, considering a stream of several matrices, it is possible to spread them in parallel as many as the number of the available workers.



3 Implementation

The software has been implemented using Java 8; in particular both the Map and the Farm skeletons have a double implementation using Skandium and the new stream features of Java.

Skandium is a framework not supported anymore, even if it still works properly. It is interesting to notice how the new stream features allow obtaining similar results without the need of an external library.

3.1 Sequential

The sequential implementations, which is very basic, is based on the augmentation of the original matrix containing a replication of the edge pixels, the execution of the median filter and then the removal of the not computed edge pixels.

This implementation has been used to compare the results with the parallel ones and to calculate performance parameters.

3.2 Skandium Map

Skandium Map wraps the *Map* skeleton provided by Skandium. It needs the implementations of the *Split*, *Merge* and *Execute* classes. The algorithms of these classes are shared with the ones used in the Java 8 Map.

In both Skandium and Java 8 Map implementation the edge pixels (useful for filter execution on the border) are added and removed chuck-by-chunk and not for the whole matrix.

The Split implementation generates, starting for a single matrix, new sub-matrix objects which are as well of type Matrix. The division is row based, the all width of the matrix is taken; the height instead is divided by the number of workers expect for the last one which could be a bit longer in case of oddment. Another implementation could have been coded without generating new sub-matrices objects, but returning only the coordinates representing the sub-matrix portion since the original matrix is in a shared memory environment.

The merge implementation is just an ordered concatenation of the split chunks.

3.3 Java 8 Map

Java 8 Map uses the new stream features, in particular the map function. The input matrix is first split and then, using the Java Arrays class, it is converted to a parallel stream. The stream is mapped in parallel applying the filter algorithm and reduced to an array of sub-matrices. They are, finally, folded using the merge algorithm.

A dedicated *ForkJoinPool* is used instead of the "*ForkJoinPool.common.parallelism*".

3.4 Skandium Farm

In the Skandium Farm implementation no split nor merge is needed, indeed each worker execute the filter algorithm on the whole matrix. The execute class is the same used by the Skandium *Map* implementation.

Also in this case and in the Java 8 Farm implementation, each matrix of the array is first augmented and after diminished for the correct filter execution on the edges.

3.5 Java 8 Farm

In the Java 8 Farm implementation the array of matrices is converted to a parallel stream, each of them is mapped to the filter executor function and, finally, the computed stream saved back to an array.

3.6 Test Executor

To execute automatic tests for each implementation, the class *Test* have to be extended defining the *compute()* and the *shutdown()* methods. The class *TestExec*, which contains a static main method, will instantiate them, execute tests and collect results.

When all the tests are finished, the generation of the excel results file is requested through the method *generate()* of the *ExcelGenerator* class.

4 Results

Results are collected automatically in an Excel spreadsheet using the auxiliary Apache POI library. Then it is possible to easily manipulate the data to get some visually effective graphs or to export them.

For the performance evaluations, test are executed automatically on random generated matrices, results of each execution is collected in a Java bean and then all the "experiments" properly sorted and printed in the excel file.

The software has been tested on a dual AMD Opteron(tm) Processor 6176, with in total 24 cores and on an instance of AWS EC2 c4.8xlarge with 36 vCPUs. Complete data collected from tests executed and graphs obtained can be found in the appendix. Only the tests executed on the AMD Processor will be presented for reason of brevity.

All the skeletons implementations have been tested on three different matrix sizes (100x100, 500x500 and 1000x1000), for each matrix size three streams size are generated (1, 50 and 100).

The Map skeleton service time expected is estimated using the formula

$$T_s(n) = T_e + \frac{T_w}{n} + T_c$$

where T_e, T_w, T_c are respectively time to split, compute and merge. This formula can be approximated to $\frac{T_w}{n}$ when the split and merge are significantly smaller then the compute time. From the data collected from the test with one single matrix, it is evident that this is true in the coarse grain computation and partially true for the fine grain with bigger matrices. In fact, the smallest size matrix reaches the asymptote (probably the time spent computing the serial fraction) at 4 cores, dramatically reducing the scalability and consequentially the efficiency. Predictably, scalability and efficiency of this model turn even better with bigger streams of matrices.

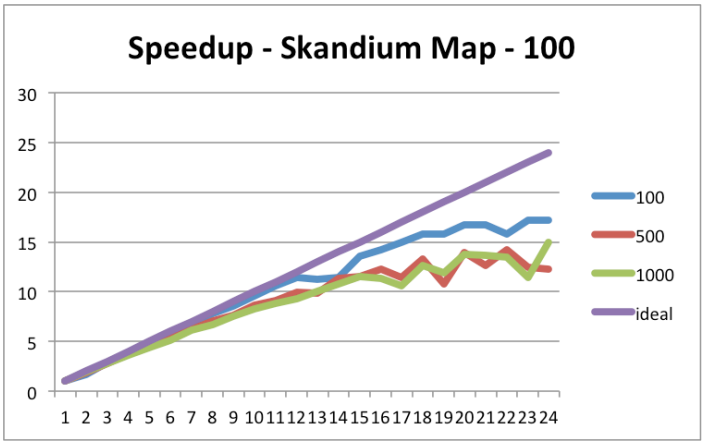
Another interesting aspect is that both the map implementations reached very close results in the single matrix test, the Java 8 stream implementation, on the other hand, goes a bit lower in the many-matrices test.

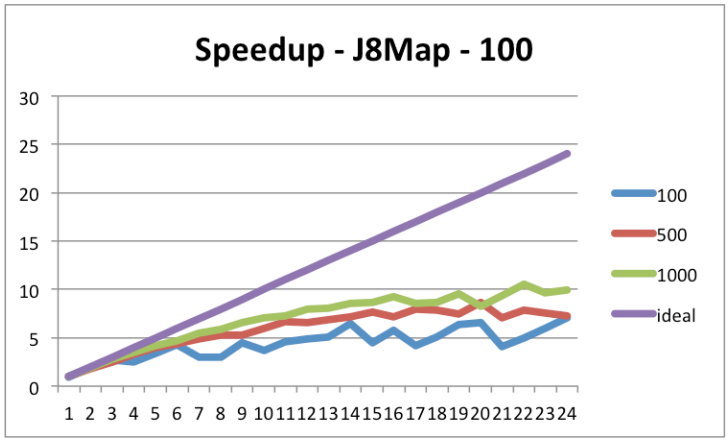
Skandium Map – 1 Matrix

Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	19	169	595
2	14	136	302
3	8	79	206
4	5	59	198
5	4	65	134
6	4	39	117
7	4	44	133
8	3	45	93
9	4	32	122
10	3	37	76
11	4	26	76
12	5	27	69
13	5	27	67
14	4	25	66
15	5	25	58
16	4	35	73
17	3	18	66
18	3	20	62
19	3	20	64
20	2	21	57
21	5	26	54
22	4	22	55
23	3	20	89
24	3	23	59

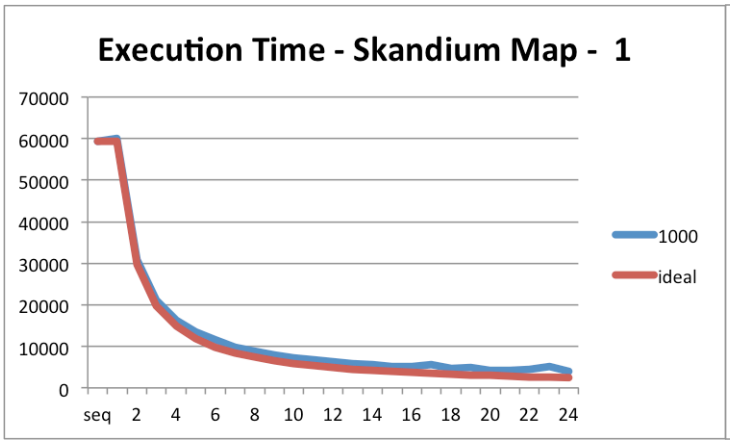
Java 8 Map – 1 Matrix

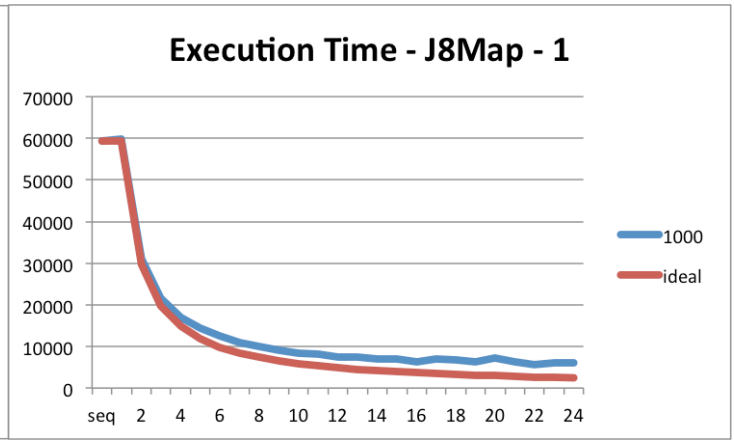
Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	14	150	601
2	6	76	302
3	4	80	212
4	4	39	166
5	4	36	138
6	3	40	112
7	20	41	109
8	3	57	94
9	2	26	81
10	2	22	80
11	3	29	72
12	5	26	71
13	4	17	111
14	3	16	90
15	5	31	67
16	2	18	66
17	2	22	67
18	2	18	90
19	3	17	78
20	4	19	53
21	2	15	54
22	3	16	82
23	3	21	76
24	5	18	61





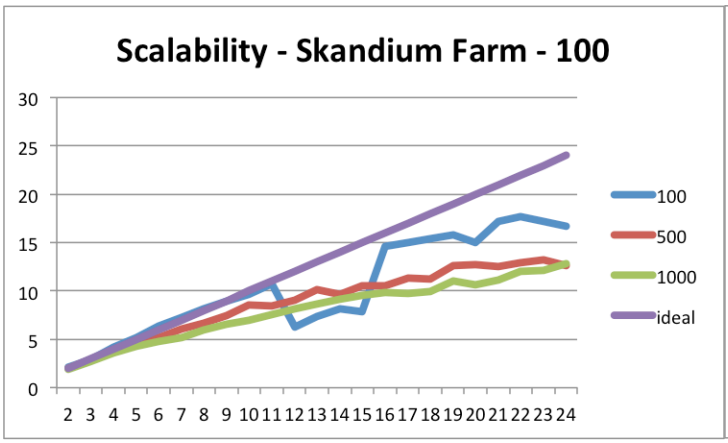
The graphs below shows a comparison of the results obtained using Skandium e Java 8 Map on a 1000 x 1000 matrix. The ideal value is calculated dividing the sequential execution time by the parallelism degree. It is even more evident the performance differences between the two implementations.

