1 Description

This is the supplementary material for the paper *Reducing Over-Synchronization in JavaScript Applications*. It includes the following contents:

- pseudocode for the REF, $stmtCanSwapDownTo$, and $latestStmtToSwapWith$, and $isIODependent$ algorithms
- graphs of the average speedups for all tests in the 20 projects we used for evaluation (Kactus is included in the paper)
- graphs of the runtimes of some more select Kactus tests (test 117 is included in the paper)
- graphs of the runtimes of some select network-dependent tests (to show that the trend of higher runtime variance in non-reordered code follows with network-dependent await-expressions)
- list of functions with environment-specific REF side effects (the corresponding list with MOD side effects is included in the paper)
- tables of average run times (in seconds) for every affected test in all the projects evaluated, with and without the reordering applied

Our artifact will contain the full source code of all stages of our tool, all data analysis code, the full data from our experiments, and an environment set up to easily rerun our experiments.

2 Extra Algorithms

This section includes pseudocode for the algorithms not included in the paper.

First we present the pseudocode for REF, the predicate for determining if a statement s references an access path a. This is a mirror of our algorithm for MOD presented in the paper. The only difference is that in the base case (case *i)*), the environmental side effects are slightly different for MOD than it is for REF. As described in the paper, with our environment-specific side effect heuristic, we track a list of functions as modifying the file system. In the REF algorithm, *any* reference to an access path rooted in a file system or network dependent package import results in the inclusion of the pseudo-variable `__FILE_SYSTEM__` or `__NETWORK__` to the REF set respectively. For global side effects, the treatment is the same for both algorithms: a statement s including to a call tracked as having global side effects has both $MOD(s) = \top$ and $REF(s) = \top$.

Input: s statement and a access path
Result: boolean to indicate if s references a

predicate $REF(s, a)$
  // (i) base case: direct reference of a
  (s has environmental side-effect a $\vee$ s references a)
  $\vee$ // recursive cases...
    // (ii) check if there's a statement nested in s (in the AST) that references a
    $\exists\ s_{in},\ nestedIn(s_{in}, s) \wedge REF(s_{in}, a)$
    // (iii) check if s references a base path of a
    $\vee \exists\ b,\ b.p == a \wedge REF(s, b)$
    // (iv) check if s references a property of a using a dynamic property expression
    $\vee$ s references to $a[p]$
    // (v) check if s contains a call to a function that references a
    $\vee \exists\ f,\ calledIn(f, s) \wedge \exists\ s_f \in f_{body}$,
      // direct reference to a in the function
      $REF(s_f, a)$
      $\vee$ // parameter alias to a is referenced in the function
      a is the i^{th} argument to f $\wedge$ $\exists\ a_{pi},\ REF(s_f, a_{pi}) \wedge a_{pi}$ is the i^{th} parameter of f
end predicate

Figure 1: Predicate for determining if an access path a is referenced by a statement s

Next, we have the pseudocode for the $stmtCanSwapDownTo$ and $latestStmtToSwapWith$, which are (respectively) the mirrors of $stmtCanSwapUpTo$ and $earliestStmtToSwapWith$ which are presented in the paper. The only differences are that in $stmtCanSwapDownTo$ the check for intermediate statements is now on $s_{down}.previousStmt$ instead of $s_{up}.nextStmt$, since we are now checking for consecutive statements from s_{down} up to s instead of from s_{up} down to s. For $latestStmtToSwapWith$, here we look for the maximum of the statements which can be swapped down to (to find the latest possible statement), instead of the minimum of the statements which can be swapped up to (to find the earliest possible statement).

Input: s and s_{down} statements
Result: boolean to indicate if s can be reordered below s_{down}

predicate $stmtCanSwapDownTo(s, s_{down})$
    // base case
        $s == s_{down}$
    $\vee$ // recursive case
        $\exists\ s_{mid},\ (\ stmtCanSwapDownTo(s, s_{mid})\ \wedge s_{down}.previousStmt == s_{mid}\ \wedge exchangeable(s, s_{down})\ )$
end predicate

Figure 2: Predicate for determining if statement s can be reordered below another statement s_{down}.

Input: s and **result** statements
Result: boolean to indicate if **result** is the latest statement below which s can be swapped

predicate $latestStmtToSwapWith(s, \textbf{result})$
    // find the latest statement s can swap below (max by source code location)
    result $== max($ all stmts s_i where $inSameBlock(s, s_i) \wedge stmtCanSwapDownTo(s, s_i))$
end predicate

Figure 3: Predicate for finding the latest statement below which s can be placed.

Finally, we include pseudocode of the algorithm for determining if a statement is dependent on some I/O-dependent package. This is described verbally in section 4.7 of the paper, and amounts to checking if any function originating from the package in question is invoked by the statement either directly or transitively. In our work, we apply this algorithm for all the packages we have identified as being I/O-dependent (for both network and file system sources of I/O).

Input: s statement and `IO_pkg` I/O-dependent package
Result: boolean to indicate if s executes a function (directly or transitively) that originates from `IO_pkg`

predicate $isIODependent(s, \texttt{IO\_PKG})$
    // base case: s directly executes a function from IO_pkg
        $\exists\ \texttt{IO\_pkg}.f,\ calledIn(\texttt{IO\_pkg}.f, s)$
    $\vee$ // recursive case
        $\exists\ f,\ calledIn(f, s) \wedge \exists\ s_f \in f_{body},\ isIODependent(s_f, \texttt{IO\_pkg})$
end predicate

Figure 4: Predicate for determining if a statement is dependent on some I/O-dependent package.

3 Average speedups for all tests

In the paper we showed a graph of the average percentage speedups for each affected test for Kactus. Here, we show the same graph for all the applications we tested. We have included Kactus again for completeness.

These graphs reflect the results presented in Table 4 of the paper (particularly, the average and maximum percentage speedups, and the proportion of tests which see significant speedups, can be compared with the values in the relevant table in the paper). Curious readers can look through these graphs to see the distribution of average speedups over all the tests for each package. Note again that in the final artifact we will make our entire data set of experiment timings available, along with an environment where readers can rerun the experiments themselves.

