2D Convolution documentation

Requirement

Consider an image represented by a matrix of pixels, of size M x N. It is required to transform it by applying a filter with a window defined by the set of indices

Ind[m, n] = {(k, l) | -m/2 <= k <= m/2, -n/2 <= l <= m/2}

and by the w_{kl} coefficients. (m < M, n < N; *m* and *n* are odd)

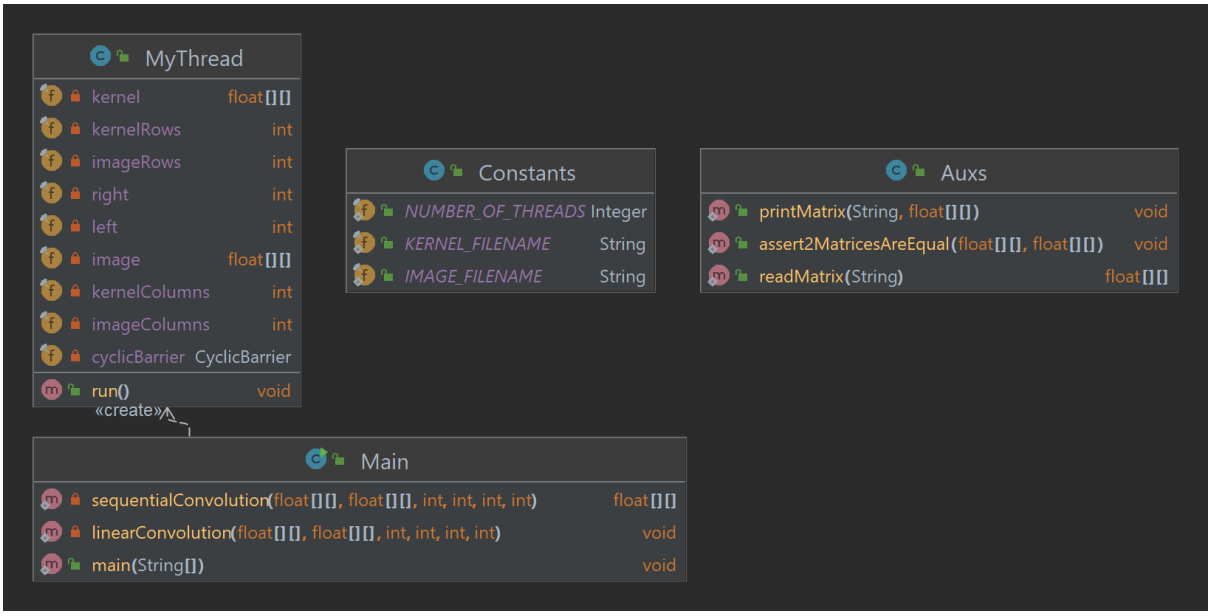
Transform a pixel:

V[i, j] = {+(k, l): -m/2 <= k <= m/2, -n/2 <= l <= n/2: w[k, l] * F[i - k, j - l]}

For borders, it is considered that an element is equal to the element from the neighboring cell in the matrix.

Target: a balanced and efficient division of the calculation on threads

Project structure



Testing results

Java:

Matrix type	**Number of threads**	**Execution time first lab (ms) (auxiliary matrix)**	**Execution time second lab (ms) (no auxiliary matrix)**
N=M=10 n=m=3	sequential	0.14187	0.14679
	4	1.98067	6.88835
N=M=1000 n=m=5	sequential	77.83433	89.91141
	1	82.83647	89.66883

	2	59.43084	60.63105
	4	51.36214	48.07502
	8	56.61964	56.66569
	16	67.76866	53.06092
N=10 M=10000 n=m=5	sequential	31.23934	32.20779
	1	25.77283	21.27519
	2	28.64151	26.20048
	4	34.98695	27.2065
	8	68,06134	48.57663
	16	117.46914	92.81338
N=10000 M=10 n=m=5	sequential	24.34933	25.88902
	1	27.19621	22.09418
	2	32.03783	27.53515
	4	35.50041	32.05028
	8	70.70573	69.78682
	16	121.8269	96.44996

C++:

Matrix type	**Allocation time**	**Number of threads**	**Execution time first lab (ms)**	**Execution time second lab (ms)**
N=M=10 n=m=3	static	4	3.29432	
	dynamic	4	2.41835	2.52974
N=M=1000 n=m=5	static	1	66.86686	
		2	34.66549	
		4	19.279	
		8	15.29438	
		16	13.89691	
	dynamic	1	59.00985	66.53036
		2	31.73885	35.18924

		4	17.53187	21.22637
		8	13.40658	14.86235
		16	12.86264	13.93203
N=10 M=10000 n=m=5	static	1	8.57784	
		2	5.66739	
		4	4.27912	
		8	4.13614	
		16	5.17771	
	dynamic	1	7.95638	8.55692
		2	5.19896	5.69817
		4	4.15227	4.51407
		8	4.15838	4.60203
		16	5.69901	5.56275
N=10000 M=10 n=m=5	static	1	24.73986	
		2	15.45019	
		4	9.87226	
		8	7.96582	
		16	7.694	
	dynamic	1	9.73673	8.64292
		2	5.81037	5.51
		4	4.22994	4.48356
		8	4.24219	4.69594
		16	5.21749	5.49456

Analysis

- In Java, the sequential variant is usually faster, except for case 2 (array 1000x1000 = 1000000 elements). The main reason is that the need for processing power is higher in that case and thus the thread creation time becomes less significant. If a thread has less to process, the time cost to create it becomes greater than the time cost to perform the calculations.
- In Java, in most cases the execution time increases proportionally to the number of threads created, the reason being the same as mentioned in the previous point.

- In C++, in most cases the execution time decreases proportionally to the number of threads created because, unlike Java, threads in C++ are native threads and cost less time to create them.
- In C++, in most cases the variant that allocates memory dynamically is faster.
- Overall, C++ is much faster than Java.