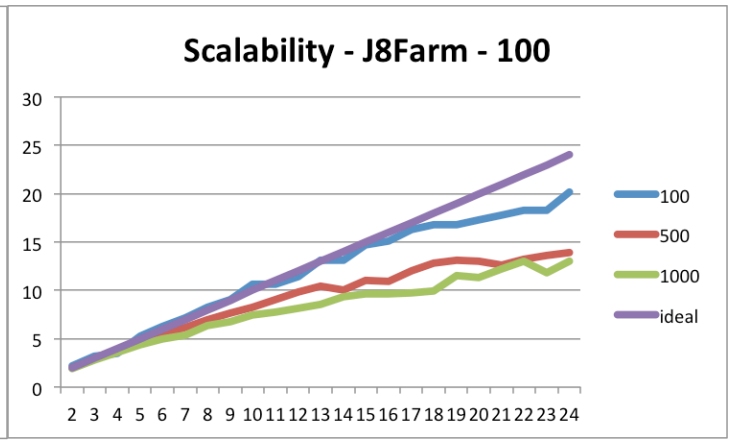




The Farm model is, obviously, totally useless with the single matrix stream and, so, only the many matrices test will be analysed.

The execution time of the Skandium Map and the Skandium Farm are comparable for both the 50 and 100 matrices tests, slightly better for the Java 8 stream one, leading to a very linear scalability and better efficiency.





The service time expected for the Farm model is estimated using the following formula

$$T_s(n) = max\{T_e, \frac{T_w}{n}, T_c\}$$

while the completion time expected is

$$T_c(n) = n * T_e + \frac{m * T_w}{n} + T_c$$

where the components of the formula are respectively the time to fill the farm, compute and collect.

5 Conclusion

Even if the software is just for didactic purpose, it actually works pretty well on real images. A comparison with a very well known image-processing library (OpenCV) has been done and the results are almost identical for quality obtained.

Compared to other libraries this software can scale on multi-core architectures and, taking the advantages of a farm skeleton, be executed on an entire folder of images.

Original Image OpenCV MedianFilter

Some improvements for future releases are possible, both functional and performance ones.

1. Dynamic windows size: the software can be modified to accept as a parameter an extra integer (possibly odd) representing the window size. At the moment the side size is fixed, but increasing it a greater blur effect can be obtained, which can be useful depending on the noise level.
2. Noise identification: it can be introduced a two-step pipeline, the first to detect the noisy pixels and the second to compute just these ones, probably obtaining a better output. The detect step should output a matrix of the same size of the original image, made of just Boolean values used to define if the correspondent pixel is noisy or not. To detect if a pixel is considered noisy, the algorithm is similar to the filter one, you get the median value from the window and compare with the actual value. If there is a huge difference then it is a noisy pixel and you mark as true the value at the correspondent coordinates. The second stage of the pipeline could be implemented as a Map, but this will make difficult to split the matrix in a computationally balanced way since noisy pixel could be not homogeneously distributed. To simplify, it could be implemented with a sort of particular farm, which takes the input matrix, transforms it in a stream of fine-grained sub-matrices (at least single pixel with neighbourhoods) and then route them to one of the available worker. Finally, the collector has to merge all the sub-matrices and output only when a matrix is

built. In case of single pixel (with neighbourhoods) matrix stream, the collector could route them directly to the collector if they are not noisy ones.

3. Generalize the J8Map and J8Farm implementation: these classes have been realized to simplify the usage of the Java 8 stream features. It would be interesting to generalize them using interfaces for the Split, Merge, Execute class, to be able to reuse the skeleton, for example, to apply other kind of filter to images.

A Appendix

A.1 Usage

The software can be compiled and packed automatically using Apache Ant.

Several targets are defined in the build.xml file, but the default is jar which package the software in a jar file ready to be run.

```
tar -xvzf medianFilter.tar.gz
cd medianFilter
ant
```

A jar file named MedianFilter.jar will be generated in the dist/ folder.

```
java –jar dist/MedianFilter.jar
```

The above command will show the following helper menu.

```
__  __           _ _                 _____ _ _ _
|  \/  | ___  __| (_) __ _ _ __ |  ___(_) |_ ___ _ __
| |\/| |/ _ \/ _` | |/ _` | '_ \| |_  | | | __/ _ \ '__|
| |  | |  __/ (_| | | (_| | | | |  _| | | | ||  __/ |
|_|  |_|\___|\__,_|_|\__,_|_| |_|_|   |_|_|\__\___|_|
usage: MedianFilter -d <200,400,1000...> [-h] -s <1,10,50...> -t <4>
 -d,--dim <200,400,1000...>   matrix dimension in pixel
 -h,--help                    print help message
 -s,--stream <1,10,50...>     number of matrices to test
 -t,--threads <4>             threads number
```

- -d, --dim : it allows defining the sizes of the (square) matrices generated for the tests. Several sizes can be provided using a comma separated format
- -s, --stream : it allows defining the number of matrices in the input stream. Several stream numbers can be provided using a comma-separated format.
- -t, --thread : it allows specifying the maximum number of threads to use.

The helper menu and the parsing of the parameters is done using Apache Commons Cli library.

When the test starts executing a log files is kept named execution.log in folder logs/ and, if run in foreground, the log is printed in the console. The log is generated using the Apache Log4J2 library. A short example of the output is show below:

```
2015-08-27 23:03:01,982 INFO  - Performance Test started
2015-08-27 23:03:01,984 INFO  - Executing Test ==> Type: Sequential   Thread number: 0    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,512 INFO  - Execution time: 21 ms
2015-08-27 23:03:02,515 INFO  - Executing Test ==> Type: Sequential   Thread number: 0    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,528 INFO  - Execution time: 8 ms
2015-08-27 23:03:02,533 INFO  - Executing Test ==> Type: Skandium-MapReduce     Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,553 INFO  - Execution time: 18 ms
2015-08-27 23:03:02,554 INFO  - Executing Test ==> Type: Skandium-MapReduce     Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,578 INFO  - Execution time: 22 ms
2015-08-27 23:03:02,580 INFO  - Executing Test ==> Type: Skandium-Farm          Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,586 INFO  - Execution time: 5 ms
2015-08-27 23:03:02,586 INFO  - Executing Test ==> Type: Skandium-Farm          Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,593 INFO  - Execution time: 4 ms
2015-08-27 23:03:02,595 INFO  - Executing Test ==> Type: J8-MapReduce           Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,608 INFO  - Execution time: 12 ms
2015-08-27 23:03:02,608 INFO  - Executing Test ==> Type: J8-MapReduce           Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,622 INFO  - Execution time: 8 ms
2015-08-27 23:03:02,625 INFO  - Executing Test ==> Type: J8-Farm      Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,634 INFO  - Execution time: 7 ms
2015-08-27 23:03:02,634 INFO  - Executing Test ==> Type: J8-Farm      Thread number: 1    Matrices number: 1  Matrix size: 100 x 100
2015-08-27 23:03:02,641 INFO  - Execution time: 6 ms
```

A.2 Documentation

Javadoc documentation can be generated using the command

```
ant javadoc
```

The documentation will be placed under the folder "/doc"

A.3 Data collected and graphs

Skandium Map

Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	19	169	595
2	14	136	302
3	8	79	206
4	5	59	198
5	4	65	134
6	4	39	117
7	4	44	133
8	3	45	93
9	4	32	122
10	3	37	76
11	4	26	76
12	5	27	69
13	5	27	67
14	4	25	66
15	5	25	58
16	4	35	73
17	3	18	66
18	3	20	62
19	3	20	64
20	2	21	57
21	5	26	54
22	4	22	55
23	3	20	89
24	3	23	59

Speedup				
Stream size: 1	100	500	1000	ideal
seq				
1	0,68	1,02	1	1
2	0,93	1,26	1,98	2
3	1,62	2,18	2,9	3
4	2,6	2,92	3,02	4
5	3,25	2,65	4,46	5
6	3,25	4,41	5,1	6
7	3,25	3,91	4,49	7
8	4,33	3,82	6,42	8
9	3,25	5,38	4,89	9
10	4,33	4,65	7,86	10
11	3,25	6,62	7,86	11
12	2,6	6,37	8,65	12
13	2,6	6,37	8,91	13
14	3,25	6,88	9,05	14
15	2,6	6,88	10,29	15
16	3,25	4,91	8,18	16
17	4,33	9,56	9,05	17
18	4,33	8,6	9,63	18
19	4,33	8,6	9,33	19
20	6,5	8,19	10,47	20
21	2,6	6,62	11,06	21
22	3,25	7,82	10,85	22
23	4,33	8,6	6,71	23
24	4,33	7,48	10,12	24

Scalability				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	1,36	1,24	1,97	2
3	2,38	2,14	2,89	3
4	3,8	2,86	3,01	4
5	4,75	2,6	4,44	5
6	4,75	4,33	5,09	6
7	4,75	3,84	4,47	7
8	6,33	3,76	6,4	8
9	4,75	5,28	4,88	9
10	6,33	4,57	7,83	10
11	4,75	6,5	7,83	11
12	3,8	6,26	8,62	12
13	3,8	6,26	8,88	13
14	4,75	6,76	9,02	14
15	3,8	6,76	10,26	15
16	4,75	4,83	8,15	16
17	6,33	9,39	9,02	17
18	6,33	8,45	9,6	18
19	6,33	8,45	9,3	19
20	9,5	8,05	10,44	20
21	3,8	6,5	11,02	21
22	4,75	7,68	10,82	22
23	6,33	8,45	6,69	23
24	6,33	7,35	10,08	24

Efficiency				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	0,46	0,63	0,99	1
3	0,54	0,73	0,97	1
4	0,65	0,73	0,75	1
5	0,65	0,53	0,89	1
6	0,54	0,74	0,85	1
7	0,46	0,56	0,64	1
8	0,54	0,48	0,8	1
9	0,36	0,6	0,54	1
10	0,43	0,46	0,79	1
11	0,3	0,6	0,71	1
12	0,22	0,53	0,72	1
13	0,2	0,49	0,69	1
14	0,23	0,49	0,65	1
15	0,17	0,46	0,69	1
16	0,2	0,31	0,51	1
17	0,25	0,56	0,53	1
18	0,24	0,48	0,53	1
19	0,23	0,45	0,49	1
20	0,33	0,41	0,52	1
21	0,12	0,32	0,53	1
22	0,15	0,36	0,49	1
23	0,19	0,37	0,29	1
24	0,18	0,31	0,42	1