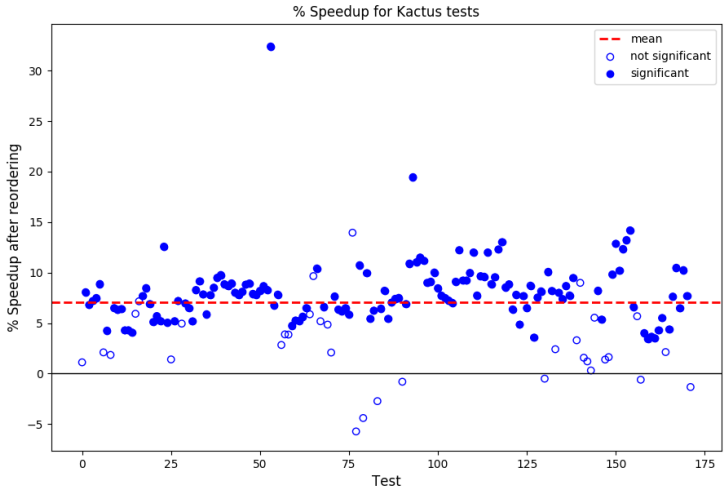

Figure 5: Average percentage speedups for all Kactus tests

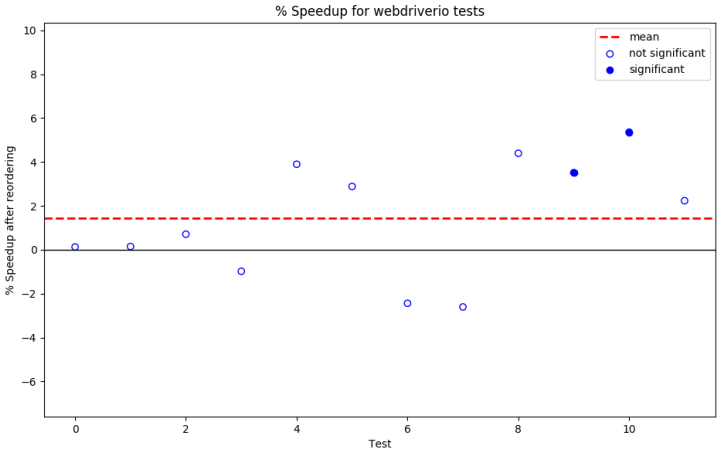

Figure 6: Average percentage speedups for all webdriverio tests

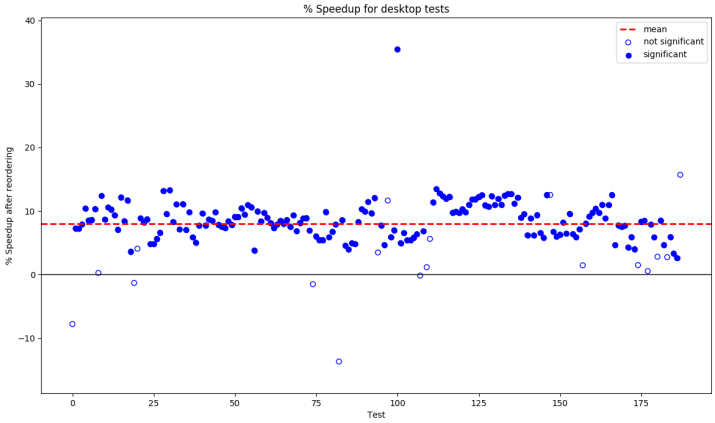

Figure 7: Average percentage speedups for all desktop tests

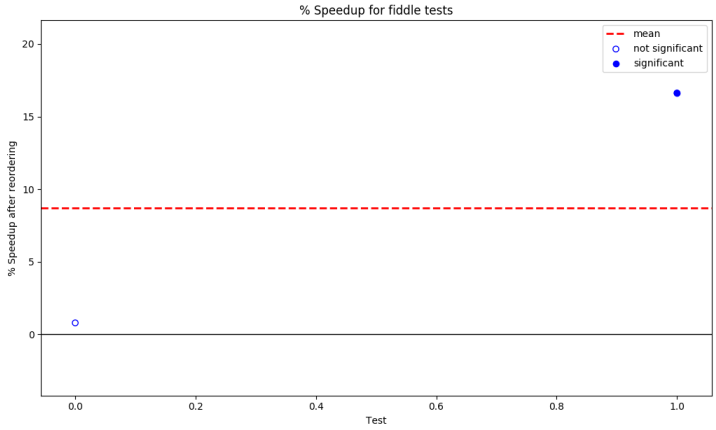

Figure 8: Average percentage speedups for all fiddle tests

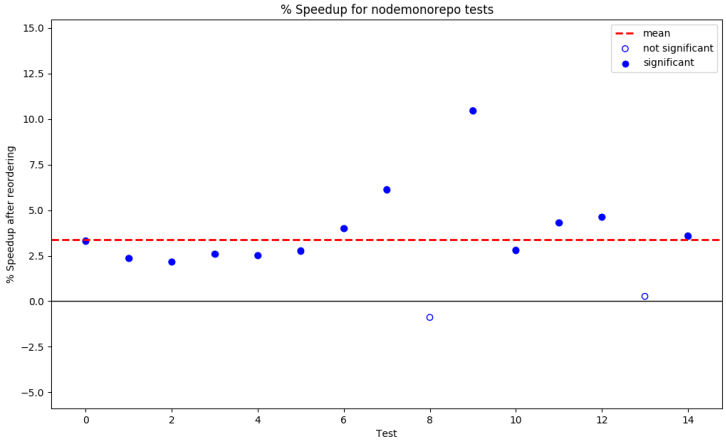

Figure 9: Average percentage speedups for all nodemonorepo tests

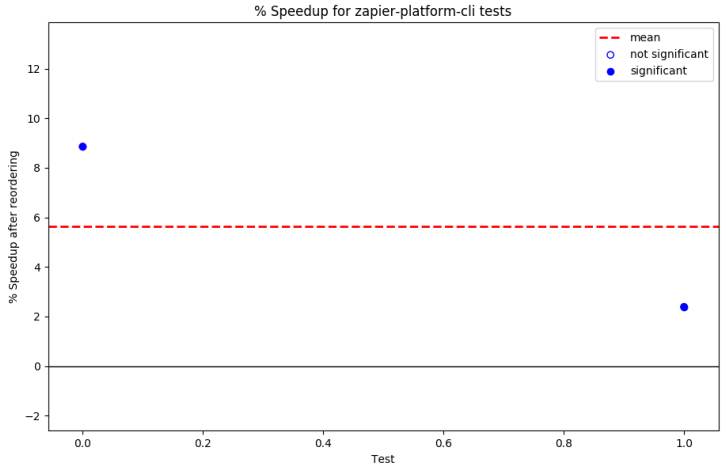

Figure 10: Average percentage speedups for all zapier-platform-cli tests

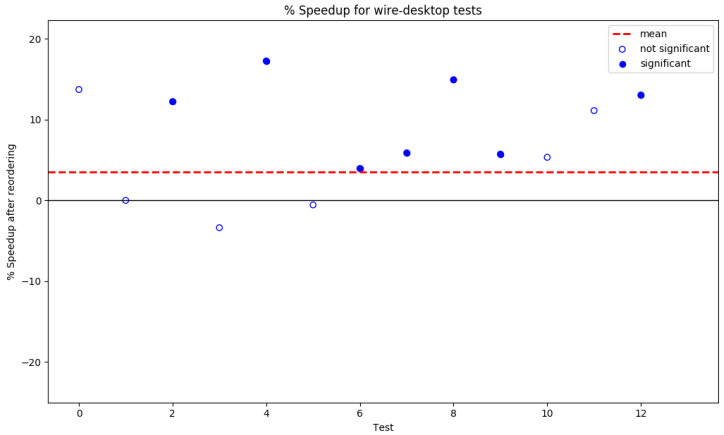

Figure 11: Average percentage speedups for all wire-desktop tests

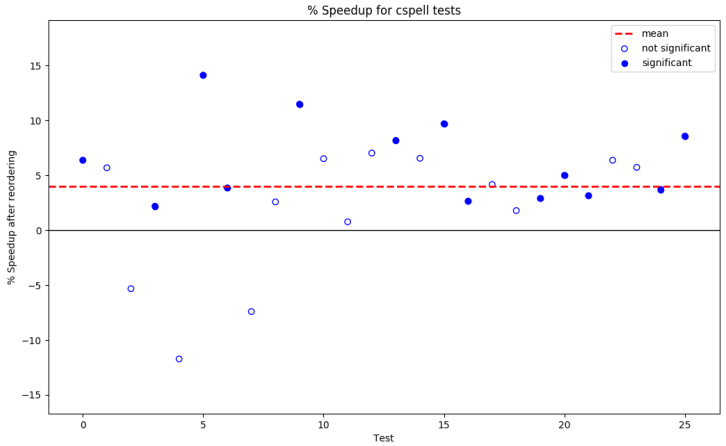

Figure 12: Average percentage speedups for all cspell tests

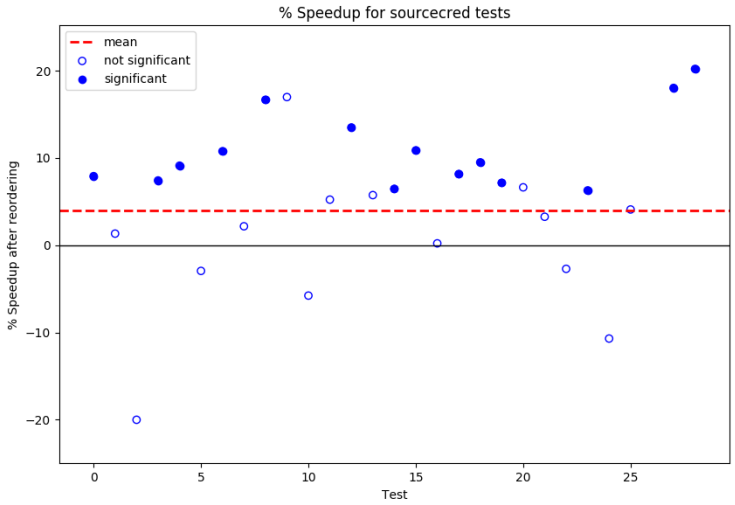

Figure 13: Average percentage speedups for all sourcecred tests

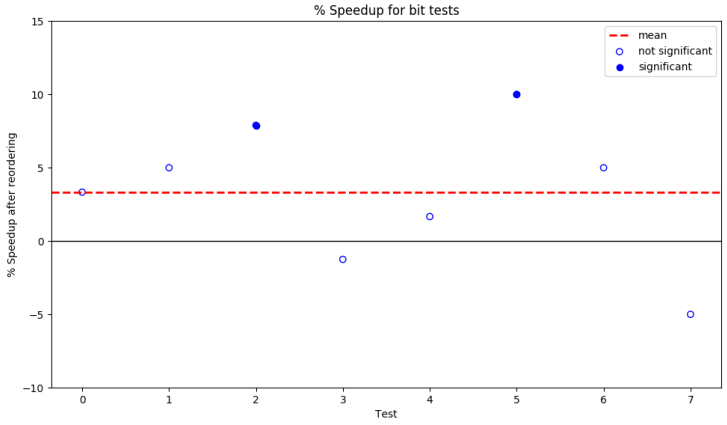

Figure 14: Average percentage speedups for all bit tests

