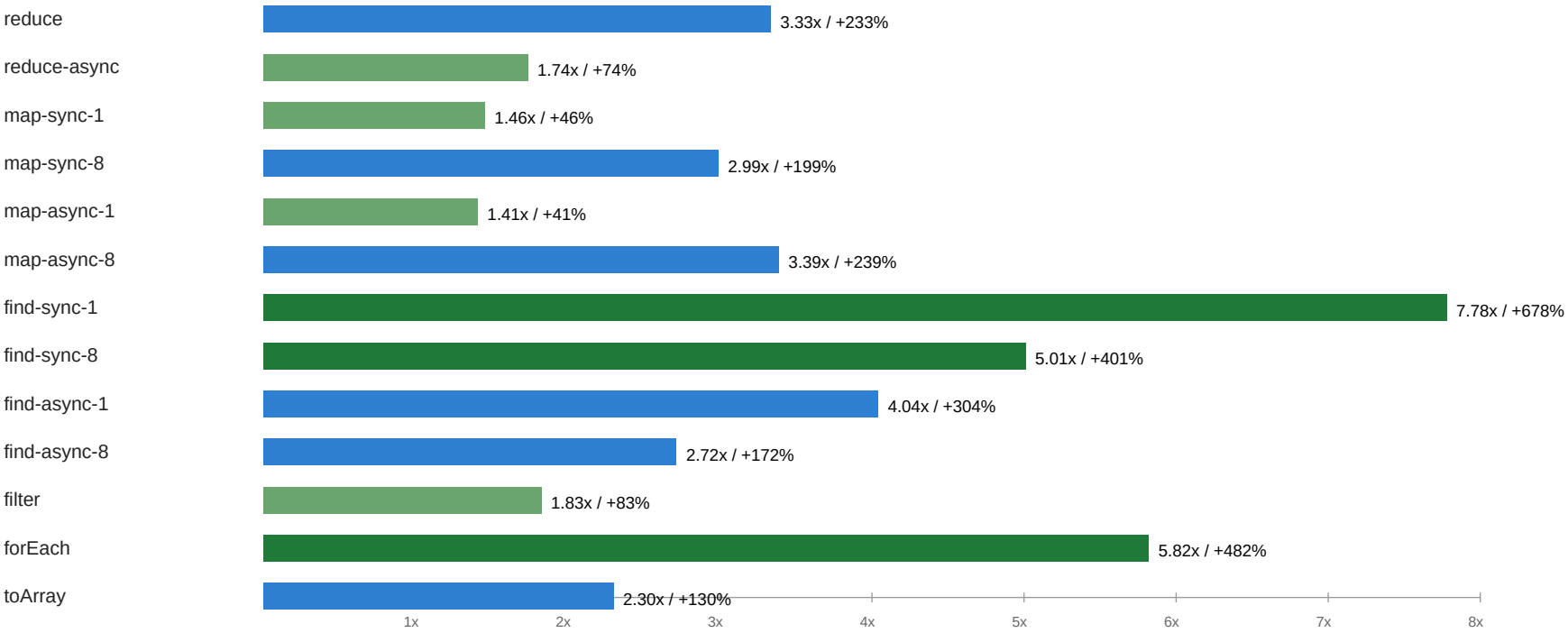
Node.js stream operator throughput benchmark

Branch implementation compared with the main worktree. Results are medians from 10 runs per configuration.

2.95x overall geomean speedup	**1.40x** minimum speedup map-async-1 / iterator	**9.46x** maximum speedup find-sync-1 / readable

geomean speedup by operation



Note: composed benchmarks such as map(...).reduce(...) and filter(...).reduce(...) measure the end-to-end operator pipeline using the implementation in each binary, including the reduce sink.

Benchmark setup and aggregate results

The benchmark file lives on the branch and was executed with both binaries. old is the main worktree binary; new is the branch binary.

branch	chore/faster-map-2	**branch commit**	64a5a60bf6
main worktree	../node.git.main	**main commit**	9ad097b82a
branch node	v27.0.0-pre	**main node**	v27.0.0-pre
runs/config	10	**n**	500,000 chunks
sources	readable, iterator, array	**operations**	13
generated	2026-08-16 19:00 PDT	**platform**	Linux 7.1.3+deb13-amd64 x86_64 unknown
raw CSV	output/pdf/stream-operator-throughput-raw.csv	**benchmark**	streams/operator-throughput.js

Command: ./out/Release/node benchmark/compare.js --old ../node.git.main/out/Release/node --new ./out/Release/node --runs 10 --filter operator-throughput --set n=500000 --no-progress streams

Aggregate speedup by operation

Geomean is computed across the readable, iterator, and array sources.