Stream size: 50	100	500	1000
seq	296	7524	29660
1	298	7578	30707
2	151	3845	16082
3	101	2520	10572
4	75	1999	7943
5	64	1632	6370
6	52	1315	5738
7	44	1238	4899
8	39	1184	4283
9	36	896	3862
10	32	832	3501
11	30	763	3404
12	27	749	3114
13	25	797	3024
14	25	625	2740
15	26	695	2773
16	25	583	2370
17	22	604	2271
18	23	696	2252
19	21	566	2228
20	21	490	2343
21	20	514	2480
22	21	498	2063
23	22	466	1972
24	24	458	2475

Stream size: 50	100	500	1000	ideal
seq				
1	0,99	0,99	0,97	1
2	1,96	1,96	1,84	2
3	2,93	2,99	2,81	3
4	3,95	3,76	3,73	4
5	4,62	4,61	4,66	5
6	5,69	5,72	5,17	6
7	6,73	6,08	6,05	7
8	7,59	6,35	6,93	8
9	8,22	8,4	7,68	9
10	9,25	9,04	8,47	10
11	9,87	9,86	8,71	11
12	10,96	10,05	9,52	12
13	11,84	9,44	9,81	13
14	11,84	12,04	10,82	14
15	11,38	10,83	10,7	15
16	11,84	12,91	12,51	16
17	13,45	12,46	13,06	17
18	12,87	10,81	13,17	18
19	14,1	13,29	13,31	19
20	14,1	15,36	12,66	20
21	14,8	14,64	11,96	21
22	14,1	15,11	14,38	22
23	13,45	16,15	15,04	23
24	12,33	16,43	11,98	24

Stream size: 50	100	500	1000	ideal
seq				
1				
2	1,97	1,97	1,91	2
3	2,95	3,01	2,9	3
4	3,97	3,79	3,87	4
5	4,66	4,64	4,82	5
6	5,73	5,76	5,35	6
7	6,77	6,12	6,27	7
8	7,64	6,4	7,17	8
9	8,28	8,46	7,95	9
10	9,31	9,11	8,77	10
11	9,93	9,93	9,02	11
12	11,04	10,12	9,86	12
13	11,92	9,51	10,15	13
14	11,92	12,12	11,21	14
15	11,46	10,9	11,07	15
16	11,92	13	12,96	16
17	13,55	12,55	13,52	17
18	12,96	10,89	13,64	18
19	14,19	13,39	13,78	19
20	14,19	15,47	13,11	20
21	14,9	14,74	12,38	21
22	14,19	15,22	14,88	22
23	13,55	16,26	15,57	23
24	12,42	16,55	12,41	24

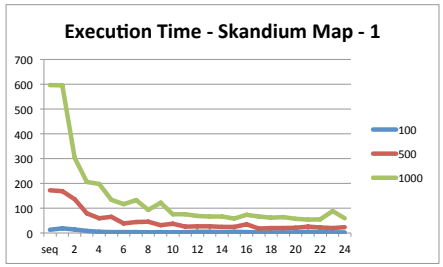
Stream size: 50	100	500	1000	ideal
seq				
1				
2	0,98	0,98	0,92	1
3	0,98	1	0,94	1
4	0,99	0,94	0,93	1
5	0,93	0,92	0,93	1
6	0,95	0,95	0,86	1
7	0,96	0,87	0,86	1
8	0,95	0,79	0,87	1
9	0,91	0,93	0,85	1
10	0,93	0,9	0,85	1
11	0,9	0,9	0,79	1
12	0,91	0,84	0,79	1
13	0,91	0,73	0,75	1
14	0,85	0,86	0,77	1
15	0,76	0,72	0,71	1
16	0,74	0,81	0,78	1
17	0,79	0,73	0,77	1
18	0,71	0,6	0,73	1
19	0,74	0,7	0,7	1
20	0,7	0,77	0,63	1
21	0,7	0,7	0,57	1
22	0,64	0,69	0,65	1
23	0,58	0,7	0,65	1
24	0,51	0,68	0,5	1

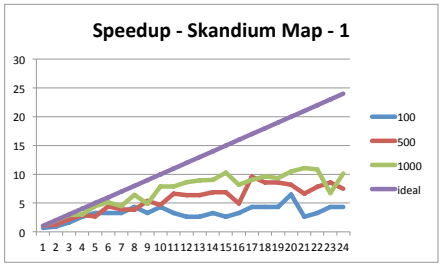
Stream size: 100	100	500	1000
seq	584	14879	59286
1	598	15004	60117
2	343	7856	30812
3	200	5331	21235
4	149	3959	16239
5	120	3309	13556
6	100	2661	11561
7	86	2409	9673
8	75	2112	8868
9	68	1958	7828
10	61	1721	7175
11	55	1640	6755
12	51	1499	6382
13	52	1508	5904
14	51	1312	5524
15	43	1291	5160
16	41	1214	5220
17	39	1301	5618
18	37	1121	4698
19	37	1377	4974
20	35	1071	4315
21	35	1180	4327
22	37	1047	4392
23	34	1194	5210
24	34	1213	3970

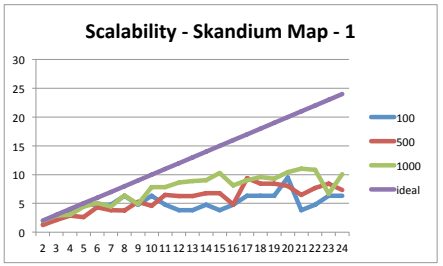
Stream size: 100	100	500	1000	ideal
seq				
1	0,98	0,99	0,99	1
2	1,7	1,89	1,92	2
3	2,92	2,79	2,79	3
4	3,92	3,76	3,65	4
5	4,87	4,5	4,37	5
6	5,84	5,59	5,13	6
7	6,79	6,18	6,13	7
8	7,79	7,04	6,69	8
9	8,59	7,6	7,57	9
10	9,57	8,65	8,26	10
11	10,62	9,07	8,78	11
12	11,45	9,93	9,29	12
13	11,23	9,87	10,04	13
14	11,45	11,34	10,73	14
15	13,58	11,53	11,49	15
16	14,24	12,26	11,36	16
17	14,97	11,44	10,55	17
18	15,78	13,27	12,62	18
19	15,78	10,81	11,92	19
20	16,69	13,89	13,74	20
21	16,69	12,61	13,7	21
22	15,78	14,21	13,5	22
23	17,18	12,46	11,38	23
24	17,18	12,27	14,93	24

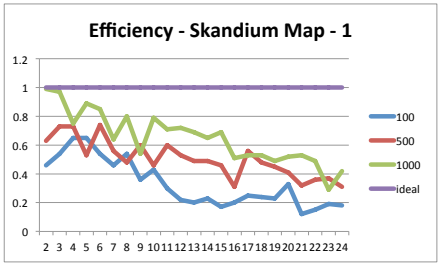
Stream size: 100	100	500	1000	ideal
seq				
1				
2	1,74	1,91	1,95	2
3	2,99	2,81	2,83	3
4	4,01	3,79	3,7	4
5	4,98	4,53	4,43	5
6	5,98	5,64	5,2	6
7	6,95	6,23	6,21	7
8	7,97	7,1	6,78	8
9	8,79	7,66	7,68	9
10	9,8	8,72	8,38	10
11	10,87	9,15	8,9	11
12	11,73	10,01	9,42	12
13	11,5	9,95	10,18	13
14	11,73	11,44	10,88	14
15	13,91	11,62	11,65	15
16	14,59	12,36	11,52	16
17	15,33	11,53	10,7	17
18	16,16	13,38	12,8	18
19	16,16	10,9	12,09	19
20	17,09	14,01	13,93	20
21	17,09	12,72	13,89	21
22	16,16	14,33	13,69	22
23	17,59	12,57	11,54	23
24	17,59	12,37	15,14	24

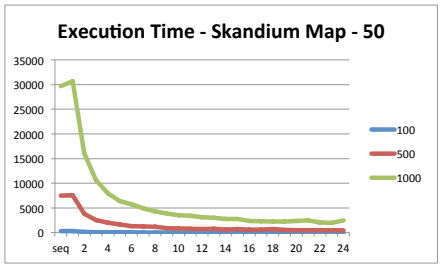
Stream size: 100	100	500	1000	ideal
seq				
1				
2	0,85	0,95	0,96	1
3	0,97	0,93	0,93	1
4	0,98	0,94	0,91	1
5	0,97	0,9	0,87	1
6	0,97	0,93	0,85	1
7	0,97	0,88	0,88	1
8	0,97	0,88	0,84	1
9	0,95	0,84	0,84	1
10	0,96	0,86	0,83	1
11	0,97	0,82	0,8	1
12	0,95	0,83	0,77	1
13	0,86	0,76	0,77	1
14	0,82	0,81	0,77	1
15	0,91	0,77	0,77	1
16	0,89	0,77	0,71	1
17	0,88	0,67	0,62	1
18	0,88	0,74	0,7	1
19	0,83	0,57	0,63	1
20	0,83	0,69	0,69	1
21	0,79	0,6	0,65	1
22	0,72	0,65	0,61	1
23	0,75	0,54	0,49	1
24	0,72	0,51	0,62	1