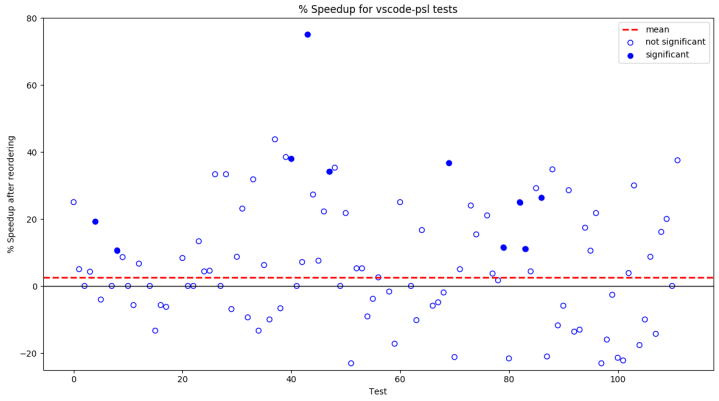

Figure 15: Average percentage speedups for all vscode-psl tests

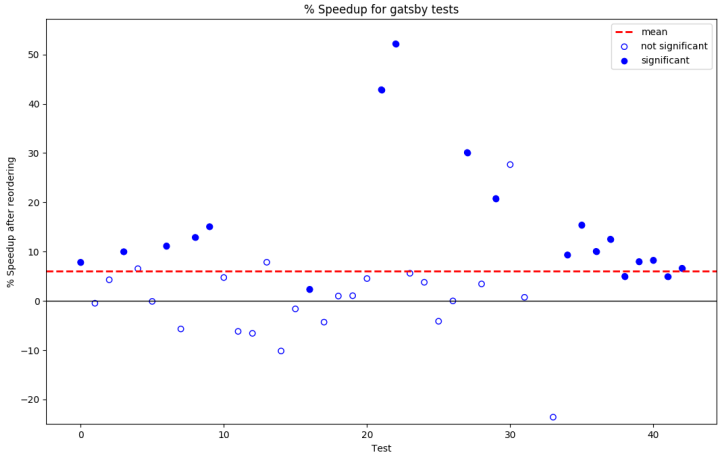

Figure 16: Average percentage speedups for all gatsby tests

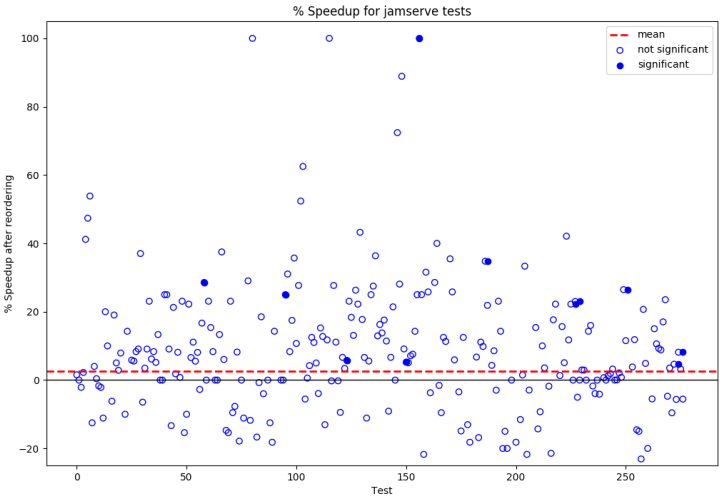

Figure 17: Average percentage speedups for all jamserve tests

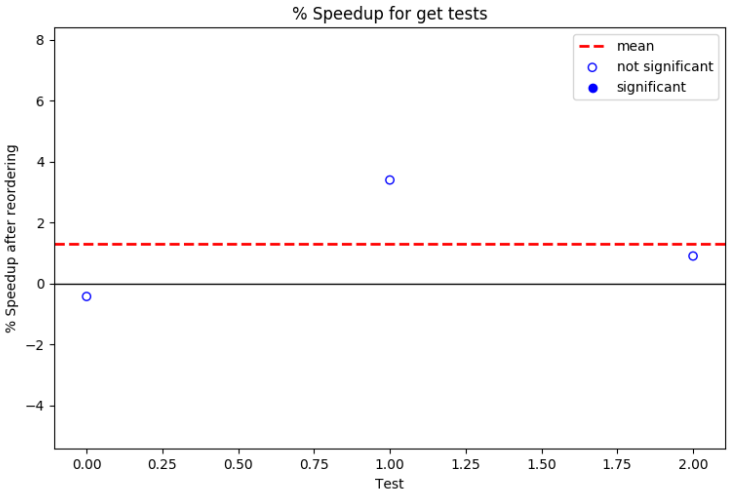

Figure 18: Average percentage speedups for all get tests

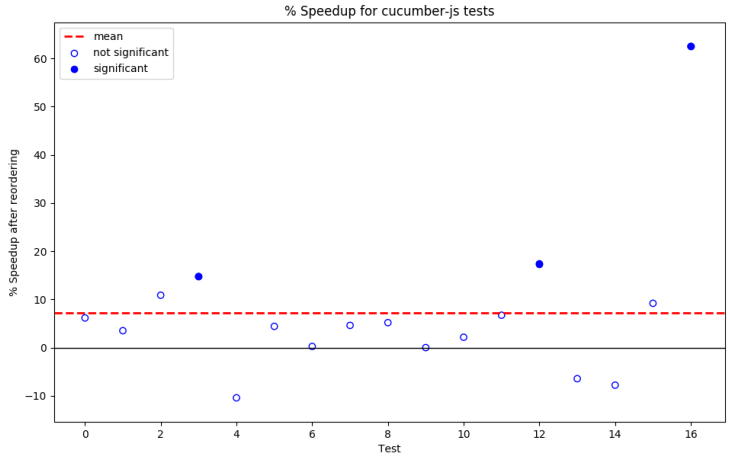

Figure 19: Average percentage speedups for all cucumber-js tests

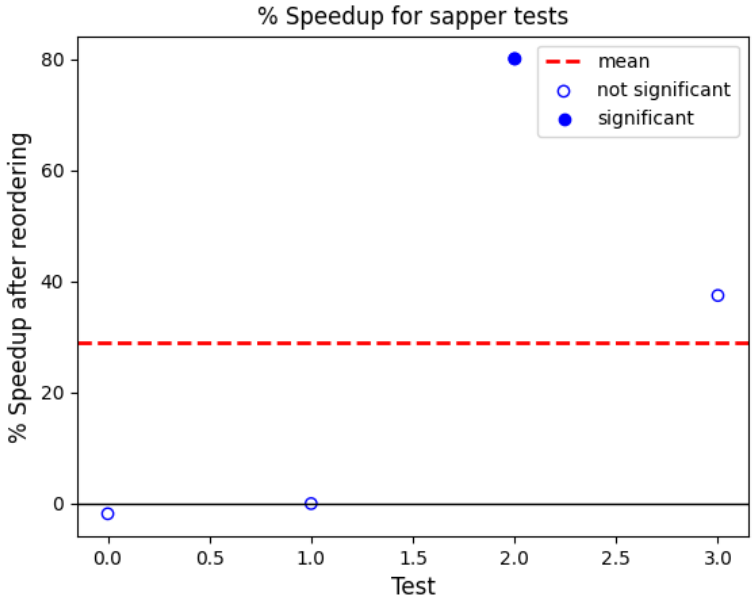

Figure 20: Average percentage speedups for all sapper tests

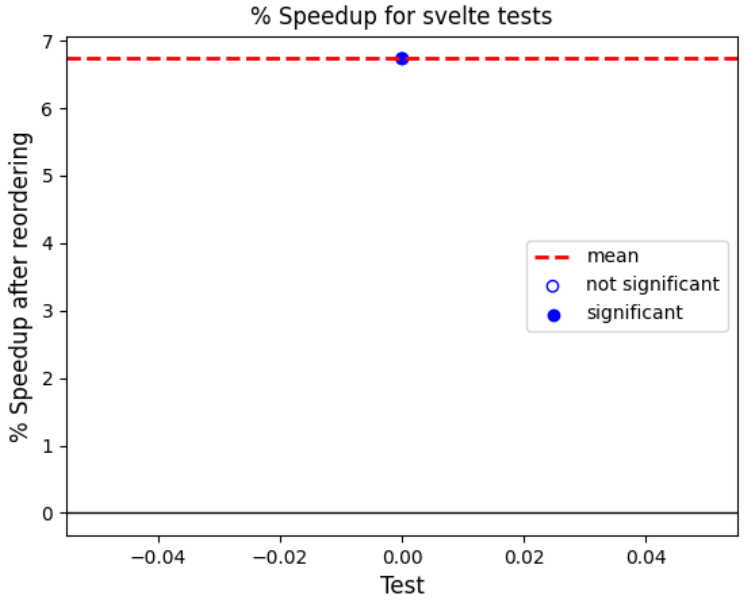

Figure 21: Average percentage speedups for all svelte tests

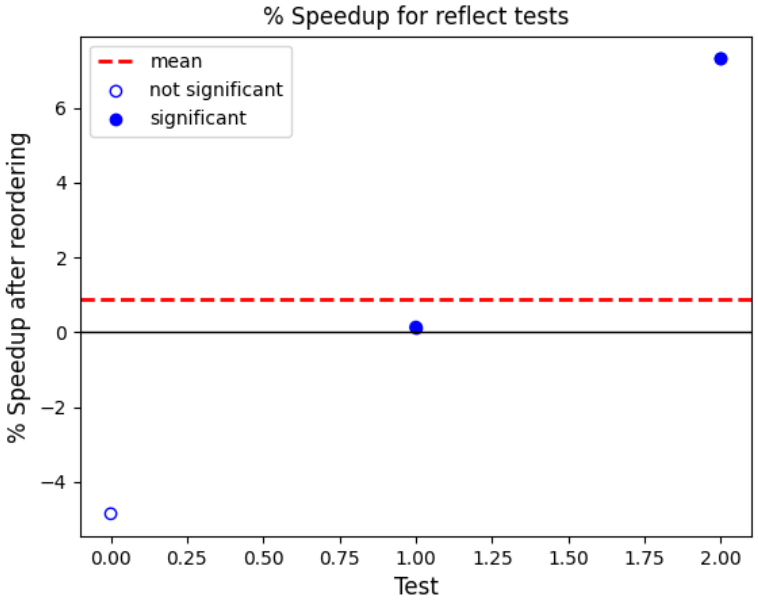

Figure 22: Average percentage speedups for all reflect tests

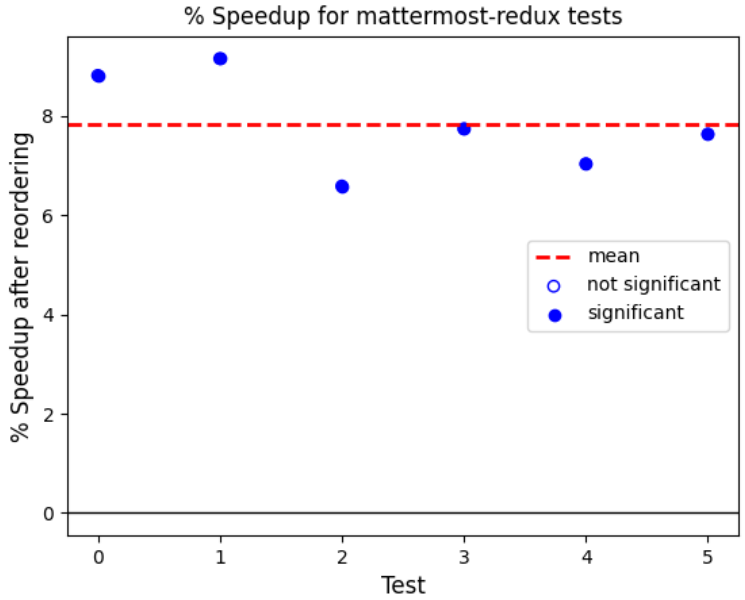

Figure 23: Average percentage speedups for all mattermost-redux tests

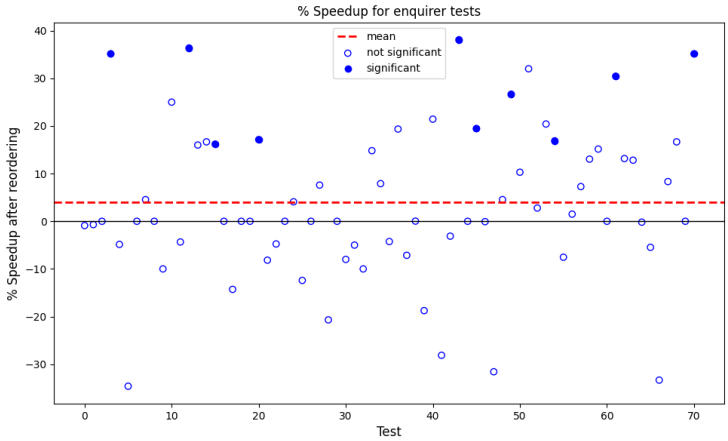

Figure 24: Average percentage speedups for all enquirer tests

4 Runtimes for all experiment runs of select Kactus tests

In the paper, we include a graph of all the runtimes for all experiment runs of Kactus test 117, both with and without reordering, along with the means for each. Here we include the same graph for a few more tests, just to show that the trend is consistent. We also include the graph for test 117 again, for completeness.

Specifically, note how there are consistently more (very slow) outlier tests in the code without reordering. In the paper, we discuss our theory for why this could be the case: Our conjecture is that this is because without the reorderings it is possible to get the sum of the worst cases for each file system access, whereas in the reordered case the effect of the existing variability in file system access is "muted", since more computations can execute in parallel.

When we provide the artifact, we will also provide a simple interface to generate these graphs for any of the 172 tests in Kactus (or for any tests in any of the other projects).