Operation	geomean speedup	change	best source	best speedup
reduce	3.33x	+233.5%	readable	4.11x
reduce-async	1.74x	+73.7%	readable	1.91x
map-sync-1	1.46x	+45.6%	array	1.48x
map-sync-8	2.99x	+199.1%	readable	3.02x
map-async-1	1.41x	+40.9%	array	1.42x
map-async-8	3.39x	+238.9%	array	3.44x
find-sync-1	7.78x	+678.0%	readable	9.46x
find-sync-8	5.01x	+401.1%	readable	6.10x
find-async-1	4.04x	+304.1%	readable	4.56x
find-async-8	2.72x	+171.5%	readable	3.05x
filter	1.83x	+82.7%	array	1.88x
forEach	5.82x	+482.1%	readable	6.65x
toArray	2.30x	+130.1%	readable	2.68x

Detailed median throughput by source

Each table compares the main worktree binary with the branch binary. Rates are median millions of chunks per second.

readable source

Median throughput in millions of chunks per second.

Operation	main M/s	branch M/s	speedup	change
reduce	5.286	21.730	4.11x	+311.1%
reduce-async	4.866	9.305	1.91x	+91.2%
map-sync-1	0.678	0.985	1.45x	+45.3%
map-sync-8	0.716	2.165	3.02x	+202.5%
map-async-1	0.686	0.967	1.41x	+41.0%
map-async-8	0.715	2.428	3.40x	+239.6%
find-sync-1	2.005	18.965	9.46x	+846.0%
find-sync-8	3.115	18.998	6.10x	+509.8%
find-async-1	1.965	8.959	4.56x	+355.9%
find-async-8	2.979	9.079	3.05x	+204.8%
filter	0.934	1.681	1.80x	+80.0%
forEach	1.917	12.749	6.65x	+565.1%
toArray	7.297	19.542	2.68x	+167.8%

iterator source

Median throughput in millions of chunks per second.

Operation	main M/s	branch M/s	speedup	change
reduce	4.862	15.405	3.17x	+216.9%
reduce-async	4.678	7.904	1.69x	+69.0%
map-sync-1	0.681	0.980	1.44x	+43.9%
map-sync-8	0.714	2.099	2.94x	+194.1%
map-async-1	0.683	0.955	1.40x	+39.8%
map-async-8	0.703	2.343	3.33x	+233.5%
find-sync-1	1.973	14.824	7.51x	+651.4%
find-sync-8	3.002	14.851	4.95x	+394.8%
find-async-1	1.926	7.543	3.92x	+291.7%
find-async-8	2.932	7.598	2.59x	+159.1%
filter	0.921	1.664	1.81x	+80.7%
forEach	1.926	10.619	5.51x	+451.4%
toArray	6.679	15.175	2.27x	+127.2%

Detailed median throughput by source - array

Array source uses Readable.from(Array.from({ length: n }, ...)).

array source

Median throughput in millions of chunks per second.

Operation	main M/s	branch M/s	speedup	change
reduce	5.235	14.901	2.85x	+184.6%
reduce-async	4.936	8.013	1.62x	+62.3%
map-sync-1	0.676	0.997	1.48x	+47.6%
map-sync-8	0.713	2.145	3.01x	+200.7%
map-async-1	0.683	0.969	1.42x	+41.8%
map-async-8	0.706	2.425	3.44x	+243.6%
find-sync-1	2.058	13.632	6.62x	+562.5%
find-sync-8	3.275	13.655	4.17x	+316.9%
find-async-1	2.045	7.556	3.70x	+269.6%
find-async-8	3.176	8.048	2.53x	+153.4%
filter	0.917	1.722	1.88x	+87.7%
forEach	2.004	10.780	5.38x	+437.9%
toArray	7.085	14.180	2.00x	+100.1%