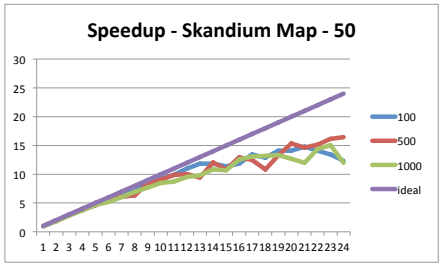


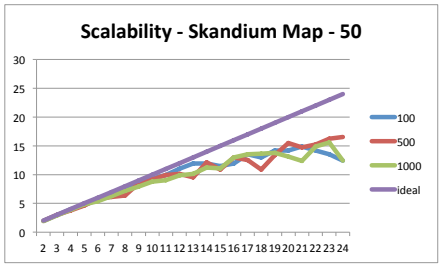


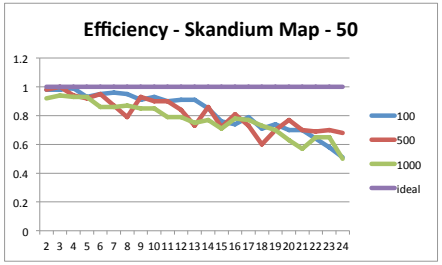


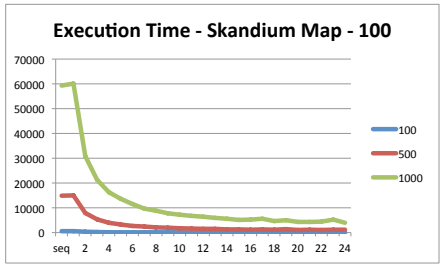


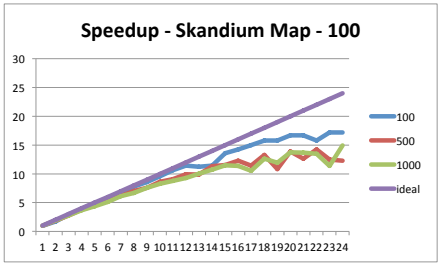


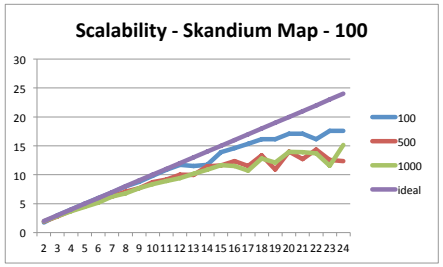


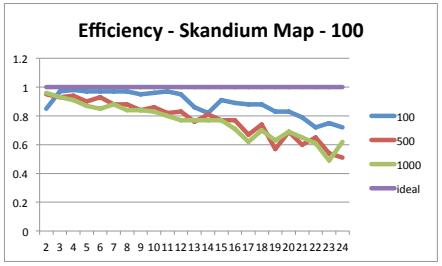












Each graph represents different test conducted. Each row contains the graph of the time spent, the speedup, the scalability and the efficiency. First row is executed on a single matrix stream, second and third on 50 and 100 matrix stream respectively. The size of the matrix is indicated in the legend of the graph

Java 8 Map

Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	14	150	601
2	6	76	302
3	4	80	212
4	4	39	166
5	4	36	138
6	3	40	112
7	20	41	109
8	3	57	94
9	2	26	81
10	2	22	80
11	3	29	72
12	5	26	71
13	4	17	111
14	3	16	90
15	5	31	67
16	2	18	66
17	2	22	67
18	2	18	90
19	3	17	78
20	4	19	53
21	2	15	54
22	3	16	82
23	3	21	76
24	5	18	61

Speedup				
Stream size: 1	100	500	1000	ideal
seq				
1	0,93	1,15	0,99	1
2	2,17	2,26	1,98	2
3	3,25	2,15	2,82	3
4	3,25	4,41	3,6	4
5	3,25	4,78	4,33	5
6	4,33	4,3	5,33	6
7	0,65	4,2	5,48	7
8	4,33	3,02	6,35	8
9	6,5	6,62	7,37	9
10	6,5	7,82	7,46	10
11	4,33	5,93	8,29	11
12	2,6	6,62	8,41	12
13	3,25	10,12	5,38	13
14	4,33	10,75	6,63	14
15	2,6	5,55	8,91	15
16	6,5	9,56	9,05	16
17	6,5	7,82	8,91	17
18	6,5	9,56	6,63	18
19	4,33	10,12	7,65	19
20	3,25	9,05	11,26	20
21	6,5	11,47	11,06	21
22	4,33	10,75	7,28	22
23	4,33	8,19	7,86	23
24	2,6	9,56	9,79	24

Scalability				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	2,33	1,97	1,99	2
3	3,5	1,88	2,83	3
4	3,5	3,85	3,62	4
5	3,5	4,17	4,36	5
6	4,67	3,75	5,37	6
7	0,7	3,66	5,51	7
8	4,67	2,63	6,39	8
9	7	5,77	7,42	9
10	7	6,82	7,51	10
11	4,67	5,17	8,35	11
12	2,8	5,77	8,46	12
13	3,5	8,82	5,41	13
14	4,67	9,38	6,68	14
15	2,8	4,84	8,97	15
16	7	8,33	9,11	16
17	7	6,82	8,97	17
18	7	8,33	6,68	18
19	4,67	8,82	7,71	19
20	3,5	7,89	11,34	20
21	7	10	11,13	21
22	4,67	9,38	7,33	22
23	4,67	7,14	7,91	23
24	2,8	8,33	9,85	24

Efficiency				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	1,08	1,13	0,99	1
3	1,08	0,72	0,94	1
4	0,81	1,1	0,9	1
5	0,65	0,96	0,87	1
6	0,72	0,72	0,89	1
7	0,09	0,6	0,78	1
8	0,54	0,38	0,79	1
9	0,72	0,74	0,82	1
10	0,65	0,78	0,75	1
11	0,39	0,54	0,75	1
12	0,22	0,55	0,7	1
13	0,25	0,78	0,41	1
14	0,31	0,77	0,47	1
15	0,17	0,37	0,59	1
16	0,41	0,6	0,57	1
17	0,38	0,46	0,52	1
18	0,36	0,53	0,37	1
19	0,23	0,53	0,4	1
20	0,16	0,45	0,56	1
21	0,31	0,55	0,53	1
22	0,2	0,49	0,33	1
23	0,19	0,36	0,34	1
24	0,11	0,4	0,41	1

Stream size: 50	100	500	1000
seq	296	7524	29660
1	306	7543	29925
2	158	3918	16062
3	109	2922	10839
4	86	2151	8470
5	83	1869	7316
6	117	1675	6195
7	66	1536	5408
8	71	1274	5070
9	91	1316	4647
10	56	1254	4084
11	71	1169	3962
12	57	1098	3757
13	60	1084	3602
14	78	1097	3490
15	72	969	3457
16	51	957	3472
17	77	959	3214
18	59	909	3102
19	43	1035	3063
20	40	944	3043
21	71	939	3118
22	61	1082	2759
23	53	956	3344
24	45	951	3328

Stream size: 50	100	500	1000	ideal
seq				
1	0,97	1	0,99	1
2	1,87	1,92	1,85	2
3	2,72	2,57	2,74	3
4	3,44	3,5	3,5	4
5	3,57	4,03	4,05	5
6	2,53	4,49	4,79	6
7	4,48	4,9	5,48	7
8	4,17	5,91	5,85	8
9	3,25	5,72	6,38	9
10	5,29	6	7,26	10
11	4,17	6,44	7,49	11
12	5,19	6,85	7,89	12
13	4,93	6,94	8,23	13
14	3,79	6,86	8,5	14
15	4,11	7,76	8,58	15
16	5,8	7,86	8,54	16
17	3,84	7,85	9,23	17
18	5,02	8,28	9,56	18
19	6,88	7,27	9,68	19
20	7,4	7,97	9,75	20
21	4,17	8,01	9,51	21
22	4,85	6,95	10,75	22
23	5,58	7,87	8,87	23
24	6,58	7,91	8,91	24

Stream size: 50	100	500	1000	ideal
seq				
1				
2	1,94	1,93	1,86	2
3	2,81	2,58	2,76	3
4	3,56	3,51	3,53	4
5	3,69	4,04	4,09	5
6	2,62	4,5	4,83	6
7	4,64	4,91	5,53	7
8	4,31	5,92	5,9	8
9	3,36	5,73	6,44	9
10	5,46	6,02	7,33	10
11	4,31	6,45	7,55	11
12	5,37	6,87	7,97	12
13	5,1	6,96	8,31	13
14	3,92	6,88	8,57	14
15	4,25	7,78	8,66	15
16	6	7,88	8,62	16
17	3,97	7,87	9,31	17
18	5,19	8,3	9,65	18
19	7,12	7,29	9,77	19
20	7,65	7,99	9,83	20
21	4,31	8,03	9,6	21
22	5,02	6,97	10,85	22
23	5,77	7,89	8,95	23
24	6,8	7,93	8,99	24

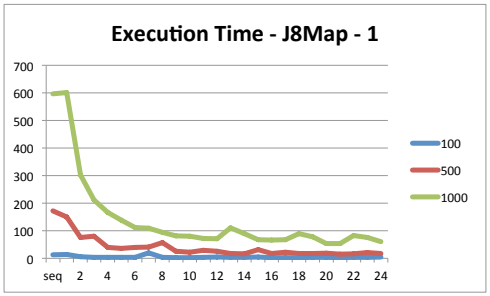
Stream size: 50	100	500	1000	ideal
seq				
1				
2	0,94	0,96	0,92	1
3	0,91	0,86	0,91	1
4	0,86	0,87	0,88	1
5	0,71	0,81	0,81	1
6	0,42	0,75	0,8	1
7	0,64	0,7	0,78	1
8	0,52	0,74	0,73	1
9	0,36	0,64	0,71	1
10	0,53	0,6	0,73	1
11	0,38	0,59	0,68	1
12	0,43	0,57	0,66	1
13	0,38	0,53	0,63	1
14	0,27	0,49	0,61	1
15	0,27	0,52	0,57	1
16	0,36	0,49	0,53	1
17	0,23	0,46	0,54	1
18	0,28	0,46	0,53	1
19	0,36	0,38	0,51	1
20	0,37	0,4	0,49	1
21	0,2	0,38	0,45	1
22	0,22	0,32	0,49	1
23	0,24	0,34	0,39	1
24	0,27	0,33	0,37	1

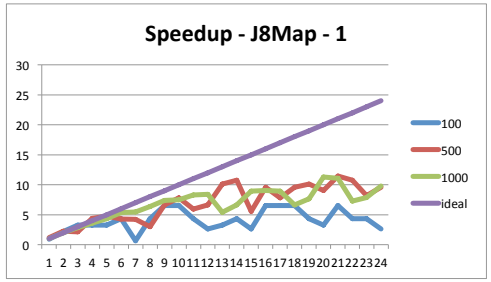
Stream size: 100	100	500	1000
seq	584	14879	59286
1	629	15045	59914
2	313	8121	31176
3	216	5862	21594
4	233	4502	17000
5	171	3767	14321
6	137	3362	12602
7	196	3048	10838
8	196	2809	10056
9	129	2830	8996
10	159	2472	8424
11	126	2239	8153
12	120	2267	7475
13	115	2171	7380
14	90	2088	6937
15	131	1932	6885
16	101	2066	6412
17	140	1872	6916
18	114	1885	6870
19	92	1983	6216
20	89	1717	7158
21	142	2096	6368
22	118	1890	5633
23	97	1976	6143
24	83	2061	5989