Figure 25: Runtimes for all experiment runs of Kactus test 22

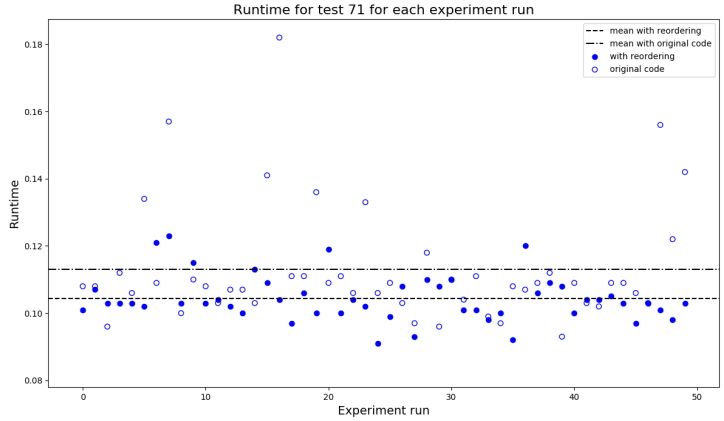

Figure 26: Runtimes for all experiment runs of Kactus test 71

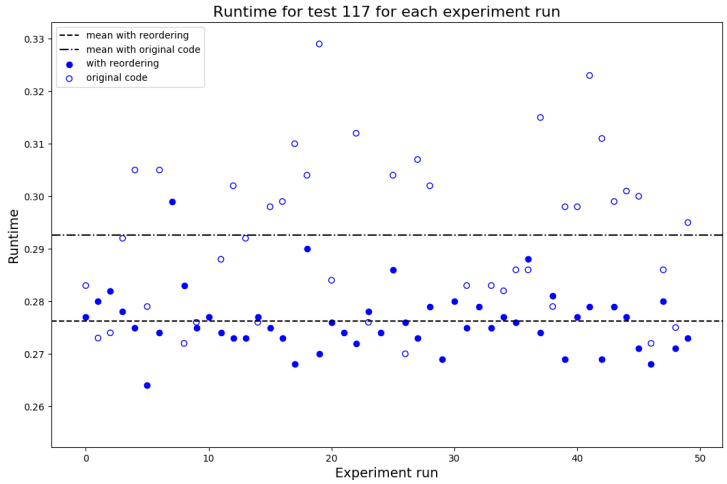

Figure 27: Runtimes for all experiment runs of Kactus test 117

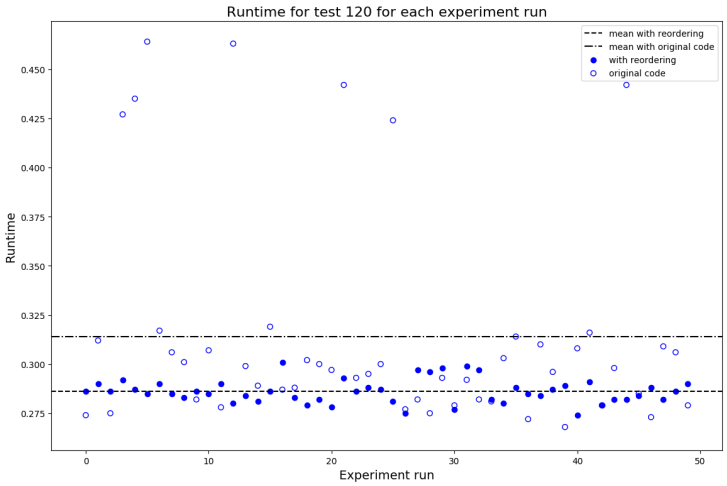

Figure 28: Runtimes for all experiment runs of Kactus test 120

5 Runtimes for all experiment runs of select network-dependent tests

Following are graphs analogous to those in the previous section, for some network-dependent tests. We have chosen test 2 from `sapper`, test 5 from `mattermost-redux`, and test 1 from `sapper` as these represent tests with a large, average, and negligible speedup respectively. We see, as with the runtimes for the `kactus` tests shown above, that the variance of the runtimes with reorderings is smaller than that of the original code. The exception is in the graph of `sapper` test 1, where the effect of the reordering is negligible and the runtime variance looks consistent across both reordered and original code. This supports our conjecture that the reordering "mutes" the effect of the existing variability in network call times.

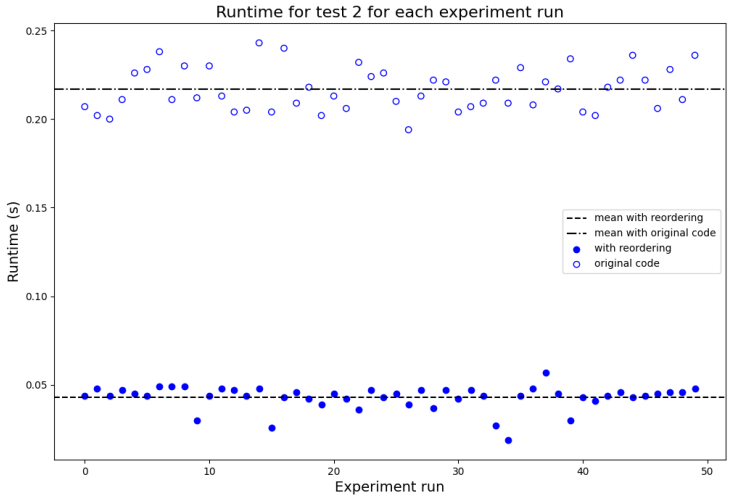

Figure 29: Runtimes for all experiment runs of Sapper test 2

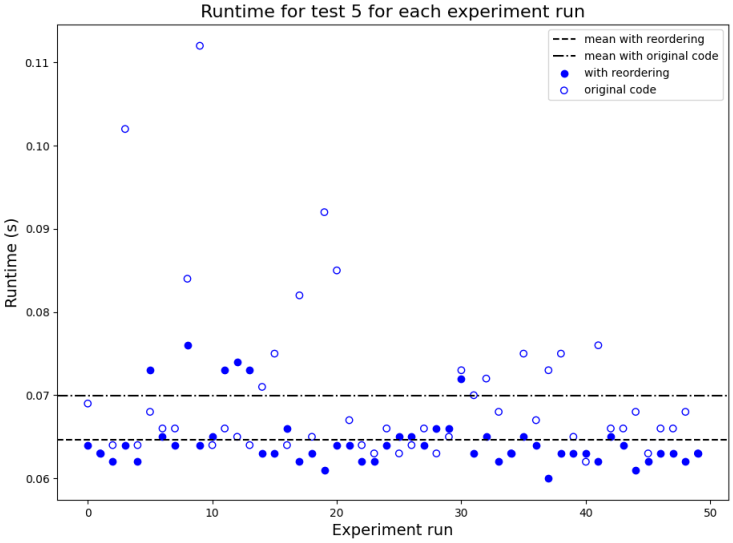

Figure 30: Runtimes for all experiment runs of Mattermost-Redux test 5

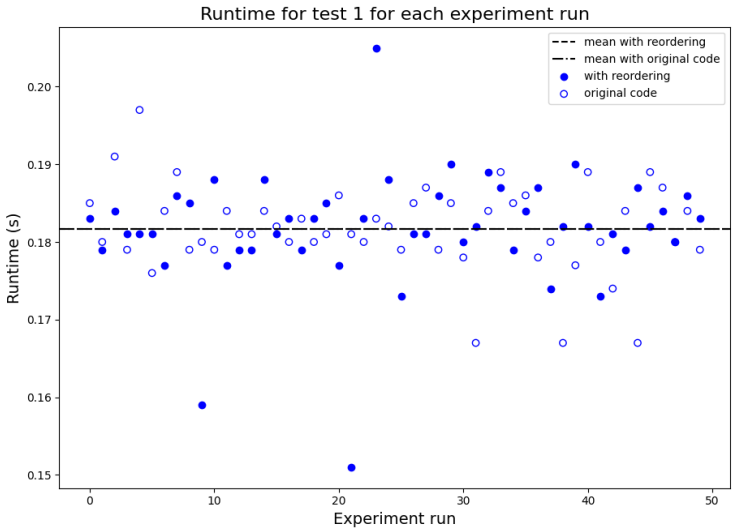

Figure 31: Runtimes for all experiment runs of Sapper test 1

6 Functions with environment-specific REF side effects

The following table is a list of all the functions with environment-specific REF side effects. It is analogous to Table 1 in the paper. However, there is one important difference: with our implementation, the list of functions with environment-specific MOD side effects is included explicitly as part of the analysis. All other functions originating from the environment-dependent packages were considered to have REF side effects specific to that environment. Therefore, this list is not coded into the analysis, and just represents all the functions we encountered in our evaluation. If we ran the analysis on more projects, then this list is liable to expand if we found other functions originating from the environment-dependent packages.

