Denis R[redacted]

[redacted]

4/29/2021

[redacted]

See the attached images for a model of the network. One of them has a color-coded manual run of the network for your convenience, the other is clean.

Why should all my channels require finite capacity?

Answer: All my actors fire 1 and consume 1 token per channel per compute-execution, with few exceptions. The are the zero actor that throws out any tokens it receives, cdr that throws out its first token, input which does not consume tokens but parses a file instead; output which does not fire tokens but writes their value to a file and the scheduler which is disconnected from the network and only runs other actors. Under these circumstances one can see that every channel can have the same bandwidth too.

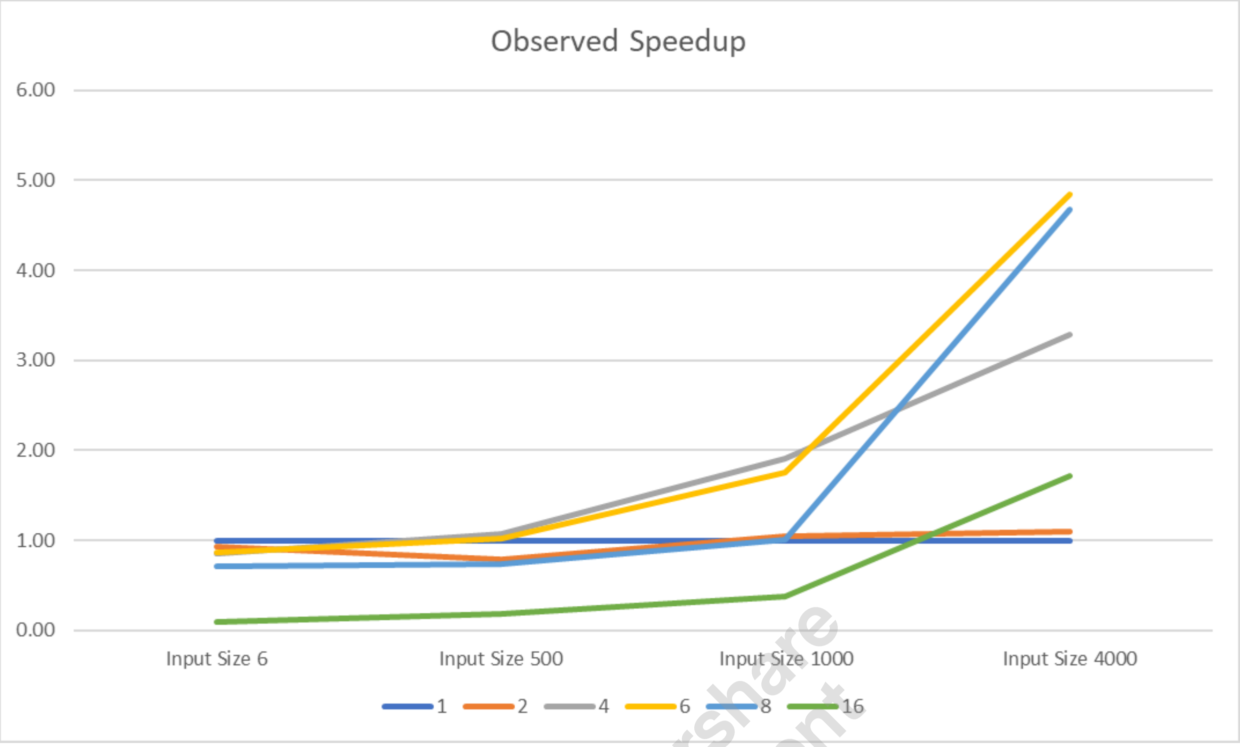
To prove this functionality every channel has a 1000 element cap. Data3 is a file with 4000 numbers in it and easily could go over the limit but not only this never happens, but the network does not clog either and keeps running. This is because actors do not run if their firing channels do not have the space to do so, thus it is guaranteed than when firing there is space in the channels.

Proof of this is at line 414, a print statement is programmed to execute if channel capacity is >1000 at firing time. That line never executes.

Speedup:

The parallelization model chosen partitions the network in areas when each thread operates exclusively. This wastes threads in later parts of the network at first but eliminates contention for intra area channels. Only inter thread area channels could have some contention. However, these channels are ConcurentLinkedQueues which by java design have not contention when there is 1 producer and 1 consumer.

See next page for speedup.



As you can observe speedup is consistently achieved for inputs of 1000 numbers and higher for 4 and 6 threads. At 4000 numbers 8 threads speedup gain is important too and 16 threads speedup gain is visible too. This table was computed with the tables in the Annex.

Finally, clearly the see that the network not only is works with a stream of data but can also process multiple consecutive uses where the network is started and terminated each time in 1 java program run: hence making it fully reusable.

Annex

Table I:

	Input Size 6	Warmup 1	Warmup 2	Warmup 3	Run 1	Run 2	Run 3	Avg T (ms)	Real Speedup
Threads									
1		20	20	20	19	27	20	22.00	1.00
2		24	22	24	23	27	21	23.67	0.93
4		24	24	21	24	23	30	25.67	0.86
6		26	29	25	24	25	27	25.33	0.87
8		43	31	96	28	30	35	31.00	0.71
16		280	215	414	242	197	242	227.00	0.10
24		2026	1783	1872	2208	751	1875	1611.33	0.01
32		4455	3784	4013	3832	3631	4096	3853.00	0.01

	Input Size 500	Warmup 1	Warmup 2	Warmup 3	Run 1	Run 2	Run 3	Avg T (ms)	Real Speedup
Threads									
1		64	60	63	65	68	63	65.33	1.00
2		69	77	65	82	84	83	83.00	0.79
4		60	62	62	65	67	51	61.00	1.07
6		65	82	68	63	72	57	64.00	1.02
8		81	100	228	96	75	94	88.33	0.74
16		504	301	405	407	431	271	369.67	0.18

	Input Size 1000	Warmup 1	Warmup 2	Warmup 3	Run 1	Run 2	Run 3	Avg T (ms)	Real Speedup
Threads									
1		151	156	153	148	154	155	152.33	1.00
2		157	137	164	154	136	145	145.00	1.05
4		60	85	93	81	81	78	80.00	1.90
6		54	72	58	95	61	105	87.00	1.75
8		108	145	106	158	217	80	151.67	1.00
16		440	401	252	396	463	356	405.00	0.38

	Input Size 4000	Warmup 1	Warmup 2	Warmup 3	Run 1	Run 2	Run 3	Avg T (ms)	Real Speedup
Threads									
1		2131	2119	2142	2169	2140	2106	2138.33	1.00
2		2003	1988	1993	1974	1880	2005	1953.00	1.09
4		585	640	587	645	589	721	651.67	3.28
6		398	498	458	428	432	463	441.00	4.85
8		488	489	434	401	455	515	457.00	4.68
16		1484	920	1288	1339	1373	1039	1250.33	1.71

Notice that the tested network has 32 actors and thus the maximum number of threads it can support is 32. See that for 32 and 24 threads computation is extremely slow for just 6 numbers to process.