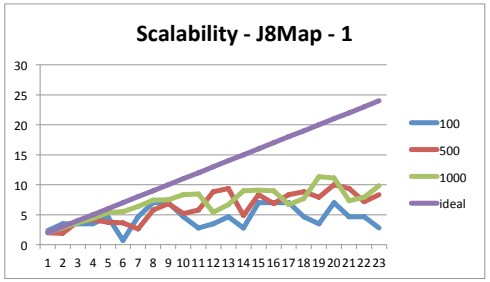
Stream size: 100	100	500	1000	ideal
seq				
1	0,93	0,99	0,99	1
2	1,87	1,83	1,9	2
3	2,7	2,54	2,75	3
4	2,51	3,3	3,49	4
5	3,42	3,95	4,14	5
6	4,26	4,43	4,7	6
7	2,98	4,88	5,47	7
8	2,98	5,3	5,9	8
9	4,53	5,26	6,59	9
10	3,67	6,02	7,04	10
11	4,63	6,65	7,27	11
12	4,87	6,56	7,93	12
13	5,08	6,85	8,03	13
14	6,49	7,13	8,55	14
15	4,46	7,7	8,61	15
16	5,78	7,2	9,25	16
17	4,17	7,95	8,57	17
18	5,12	7,89	8,63	18
19	6,35	7,5	9,54	19
20	6,56	8,67	8,28	20
21	4,11	7,1	9,31	21
22	4,95	7,87	10,52	22
23	6,02	7,53	9,65	23
24	7,04	7,22	9,9	24

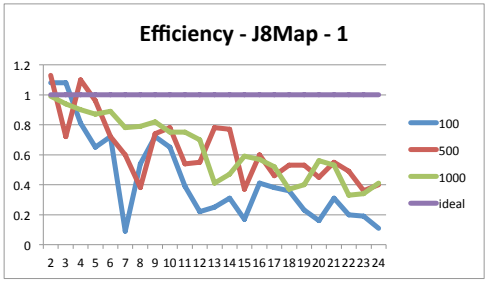
Stream size: 100	100	500	1000	ideal
seq				
1				
2	2,01	1,85	1,92	2
3	2,91	2,57	2,77	3
4	2,7	3,34	3,52	4
5	3,68	3,99	4,18	5
6	4,59	4,48	4,75	6
7	3,21	4,94	5,53	7
8	3,21	5,36	5,96	8
9	4,88	5,32	6,66	9
10	3,96	6,09	7,11	10
11	4,99	6,72	7,35	11
12	5,24	6,64	8,02	12
13	5,47	6,93	8,12	13
14	6,99	7,21	8,64	14
15	4,8	7,79	8,7	15
16	6,23	7,28	9,34	16
17	4,49	8,04	8,66	17
18	5,52	7,98	8,72	18
19	6,84	7,59	9,64	19
20	7,07	8,76	8,37	20
21	4,43	7,18	9,41	21
22	5,33	7,96	10,64	22
23	6,48	7,61	9,75	23
24	7,58	7,3	10	24

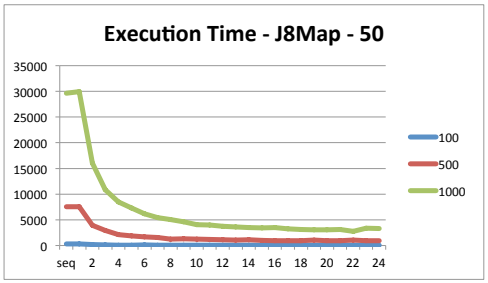
Stream size: 100	100	500	1000	ideal
seq				
1				
2	0,93	0,92	0,95	1
3	0,9	0,85	0,92	1
4	0,63	0,83	0,87	1
5	0,68	0,79	0,83	1
6	0,71	0,74	0,78	1
7	0,43	0,7	0,78	1
8	0,37	0,66	0,74	1
9	0,5	0,58	0,73	1
10	0,37	0,6	0,7	1
11	0,42	0,6	0,66	1
12	0,41	0,55	0,66	1
13	0,39	0,53	0,62	1
14	0,46	0,51	0,61	1
15	0,3	0,51	0,57	1
16	0,36	0,45	0,58	1
17	0,25	0,47	0,5	1
18	0,28	0,44	0,48	1
19	0,33	0,39	0,5	1
20	0,33	0,43	0,41	1
21	0,2	0,34	0,44	1
22	0,22	0,36	0,48	1
23	0,26	0,33	0,42	1
24	0,29	0,3	0,41	1

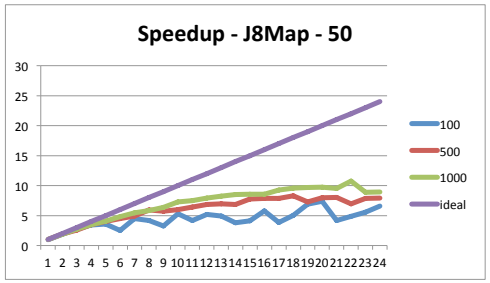


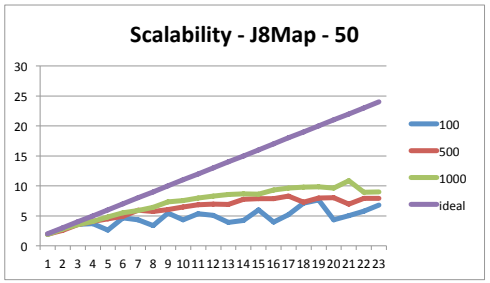


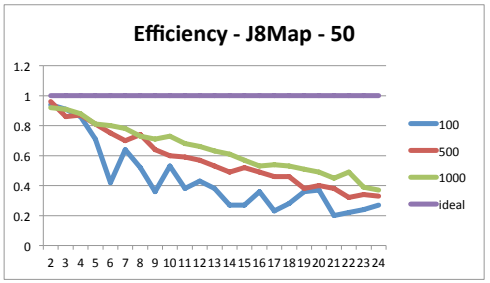


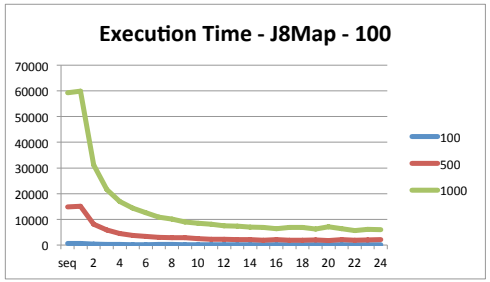


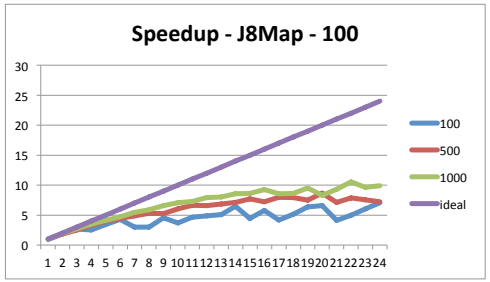


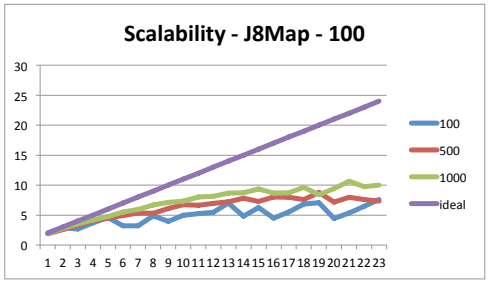


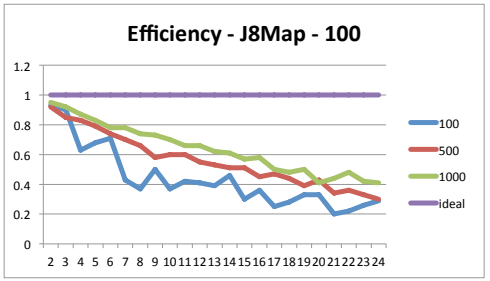












Each graph represents different test conducted. Each row contains the graph of the time spent, the speedup, the scalability and the efficiency. First row is executed on a single matrix stream, second and third on 50 and 100 matrix stream respectively. The size of the matrix is indicated in the legend of the graph

Skandium Farm

Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	15	156	586
2	9	157	599
3	6	156	580
4	8	145	573
5	8	142	584
6	7	163	579
7	7	162	582
8	12	153	739
9	9	142	580
10	6	153	578
11	8	153	574
12	12	144	575
13	8	145	581
14	6	161	575
15	6	156	574
16	7	145	576
17	6	144	603
18	9	143	685
19	8	143	575
20	6	143	609
21	6	142	586
22	6	143	591
23	6	143	582
24	6	145	580

Speedup				
Stream size: 1	100	500	1000	ideal
seq				
1	0,87	1,1	1,02	1
2	1,44	1,1	1	2
3	2,17	1,1	1,03	3
4	1,62	1,19	1,04	4
5	1,62	1,21	1,02	5
6	1,86	1,06	1,03	6
7	1,86	1,06	1,03	7
8	1,08	1,12	0,81	8
9	1,44	1,21	1,03	9
10	2,17	1,12	1,03	10
11	1,62	1,12	1,04	11
12	1,08	1,19	1,04	12
13	1,62	1,19	1,03	13
14	2,17	1,07	1,04	14
15	2,17	1,1	1,04	15
16	1,86	1,19	1,04	16
17	2,17	1,19	0,99	17
18	1,44	1,2	0,87	18
19	1,62	1,2	1,04	19
20	2,17	1,2	0,98	20
21	2,17	1,21	1,02	21
22	2,17	1,2	1,01	22
23	2,17	1,2	1,03	23
24	2,17	1,19	1,03	24