Environment	Function names
__FILE_SYSTEM__	fs.access
__FILE_SYSTEM__	fs.accessSync
__FILE_SYSTEM__	fs.basename
__FILE_SYSTEM__	fs.buffer
__FILE_SYSTEM__	fs.catch
__FILE_SYSTEM__	fs.cb
__FILE_SYSTEM__	fs.charAt
__FILE_SYSTEM__	fs.chmodSync
__FILE_SYSTEM__	fs.chown
__FILE_SYSTEM__	fs.close
__FILE_SYSTEM__	fs.closeSync
__FILE_SYSTEM__	fs.concat
__FILE_SYSTEM__	fs.createGunzip
__FILE_SYSTEM__	fs.createGzip
__FILE_SYSTEM__	fs.createInflate
__FILE_SYSTEM__	fs.createReadStream
__FILE_SYSTEM__	fs.createWriteStream
__FILE_SYSTEM__	fs.deflate
__FILE_SYSTEM__	fs.deflateSync
__FILE_SYSTEM__	fs.destroy
__FILE_SYSTEM__	fs.dir
__FILE_SYSTEM__	fs.dirSync
__FILE_SYSTEM__	fs.dirname
__FILE_SYSTEM__	fs.emptyDir
__FILE_SYSTEM__	fs.emptyDirSync
__FILE_SYSTEM__	fs.end
__FILE_SYSTEM__	fs.endsWith
__FILE_SYSTEM__	fs.ensureDir
__FILE_SYSTEM__	fs.ensureDirSync
__FILE_SYSTEM__	fs.ensureFile
__FILE_SYSTEM__	fs.ensureFileSync
__FILE_SYSTEM__	fs.ensureSymlink
__FILE_SYSTEM__	fs.ensureSymlinkSync
__FILE_SYSTEM__	fs.exists
__FILE_SYSTEM__	fs.existsSync
__FILE_SYSTEM__	fs.extname
__FILE_SYSTEM__	fs.fileExistsSync
__FILE_SYSTEM__	fs.fileSync
__FILE_SYSTEM__	fs.filter
__FILE_SYSTEM__	fs.find
__FILE_SYSTEM__	fs.flat
__FILE_SYSTEM__	fs.forEach
__FILE_SYSTEM__	fs.gunzipSync
__FILE_SYSTEM__	fs.hasOwnProperty
__FILE_SYSTEM__	fs.includes
__FILE_SYSTEM__	fs.indexOf
__FILE_SYSTEM__	fs.inflate
__FILE_SYSTEM__	fs.inflateSync
__FILE_SYSTEM__	fs.isAbsolute
__FILE_SYSTEM__	fs.isDirectory

__FILE_SYSTEM__	fs.isFile
__FILE_SYSTEM__	fs.join
__FILE_SYSTEM__	fs.lastIndexOf
__FILE_SYSTEM__	fs.localeCompare
__FILE_SYSTEM__	fs.lstat
__FILE_SYSTEM__	fs.lstatSync
__FILE_SYSTEM__	fs.makeTree
__FILE_SYSTEM__	fs.map
__FILE_SYSTEM__	fs.match
__FILE_SYSTEM__	fs.mkdtemp
__FILE_SYSTEM__	fs.mkdtempSync
__FILE_SYSTEM__	fs.mock
__FILE_SYSTEM__	fs.mockClear
__FILE_SYSTEM__	fs.mockImplementation
__FILE_SYSTEM__	fs.mockImplementationOnce
__FILE_SYSTEM__	fs.mockReset
__FILE_SYSTEM__	fs.mockResolvedValue
__FILE_SYSTEM__	fs.mockResolvedValueOnce
__FILE_SYSTEM__	fs.mockReturnValue
__FILE_SYSTEM__	fs.mockReturnValueOnce
__FILE_SYSTEM__	fs.name
__FILE_SYSTEM__	fs.nested
__FILE_SYSTEM__	fs.normalize
__FILE_SYSTEM__	fs.openSync
__FILE_SYSTEM__	fs.opendir
__FILE_SYSTEM__	fs.parse
__FILE_SYSTEM__	fs.pathEnv
__FILE_SYSTEM__	fs.pathExists
__FILE_SYSTEM__	fs.pathExistsSync
__FILE_SYSTEM__	fs.pathJoin
__FILE_SYSTEM__	fs.pathString
__FILE_SYSTEM__	fs.pipe
__FILE_SYSTEM__	fs.pop
__FILE_SYSTEM__	fs.push
__FILE_SYSTEM__	fs.read
__FILE_SYSTEM__	fs.readFile
__FILE_SYSTEM__	fs.readFileSync
__FILE_SYSTEM__	fs.readJSON
__FILE_SYSTEM__	fs.readJSONSync
__FILE_SYSTEM__	fs.readJson
__FILE_SYSTEM__	fs.readJsonSync
__FILE_SYSTEM__	fs.readdir
__FILE_SYSTEM__	fs.readdirSync
__FILE_SYSTEM__	fs.readlink
__FILE_SYSTEM__	fs.readlinkSync
__FILE_SYSTEM__	fs.realpath
__FILE_SYSTEM__	fs.realpathSync
__FILE_SYSTEM__	fs.reduce
__FILE_SYSTEM__	fs.relative
__FILE_SYSTEM__	fs.removeCallback
__FILE_SYSTEM__	fs.replace
__FILE_SYSTEM__	fs.require
__FILE_SYSTEM__	fs.resolve
__FILE_SYSTEM__	fs.setEncoding
__FILE_SYSTEM__	fs.shift
__FILE_SYSTEM__	fs.shouldThrow
__FILE_SYSTEM__	fs.slice
__FILE_SYSTEM__	fs.split
__FILE_SYSTEM__	fs.startsWith
__FILE_SYSTEM__	fs.stat

__FILE_SYSTEM__	fs.statSync
__FILE_SYSTEM__	fs.substr
__FILE_SYSTEM__	fs.substring
__FILE_SYSTEM__	fs.symlink
__FILE_SYSTEM__	fs.symlinkSync
__FILE_SYSTEM__	fs.sync
__FILE_SYSTEM__	fs.test
__FILE_SYSTEM__	fs.then
__FILE_SYSTEM__	fs.tmpNameSync
__FILE_SYSTEM__	fs.toJSON
__FILE_SYSTEM__	fs.toLowerCase
__FILE_SYSTEM__	fs.toString
__FILE_SYSTEM__	fs.toUpperCase
__FILE_SYSTEM__	fs.traverse
__FILE_SYSTEM__	fs.trim
__FILE_SYSTEM__	fs.true
__FILE_SYSTEM__	fs.utimesSync
__FILE_SYSTEM__	fs.watch
__FILE_SYSTEM__	fs.within
__NETWORK__	network.abort
__NETWORK__	network.createServer
__NETWORK__	network.end
__NETWORK__	network.express
__NETWORK__	network.forEach
__NETWORK__	network.get
__NETWORK__	network.listen
__NETWORK__	network.parse
__NETWORK__	network.parseLinkHeader
__NETWORK__	network.post
__NETWORK__	network.request
__NETWORK__	network.require
__NETWORK__	network.send
__NETWORK__	network.setKeepAlive
__NETWORK__	network.start_prefetching
__NETWORK__	network.start_server
__NETWORK__	network.startsWith
__NETWORK__	network.stopPropagation
__NETWORK__	network.stop_animation
__NETWORK__	network.stop_propagation

7 Average runtimes and speedup for every test with/without reorderings

This section presents the average raw run times (over the 50 experiment runs) with/without reorderings applied, for all projects we use in our evaluation. We also include the percentage speedup/slowdown. In the paper, we present the harmonic mean of the percentage effect of the reordering. We cannot compare the actual run times of the tests (since these run different code and are not expected to have the same execution time), and so there is no summary statistic with raw run times over all tests.