Scalability				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	1,67	0,99	0,98	2
3	2,5	1	1,01	3
4	1,88	1,08	1,02	4
5	1,88	1,1	1	5
6	2,14	0,96	1,01	6
7	2,14	0,96	1,01	7
8	1,25	1,02	0,79	8
9	1,67	1,1	1,01	9
10	2,5	1,02	1,01	10
11	1,88	1,02	1,02	11
12	1,25	1,08	1,02	12
13	1,88	1,08	1,01	13
14	2,5	0,97	1,02	14
15	2,5	1	1,02	15
16	2,14	1,08	1,02	16
17	2,5	1,08	0,97	17
18	1,67	1,09	0,86	18
19	1,88	1,09	1,02	19
20	2,5	1,09	0,96	20
21	2,5	1,1	1	21
22	2,5	1,09	0,99	22
23	2,5	1,09	1,01	23
24	2,5	1,08	1,01	24

Efficiency				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	0,72	0,55	0,5	1
3	0,72	0,37	0,34	1
4	0,41	0,3	0,26	1
5	0,33	0,24	0,2	1
6	0,31	0,18	0,17	1
7	0,27	0,15	0,15	1
8	0,14	0,14	0,1	1
9	0,16	0,13	0,11	1
10	0,22	0,11	0,1	1
11	0,15	0,1	0,09	1
12	0,09	0,1	0,09	1
13	0,12	0,09	0,08	1
14	0,15	0,08	0,07	1
15	0,14	0,07	0,07	1
16	0,12	0,07	0,06	1
17	0,13	0,07	0,06	1
18	0,08	0,07	0,05	1
19	0,09	0,06	0,05	1
20	0,11	0,06	0,05	1
21	0,1	0,06	0,05	1
22	0,1	0,05	0,05	1
23	0,09	0,05	0,04	1
24	0,09	0,05	0,04	1

Stream size: 50	100	500	1000
seq	296	7524	29660
1	278	7246	29063
2	140	3706	14905
3	96	2595	10730
4	73	2021	8131
5	57	1489	6635
6	51	1391	6235
7	45	1235	5154
8	40	1108	5145
9	34	957	4467
10	29	830	4076
11	29	931	3866
12	29	772	3843
13	24	658	3587
14	24	777	3331
15	25	672	3458
16	23	653	3318
17	20	638	3015
18	22	550	2847
19	20	553	2654
20	23	594	2669
21	17	580	2787
22	21	577	2580
23	22	569	2613
24	23	524	2349

Stream size: 50	100	500	1000	ideal
seq				
1	1,06	1,04	1,02	1
2	2,11	2,03	1,99	2
3	3,08	2,9	2,76	3
4	4,05	3,72	3,65	4
5	5,19	5,05	4,47	5
6	5,8	5,41	4,76	6
7	6,58	6,09	5,75	7
8	7,4	6,79	5,76	8
9	8,71	7,86	6,64	9
10	10,21	9,07	7,28	10
11	10,21	8,08	7,67	11
12	10,21	9,75	7,72	12
13	12,33	11,43	8,27	13
14	12,33	9,68	8,9	14
15	11,84	11,2	8,58	15
16	12,87	11,52	8,94	16
17	14,8	11,79	9,84	17
18	13,45	13,68	10,42	18
19	14,8	13,61	11,18	19
20	12,87	12,67	11,11	20
21	17,41	12,97	10,64	21
22	14,1	13,04	11,5	22
23	13,45	13,22	11,35	23
24	12,87	14,36	12,63	24

Stream size: 50	100	500	1000	ideal
seq				
1				
2	1,99	1,96	1,95	2
3	2,9	2,79	2,71	3
4	3,81	3,59	3,57	4
5	4,88	4,87	4,38	5
6	5,45	5,21	4,66	6
7	6,18	5,87	5,64	7
8	6,95	6,54	5,65	8
9	8,18	7,57	6,51	9
10	9,59	8,73	7,13	10
11	9,59	7,78	7,52	11
12	9,59	9,39	7,56	12
13	11,58	11,01	8,1	13
14	11,58	9,33	8,73	14
15	11,12	10,78	8,4	15
16	12,09	11,1	8,76	16
17	13,9	11,36	9,64	17
18	12,64	13,17	10,21	18
19	13,9	13,1	10,95	19
20	12,09	12,2	10,89	20
21	16,35	12,49	10,43	21
22	13,24	12,56	11,26	22
23	12,64	12,73	11,12	23
24	12,09	13,83	12,37	24

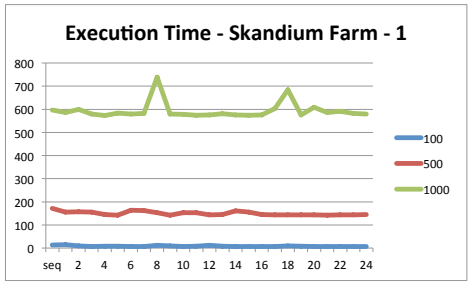
Stream size: 50	100	500	1000	ideal
seq				
1				
2	1,06	1,02	0,99	1
3	1,03	0,97	0,92	1
4	1,01	0,93	0,91	1
5	1,04	1,01	0,89	1
6	0,97	0,9	0,79	1
7	0,94	0,87	0,82	1
8	0,93	0,85	0,72	1
9	0,97	0,87	0,74	1
10	1,02	0,91	0,73	1
11	0,93	0,73	0,7	1
12	0,85	0,81	0,64	1
13	0,95	0,88	0,64	1
14	0,88	0,69	0,64	1
15	0,79	0,75	0,57	1
16	0,8	0,72	0,56	1
17	0,87	0,69	0,58	1
18	0,75	0,76	0,58	1
19	0,78	0,72	0,59	1
20	0,64	0,63	0,56	1
21	0,83	0,62	0,51	1
22	0,64	0,59	0,52	1
23	0,58	0,57	0,49	1
24	0,54	0,6	0,53	1

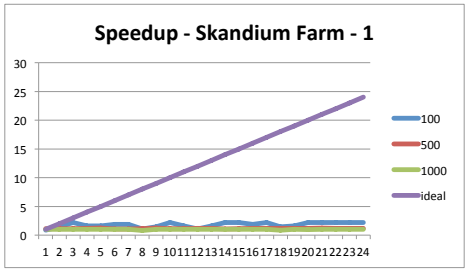
Stream size: 100	100	500	1000
seq	584	14879	59286
1	601	14615	58437
2	282	7422	30766
3	207	5084	21477
4	145	3966	16383
5	116	3151	13750
6	95	2809	12166
7	83	2422	11277
8	74	2203	9833
9	67	1963	8839
10	62	1717	8333
11	56	1726	7681
12	96	1619	7148
13	82	1434	6777
14	74	1523	6399
15	76	1385	6130
16	41	1384	5925
17	40	1284	5996
18	39	1300	5872
19	38	1160	5282
20	40	1148	5499
21	35	1166	5261
22	34	1129	4872
23	35	1104	4798
24	36	1156	4563

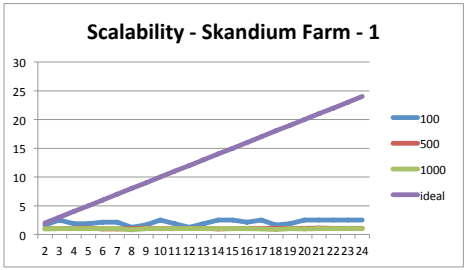
Stream size: 100	100	500	1000	ideal
seq				
1	0,97	1,02	1,01	1
2	2,07	2	1,93	2
3	2,82	2,93	2,76	3
4	4,03	3,75	3,62	4
5	5,03	4,72	4,31	5
6	6,15	5,3	4,87	6
7	7,04	6,14	5,26	7
8	7,89	6,75	6,03	8
9	8,72	7,58	6,71	9
10	9,42	8,67	7,11	10
11	10,43	8,62	7,72	11
12	6,08	9,19	8,29	12
13	7,12	10,38	8,75	13
14	7,89	9,77	9,26	14
15	7,68	10,74	9,67	15
16	14,24	10,75	10,01	16
17	14,6	11,59	9,89	17
18	14,97	11,45	10,1	18
19	15,37	12,83	11,22	19
20	14,6	12,96	10,78	20
21	16,69	12,76	11,27	21
22	17,18	13,18	12,17	22
23	16,69	13,48	12,36	23
24	16,22	12,87	12,99	24

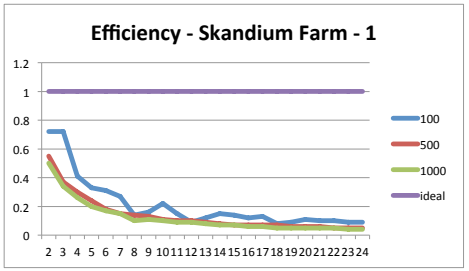
Stream size: 100	100	500	1000	ideal
seq				
1				
2	2,13	1,97	1,9	2
3	2,9	2,87	2,72	3
4	4,14	3,69	3,57	4
5	5,18	4,64	4,25	5
6	6,33	5,2	4,8	6
7	7,24	6,03	5,18	7
8	8,12	6,63	5,94	8
9	8,97	7,45	6,61	9
10	9,69	8,51	7,01	10
11	10,73	8,47	7,61	11
12	6,26	9,03	8,18	12
13	7,33	10,19	8,62	13
14	8,12	9,6	9,13	14
15	7,91	10,55	9,53	15
16	14,66	10,56	9,86	16
17	15,03	11,38	9,75	17
18	15,41	11,24	9,95	18
19	15,82	12,6	11,06	19
20	15,03	12,73	10,63	20
21	17,17	12,53	11,11	21
22	17,68	12,95	11,99	22
23	17,17	13,24	12,18	23
24	16,69	12,64	12,81	24

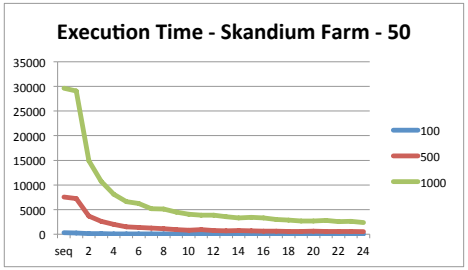
Stream size: 100	100	500	1000	ideal
seq				
1				
2	1,04	1	0,96	1
3	0,94	0,98	0,92	1
4	1,01	0,94	0,9	1
5	1,01	0,94	0,86	1
6	1,02	0,88	0,81	1
7	1,01	0,88	0,75	1
8	0,99	0,84	0,75	1
9	0,97	0,84	0,75	1
10	0,94	0,87	0,71	1
11	0,95	0,78	0,7	1
12	0,51	0,77	0,69	1
13	0,55	0,8	0,67	1
14	0,56	0,7	0,66	1
15	0,51	0,72	0,64	1
16	0,89	0,67	0,63	1
17	0,86	0,68	0,58	1
18	0,83	0,64	0,56	1
19	0,81	0,68	0,59	1
20	0,73	0,65	0,54	1
21	0,79	0,61	0,54	1
22	0,78	0,6	0,55	1
23	0,73	0,59	0,54	1
24	0,68	0,54	0,54	1

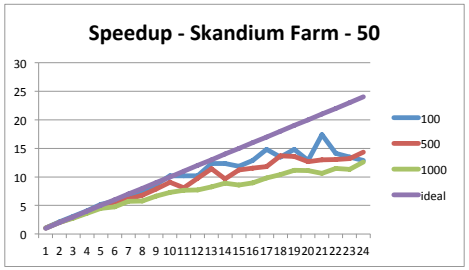


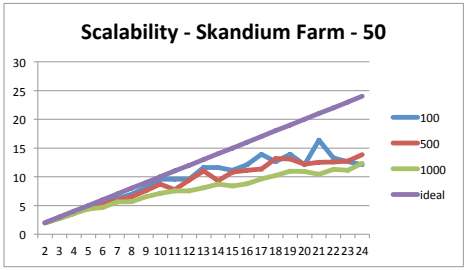


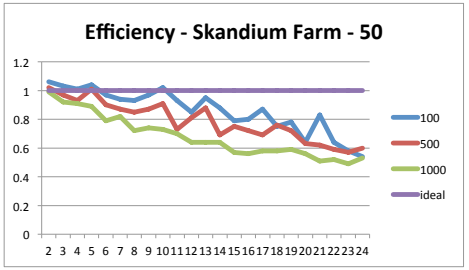


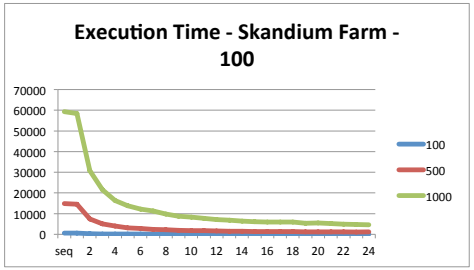


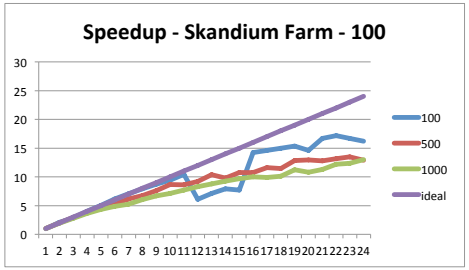


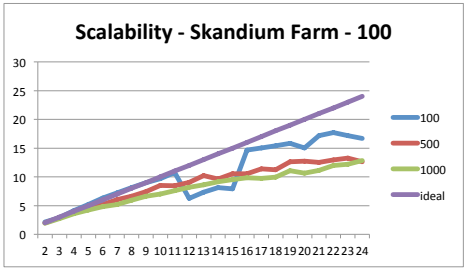


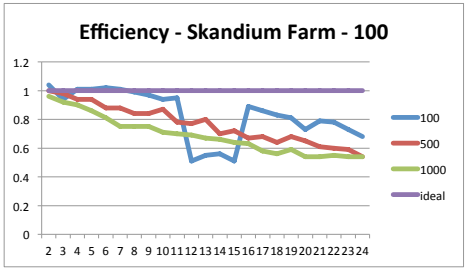












Each graph represents different test conducted. Each row contains the graph of the time spent, the speedup, the scalability and the efficiency. First row is executed on a single matrix stream, second and third on 50 and 100 matrix stream respectively. The size of the matrix is indicated in the legend of the graph

Java 8 Farm

Execution Time			
Stream size: 1	100	500	1000
seq	13	172	597
1	8	150	580
2	7	144	574
3	8	181	578
4	6	143	574
5	7	142	571
6	7	143	570
7	5	161	617
8	9	142	577
9	6	152	787
10	7	147	773
11	7	149	611
12	6	155	578
13	8	152	578
14	7	151	585
15	7	143	575
16	6	148	577
17	7	148	572
18	6	144	582
19	6	160	577
20	5	142	590
21	8	144	583
22	7	151	580
23	6	153	611
24	8	158	595

Speedup				
Stream size: 1	100	500	1000	ideal
seq				
1	1,62	1,15	1,03	1
2	1,86	1,19	1,04	2
3	1,62	0,95	1,03	3
4	2,17	1,2	1,04	4
5	1,86	1,21	1,05	5
6	1,86	1,2	1,05	6
7	2,6	1,07	0,97	7
8	1,44	1,21	1,03	8
9	2,17	1,13	0,76	9
10	1,86	1,17	0,77	10
11	1,86	1,15	0,98	11
12	2,17	1,11	1,03	12
13	1,62	1,13	1,03	13
14	1,86	1,14	1,02	14
15	1,86	1,2	1,04	15
16	2,17	1,16	1,03	16
17	1,86	1,16	1,04	17
18	2,17	1,19	1,03	18
19	2,17	1,07	1,03	19
20	2,6	1,21	1,01	20
21	1,62	1,19	1,02	21
22	1,86	1,14	1,03	22
23	2,17	1,12	0,98	23
24	1,62	1,09	1	24

Scalability				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	1,14	1,04	1,01	2
3	1	0,83	1	3
4	1,33	1,05	1,01	4
5	1,14	1,06	1,02	5
6	1,14	1,05	1,02	6
7	1,6	0,93	0,94	7
8	0,89	1,06	1,01	8
9	1,33	0,99	0,74	9
10	1,14	1,02	0,75	10
11	1,14	1,01	0,95	11
12	1,33	0,97	1	12
13	1	0,99	1	13
14	1,14	0,99	0,99	14
15	1,14	1,05	1,01	15
16	1,33	1,01	1,01	16
17	1,14	1,01	1,01	17
18	1,33	1,04	1	18
19	1,33	0,94	1,01	19
20	1,6	1,06	0,98	20
21	1	1,04	0,99	21
22	1,14	0,99	1	22
23	1,33	0,98	0,95	23
24	1	0,95	0,97	24

Efficiency				
Stream size: 1	100	500	1000	ideal
seq				
1				
2	0,93	0,6	0,52	1
3	0,54	0,32	0,34	1
4	0,54	0,3	0,26	1
5	0,37	0,24	0,21	1
6	0,31	0,2	0,17	1
7	0,37	0,15	0,14	1
8	0,18	0,15	0,13	1
9	0,24	0,13	0,08	1
10	0,19	0,12	0,08	1
11	0,17	0,1	0,09	1
12	0,18	0,09	0,09	1
13	0,12	0,09	0,08	1
14	0,13	0,08	0,07	1
15	0,12	0,08	0,07	1
16	0,14	0,07	0,06	1
17	0,11	0,07	0,06	1
18	0,12	0,07	0,06	1
19	0,11	0,06	0,05	1
20	0,13	0,06	0,05	1
21	0,08	0,06	0,05	1
22	0,08	0,05	0,05	1
23	0,09	0,05	0,04	1
24	0,07	0,05	0,04	1

Stream size: 50	100	500	1000
seq	296	7524	29660
1	307	7298	31164
2	139	3906	15205
3	95	2498	10154
4	73	1911	8028
5	56	1582	6505
6	51	1412	5612
7	45	1227	5239
8	39	1222	4556
9	34	1013	4049
10	29	832	3877
11	28	806	3900
12	28	848	3655
13	24	798	3421
14	25	680	3117
15	31	688	3066
16	23	626	2972
17	22	562	3162
18	43	616	2906
19	23	679	3068
20	45	492	2815
21	21	560	2647
22	19	577	2525
23	20	516	2994
24	23	588	2711

Stream size: 50	100	500	1000	ideal
seq				
1	0,96	1,03	0,95	1
2	2,13	1,93	1,95	2
3	3,12	3,01	2,92	3
4	4,05	3,94	3,69	4
5	5,29	4,76	4,56	5
6	5,8	5,33	5,29	6
7	6,58	6,13	5,66	7
8	7,59	6,16	6,51	8
9	8,71	7,43	7,33	9
10	10,21	9,04	7,65	10
11	10,57	9,33	7,61	11
12	10,57	8,87	8,11	12
13	12,33	9,43	8,67	13
14	11,84	11,06	9,52	14
15	9,55	10,94	9,67	15
16	12,87	12,02	9,98	16
17	13,45	13,39	9,38	17
18	6,88	12,21	10,21	18
19	12,87	11,08	9,67	19
20	6,58	15,29	10,54	20
21	14,1	13,44	11,21	21
22	15,58	13,04	11,75	22
23	14,8	14,58	9,91	23
24	12,87	12,8	10,94	24

Stream size: 50	100	500	1000	ideal
seq				
1				
2	2,21	1,87	2,05	2
3	3,23	2,92	3,07	3
4	4,21	3,82	3,88	4
5	5,48	4,61	4,79	5
6	6,02	5,17	5,55	6
7	6,82	5,95	5,95	7
8	7,87	5,97	6,84	8
9	9,03	7,2	7,7	9
10	10,59	8,77	8,04	10
11	10,96	9,05	7,99	11
12	10,96	8,61	8,53	12
13	12,79	9,15	9,11	13
14	12,28	10,73	10	14
15	9,9	10,61	10,16	15
16	13,35	11,66	10,49	16
17	13,95	12,99	9,86	17
18	7,14	11,85	10,72	18
19	13,35	10,75	10,16	19
20	6,82	14,83	11,07	20
21	14,62	13,03	11,77	21
22	16,16	12,65	12,34	22
23	15,35	14,14	10,41	23
24	13,35	12,41	11,5	24

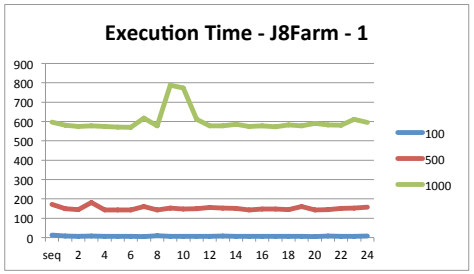
Stream size: 50	100	500	1000	ideal
seq				
1				
2	1,06	0,96	0,98	1
3	1,04	1	0,97	1
4	1,01	0,98	0,92	1
5	1,06	0,95	0,91	1
6	0,97	0,89	0,88	1
7	0,94	0,88	0,81	1
8	0,95	0,77	0,81	1
9	0,97	0,83	0,81	1
10	1,02	0,9	0,77	1
11	0,96	0,85	0,69	1
12	0,88	0,74	0,68	1
13	0,95	0,73	0,67	1
14	0,85	0,79	0,68	1
15	0,64	0,73	0,64	1
16	0,8	0,75	0,62	1
17	0,79	0,79	0,55	1
18	0,38	0,68	0,57	1
19	0,68	0,58	0,51	1
20	0,33	0,76	0,53	1
21	0,67	0,64	0,53	1
22	0,71	0,59	0,53	1
23	0,64	0,63	0,43	1
24	0,54	0,53	0,46	1