`kactus`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.1331	0.13162	1.11%
2	0.20818	0.1914	8.06%
3	0.20504	0.1911	6.80%
4	0.25982	0.24114	7.19%
5	0.23318	0.21572	7.49%
6	0.26636	0.24272	8.88%
7	0.15446	0.15122	2.10%
8	0.18838	0.18036	4.26%
9	0.34046	0.33418	1.84%
10	0.11784	0.1102	6.48%
11	0.21796	0.20422	6.30%
12	0.2102	0.19678	6.38%
13	0.12582	0.12044	4.28%
14	0.1899	0.18176	4.29%
15	0.29256	0.28066	4.07%
16	0.05338	0.05022	5.92%
17	0.07572	0.0703	7.16%
18	0.04276	0.03948	7.67%
19	0.0491	0.04494	8.47%
20	0.04274	0.0398	6.88%
21	0.08068	0.07656	5.11%
22	0.0509	0.048	5.70%
23	0.07346	0.06964	5.20%
24	0.05232	0.04574	12.58%
25	0.07764	0.07372	5.05%
26	0.03288	0.03242	1.40%
27	0.08056	0.07638	5.19%
28	0.11906	0.1105	7.19%
29	0.11684	0.11106	4.95%
30	0.1086	0.10106	6.94%
31	0.26508	0.24786	6.50%
32	0.21706	0.20584	5.17%
33	0.13238	0.12142	8.28%
34	0.15462	0.1405	9.13%
35	0.10134	0.0934	7.84%
36	0.11838	0.11142	5.88%
37	0.17944	0.1655	7.77%
38	0.17486	0.15998	8.51%
39	0.21864	0.19788	9.50%
40	0.17774	0.16038	9.77%
41	0.1592	0.1451	8.86%
42	0.16878	0.15414	8.67%
43	0.16624	0.15138	8.94%
44	0.25578	0.2353	8.01%
45	0.39398	0.36334	7.78%
46	0.3896	0.358	8.11%
47	0.32554	0.29688	8.80%
48	0.33772	0.30758	8.92%
49	0.23588	0.21724	7.90%

50	0.0892	0.08224	7.80%
51	0.16956	0.15566	8.20%
52	0.1495	0.13654	8.67%
53	0.16978	0.1557	8.29%
54	0.0631	0.04266	32.39%
55	0.0473	0.0441	6.77%
56	0.06262	0.05772	7.82%
57	0.09328	0.09064	2.83%
58	0.0562	0.05402	3.88%
59	0.04768	0.04584	3.86%
60	0.04598	0.0438	4.74%
61	0.04608	0.04366	5.25%
62	0.05494	0.05208	5.21%
63	0.08502	0.08022	5.65%
64	0.09746	0.09114	6.48%
65	0.07324	0.06894	5.87%
66	0.04994	0.04512	9.65%
67	0.06184	0.0554	10.41%
68	0.07108	0.0674	5.18%
69	0.07412	0.06922	6.61%
70	0.06932	0.06596	4.85%
71	0.09802	0.09598	2.08%
72	0.11296	0.10432	7.65%
73	0.1327	0.12428	6.35%
74	0.13594	0.12758	6.15%
75	0.11386	0.10648	6.48%
76	0.0776	0.07306	5.85%
77	0.05362	0.04614	13.95%
78	0.05058	0.05348	-5.73%
79	0.08942	0.07982	10.74%
80	0.07674	0.08012	-4.40%
81	0.07182	0.06468	9.94%
82	0.04904	0.04638	5.42%
83	0.06548	0.0614	6.23%
84	0.02708	0.02782	-2.73%
85	0.08564	0.08012	6.45%
86	0.08354	0.07668	8.21%
87	0.04538	0.04292	5.42%
88	0.04862	0.0452	7.03%
89	0.08262	0.0765	7.41%
90	0.09706	0.08978	7.50%
91	0.06164	0.06214	-0.81%
92	0.06686	0.06226	6.88%
93	0.05524	0.04922	10.90%
94	0.04328	0.03486	19.45%
95	0.34636	0.30814	11.03%
96	0.34972	0.30952	11.49%
97	0.35406	0.31446	11.18%
98	0.42578	0.3875	8.99%
99	0.41266	0.37508	9.11%
100	0.41452	0.37302	10.01%
101	0.8916	0.81618	8.46%
102	0.86072	0.79422	7.73%
103	0.88066	0.8147	7.49%
104	0.44774	0.41532	7.24%
105	0.4473	0.41592	7.02%
106	0.45178	0.41066	9.10%
107	0.48184	0.42294	12.22%
108	0.4972	0.45138	9.22%
109	0.48976	0.44466	9.21%

110	0.47152	0.4245	9.97%
111	0.47898	0.4215	12.00%
112	0.49398	0.45574	7.74%
113	0.48124	0.4348	9.65%
114	0.4846	0.4381	9.60%
115	0.48288	0.42494	12.00%
116	0.48678	0.44366	8.86%
117	0.45308	0.40984	9.54%
118	0.31496	0.27624	12.29%
119	0.33832	0.2943	13.01%
120	0.411	0.3759	8.54%
121	0.3139	0.28612	8.85%
122	0.17474	0.16362	6.36%
123	0.14336	0.13216	7.81%
124	0.05738	0.0546	4.84%
125	0.2409	0.22242	7.67%
126	0.14084	0.1317	6.49%
127	0.18184	0.166	8.71%
128	0.05446	0.05252	3.56%
129	0.06212	0.05744	7.53%
130	0.04884	0.04486	8.15%
131	0.22974	0.2309	-0.50%
132	0.04722	0.04246	10.08%
133	0.0562	0.0516	8.19%
134	0.17852	0.17422	2.41%
135	0.04476	0.04118	8.00%
136	0.0555	0.0514	7.39%
137	0.04562	0.04166	8.68%
138	0.05656	0.05218	7.74%
139	0.0433	0.0392	9.47%
140	0.04428	0.04282	3.30%
141	0.04786	0.04356	8.98%
142	0.16844	0.1658	1.57%
143	0.2125	0.20994	1.20%
144	0.2622	0.2614	0.31%
145	0.11644	0.11	5.53%
146	0.13044	0.11972	8.22%
147	0.50114	0.47434	5.35%
148	0.29272	0.28866	1.39%
149	0.2913	0.28656	1.63%
150	0.30152	0.27192	9.82%
151	0.0554	0.04828	12.85%
152	0.09366	0.08412	10.19%
153	0.0935	0.08196	12.34%
154	0.0514	0.0446	13.23%
155	0.04794	0.04114	14.18%
156	0.15924	0.14872	6.61%
157	0.11172	0.10538	5.67%
158	0.31796	0.3199	-0.61%
159	0.11258	0.10806	4.01%
160	0.10982	0.10606	3.42%
161	0.10094	0.09726	3.65%
162	0.10252	0.09894	3.49%
163	0.0959	0.0918	4.28%
164	0.0418	0.0395	5.50%
165	0.04418	0.04324	2.13%
166	0.04348	0.04158	4.37%
167	0.0223	0.0206	7.62%
168	0.0456	0.04082	10.48%
169	0.03	0.02806	6.47%

170	0.02498	0.02242	10.25%
171	0.02502	0.0231	7.67%
172	0.20074	0.20342	-1.34%

`nodemonorepo`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.01055333333	0.01054	0.13%
2	0.00446	0.004453333333	0.15%
3	0.00188	0.001866666667	0.71%
4	0.004753333333	0.0048	-0.98%
5	0.0005133333333	0.0004933333333	3.90%
6	0.0006933333333	0.0006733333333	2.88%
7	0.0005466666667	0.00056	-2.44%
8	0.002046666667	0.0021	-2.61%
9	0.00182	0.00174	4.40%
10	0.003606666667	0.00348	3.51%
11	0.00224	0.00212	5.36%
12	0.00388	0.003793333333	2.23%