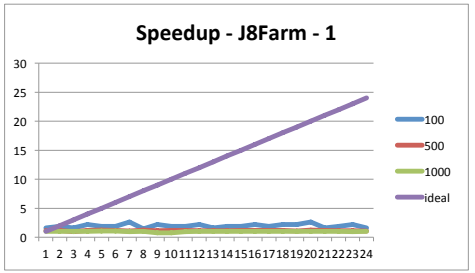
Stream size: 100	100	500	1000
seq	584	14879	59286
1	604	14778	58614
2	279	7428	30477
3	191	5082	20791
4	172	3861	16244
5	114	3120	13468
6	96	2723	11780
7	84	2382	10914
8	73	2109	9253
9	67	1938	8666
10	57	1790	7877
11	57	1628	7535
12	53	1503	7206
13	46	1415	6856
14	46	1472	6258
15	41	1335	6060
16	40	1354	6055
17	37	1227	6016
18	36	1152	5905
19	36	1126	5100
20	35	1132	5193
21	34	1170	4776
22	33	1115	4498
23	33	1083	4960
24	30	1064	4516

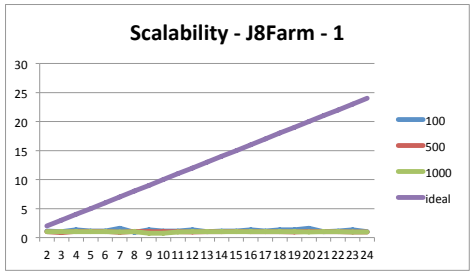
Stream size: 100	100	500	1000	ideal
seq				
1	0,97	1,01	1,01	1
2	2,09	2	1,95	2
3	3,06	2,93	2,85	3
4	3,4	3,85	3,65	4
5	5,12	4,77	4,4	5
6	6,08	5,46	5,03	6
7	6,95	6,25	5,43	7
8	8	7,06	6,41	8
9	8,72	7,68	6,84	9
10	10,25	8,31	7,53	10
11	10,25	9,14	7,87	11
12	11,02	9,9	8,23	12
13	12,7	10,52	8,65	13
14	12,7	10,11	9,47	14
15	14,24	11,15	9,78	15
16	14,6	10,99	9,79	16
17	15,78	12,13	9,85	17
18	16,22	12,92	10,04	18
19	16,22	13,21	11,62	19
20	16,69	13,14	11,42	20
21	17,18	12,72	12,41	21
22	17,7	13,34	13,18	22
23	17,7	13,74	11,95	23
24	19,47	13,98	13,13	24

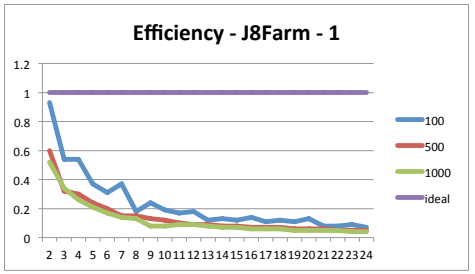
Stream size: 100	100	500	1000	ideal
seq				
1				
2	2,16	1,99	1,92	2
3	3,16	2,91	2,82	3
4	3,51	3,83	3,61	4
5	5,3	4,74	4,35	5
6	6,29	5,43	4,98	6
7	7,19	6,2	5,37	7
8	8,27	7,01	6,33	8
9	9,01	7,63	6,76	9
10	10,6	8,26	7,44	10
11	10,6	9,08	7,78	11
12	11,4	9,83	8,13	12
13	13,13	10,44	8,55	13
14	13,13	10,04	9,37	14
15	14,73	11,07	9,67	15
16	15,1	10,91	9,68	16
17	16,32	12,04	9,74	17
18	16,78	12,83	9,93	18
19	16,78	13,12	11,49	19
20	17,26	13,05	11,29	20
21	17,76	12,63	12,27	21
22	18,3	13,25	13,03	22
23	18,3	13,65	11,82	23
24	20,13	13,89	12,98	24

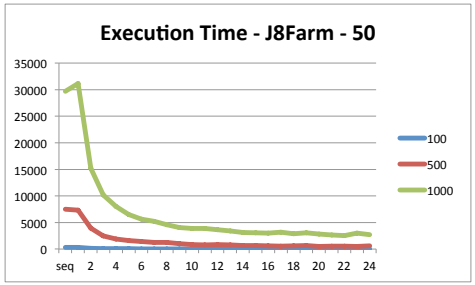
Stream size: 100	100	500	1000	ideal
seq				
1				
2	1,05	1	0,97	1
3	1,02	0,98	0,95	1
4	0,85	0,96	0,91	1
5	1,02	0,95	0,88	1
6	1,01	0,91	0,84	1
7	0,99	0,89	0,78	1
8	1	0,88	0,8	1
9	0,97	0,85	0,76	1
10	1,02	0,83	0,75	1
11	0,93	0,83	0,72	1
12	0,92	0,82	0,69	1
13	0,98	0,81	0,67	1
14	0,91	0,72	0,68	1
15	0,95	0,74	0,65	1
16	0,91	0,69	0,61	1
17	0,93	0,71	0,58	1
18	0,9	0,72	0,56	1
19	0,85	0,7	0,61	1
20	0,83	0,66	0,57	1
21	0,82	0,61	0,59	1
22	0,8	0,61	0,6	1
23	0,77	0,6	0,52	1
24	0,81	0,58	0,55	1

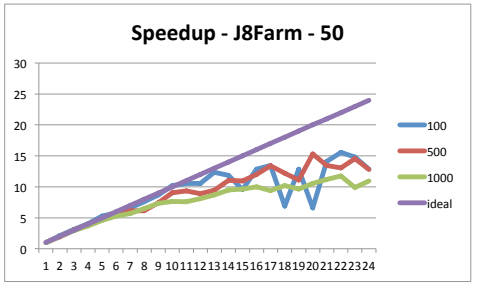


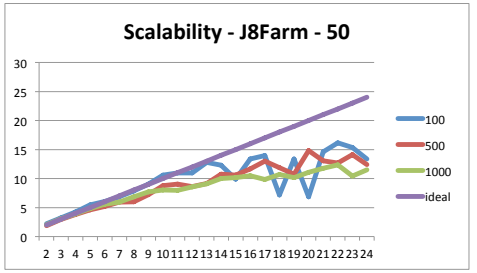


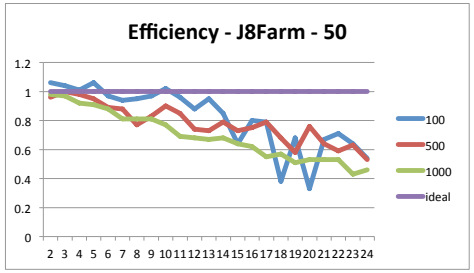


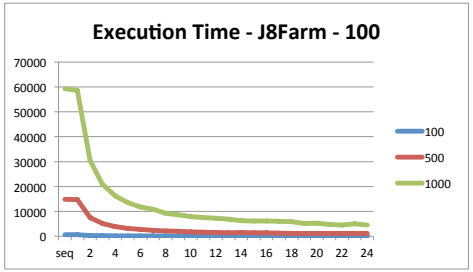


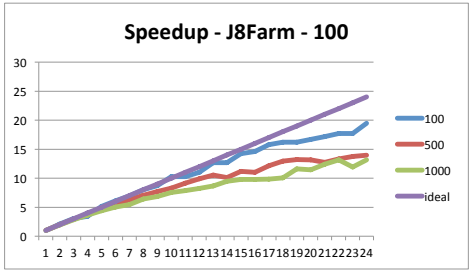


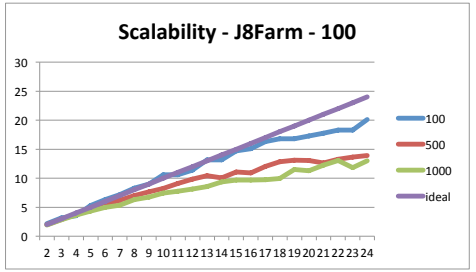


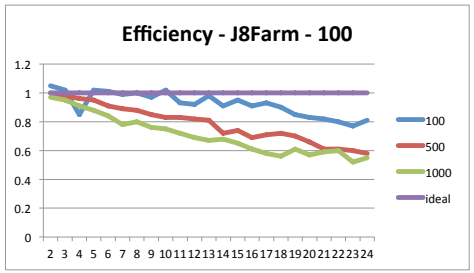












Each graph represents different test conducted. Each row contains the graph of the time spent, the speedup, the scalability and the efficiency. First row is executed on a single matrix stream, second and third on 50 and 100 matrix stream respectively. The size of the matrix is indicated in the legend of the graph

6 References

1. M. Danelutto, DISTRIBUTED SYSTEMS: PARADIGMS AND MODELS, September 5, 2014
2. M. Aldinucci, C. Spampinato, M. Drocco, M. Torquati, and S. Palazzo, “A Parallel Edge Preserving Algorithm for Salt and Pepper Image Denoising,” in Proc of 2nd Intl. Conference on Image Processing Theory Tools and Applications (IPTA), Istambul, Turkey, 2012, pp. 97-102.
3. Skandium Documentation - http://backus.di.unipi.it/~marcod/SkandiumClone/skandium.niclabs.cl/documentation/index.html
4. Richard Warburton, Java 8 Lambdas, April 7, 2014, O'Reilly Media
5. Raoul-Gabriel Urma, Mario Fusco, and Alan Mycroft, Java 8 in Action, August 2014, Manning Publications
6. Steve Holzner, Ant: The Definitive Guide, 2nd Edition, April 23, 2005, O'Reilly Media
7. Apache POI - https://poi.apache.org/
8. Project Lombok - https://projectlombok.org/features/
9. CLI - Apache Commons - https://commons.apache.org/proper/commons-cli/
10. Apache Log4j 2 - http://logging.apache.org/log4j/2.x/