`desktop`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.2769	0.25674	7.28%
2	0.26952	0.25004	7.23%
3	0.3348	0.30834	7.90%
4	0.2925	0.26206	10.41%
5	0.34652	0.317	8.52%
6	0.16854	0.15402	8.62%
7	0.1897	0.17012	10.32%
8	0.41494	0.4138	0.27%
9	0.13436	0.11772	12.38%
10	0.24182	0.22088	8.66%
11	0.23434	0.20958	10.57%
12	0.141	0.12654	10.26%
13	0.21168	0.19194	9.33%
14	0.08954	0.0832	7.08%
15	0.0936	0.08222	12.16%
16	0.09246	0.08468	8.41%
17	0.13254	0.11704	11.69%
18	0.45762	0.441	3.63%
19	0.06596	0.06682	-1.30%
20	0.08826	0.08466	4.08%
21	0.13478	0.12284	8.86%
22	0.35134	0.3224	8.24%
23	0.41278	0.37678	8.72%
24	0.05594	0.05326	4.79%
25	0.06134	0.0584	4.79%
26	0.05532	0.0522	5.64%
27	0.09322	0.08706	6.61%
28	0.049	0.04256	13.14%
29	0.08504	0.07692	9.55%
30	0.0664	0.05758	13.28%
31	0.0896	0.08218	8.28%
32	0.0315	0.02802	11.05%
33	0.09044	0.08402	7.10%
34	0.14628	0.13004	11.10%
35	0.14072	0.13082	7.04%

36	0.13562	0.12232	9.81%
37	0.33518	0.31556	5.85%
38	0.281	0.26688	5.02%
39	0.14784	0.13644	7.71%
40	0.1723	0.15572	9.62%
41	0.1184	0.10922	7.75%
42	0.1342	0.12256	8.67%
43	0.2155	0.19724	8.47%
44	0.21024	0.18962	9.81%
45	0.2538	0.23388	7.85%
46	0.2139	0.19788	7.49%
47	0.19262	0.17842	7.37%
48	0.20012	0.18332	8.39%
49	0.20324	0.18728	7.85%
50	0.28862	0.26236	9.10%
51	0.45314	0.41196	9.09%
52	0.45666	0.40894	10.45%
53	0.36536	0.33082	9.45%
54	0.37894	0.33746	10.95%
55	0.26704	0.23868	10.62%
56	0.11716	0.1127	3.81%
57	0.1966	0.17702	9.96%
58	0.16952	0.15532	8.38%
59	0.19872	0.17944	9.70%
60	0.06254	0.05696	8.92%
61	0.05844	0.0537	8.11%
62	0.07366	0.06826	7.33%
63	0.0748	0.06892	7.86%
64	0.07594	0.06952	8.45%
65	0.0729	0.0671	7.96%
66	0.07282	0.06656	8.60%
67	0.07334	0.0678	7.55%
68	0.0753	0.06828	9.32%
69	0.07518	0.07004	6.84%
70	0.07128	0.06548	8.14%
71	0.10054	0.09166	8.83%
72	0.0651	0.0593	8.91%
73	0.04738	0.0441	6.92%
74	0.07138	0.07246	-1.51%
75	0.0613	0.0576	6.04%
76	0.06058	0.05728	5.45%
77	0.05982	0.05656	5.45%
78	0.07378	0.0665	9.87%
79	0.11094	0.10442	5.88%
80	0.13114	0.1223	6.74%
81	0.10178	0.09376	7.88%
82	0.0626	0.07116	-13.67%
83	0.07668	0.0701	8.58%
84	0.0981	0.09362	4.57%
85	0.0954	0.09164	3.94%
86	0.09256	0.08798	4.95%
87	0.13088	0.1246	4.80%
88	0.14376	0.13186	8.28%
89	0.16624	0.14914	10.29%
90	0.17258	0.15544	9.93%
91	0.1391	0.1232	11.43%
92	0.09596	0.08668	9.67%
93	0.0691	0.06076	12.07%
94	0.04588	0.04428	3.49%
95	0.09692	0.08942	7.74%

96	0.09686	0.09234	4.67%
97	0.1996	0.17634	11.65%
98	0.06382	0.06006	5.89%
99	0.0791	0.07358	6.98%
100	0.0285	0.0184	35.44%
101	0.11408	0.10842	4.96%
102	0.10782	0.10076	6.55%
103	0.0609	0.0576	5.42%
104	0.0634	0.05996	5.43%
105	0.10662	0.1004	5.83%
106	0.12484	0.1169	6.36%
107	0.07716	0.07728	-0.16%
108	0.08132	0.07578	6.81%
109	0.0657	0.06494	1.16%
110	0.05228	0.04934	5.62%
111	0.29472	0.26118	11.38%
112	0.2963	0.25642	13.46%
113	0.29676	0.25888	12.76%
114	0.37022	0.32462	12.32%
115	0.38164	0.3361	11.93%
116	0.37126	0.32576	12.26%
117	0.95332	0.86058	9.73%
118	0.93448	0.84214	9.88%
119	0.94248	0.85114	9.69%
120	0.45456	0.40776	10.30%
121	0.4458	0.4021	9.80%
122	0.45514	0.4052	10.97%
123	0.43024	0.37942	11.81%
124	0.44132	0.38894	11.87%
125	0.4329	0.37998	12.22%
126	0.4313	0.3774	12.50%
127	0.44052	0.39238	10.93%
128	0.45124	0.40296	10.70%
129	0.44594	0.39086	12.35%
130	0.4357	0.388	10.95%
131	0.43576	0.3839	11.90%
132	0.44826	0.39896	11.00%
133	0.42972	0.37628	12.44%
134	0.30214	0.26382	12.68%
135	0.3074	0.26844	12.67%
136	0.35218	0.31286	11.16%
137	0.30208	0.26556	12.09%
138	0.21432	0.19518	8.93%
139	0.17396	0.1574	9.52%
140	0.07686	0.0721	6.19%
141	0.28856	0.26308	8.83%
142	0.1702	0.15968	6.18%
143	0.21716	0.1968	9.38%
144	0.06896	0.06444	6.55%
145	0.07556	0.07118	5.80%
146	0.06876	0.06014	12.54%
147	0.19746	0.1727	12.54%
148	0.06302	0.05878	6.73%
149	0.07044	0.06624	5.96%
150	0.21522	0.20166	6.30%
151	0.06202	0.05694	8.19%
152	0.0698	0.0653	6.45%
153	0.06604	0.05972	9.57%
154	0.0733	0.06862	6.38%
155	0.0574	0.05402	5.89%

156	0.05924	0.05502	7.12%
157	0.06298	0.06206	1.46%
158	0.11956	0.10994	8.05%
159	0.10874	0.09882	9.12%
160	0.15354	0.13858	9.74%
161	0.13484	0.12086	10.37%
162	0.16632	0.15012	9.74%
163	0.4498	0.40036	10.99%
164	0.35478	0.32346	8.83%
165	0.34316	0.30562	10.94%
166	0.34056	0.29784	12.54%
167	0.07016	0.0669	4.65%
168	0.1253	0.11558	7.76%
169	0.12262	0.11334	7.57%
170	0.06474	0.05976	7.69%
171	0.06216	0.0595	4.28%
172	0.20342	0.1914	5.91%
173	0.14286	0.13712	4.02%
174	0.38848	0.38266	1.50%
175	0.13534	0.12406	8.33%
176	0.13312	0.12184	8.47%
177	0.018	0.0179	0.56%
178	0.11092	0.10218	7.88%
179	0.0526	0.04952	5.86%
180	0.0555	0.05394	2.81%
181	0.05496	0.05028	8.52%
182	0.03086	0.02942	4.67%
183	0.05462	0.05312	2.75%
184	0.03926	0.03694	5.91%
185	0.03126	0.03022	3.33%
186	0.0362	0.03526	2.60%
187	0.0923	0.0778	15.71%

`fiddle`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.005973333333	0.005926666667	0.78%
2	0.002486666667	0.002073333333	16.62%

`nodemonorepo`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.01688	0.01632	3.32%
2	0.03158	0.03084	2.34%
3	0.02876	0.02814	2.16%
4	0.04948	0.0482	2.59%
5	0.04866	0.04744	2.51%
6	0.05048	0.04908	2.77%
7	0.08574	0.08232	3.99%
8	0.01174	0.01102	6.13%
9	0.0045	0.00454	-0.89%
10	0.01338	0.01198	10.46%
11	0.01354	0.01316	2.81%
12	0.02138	0.02046	4.30%
13	0.0221	0.02108	4.62%
14	0.02238	0.02232	0.27%
15	0.02284	0.02202	3.59%

`zapier-platform-cli`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	2.00992	0.22926	8.86%
2	0.07222	0.05494	2.39%

`wire-desktop`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.00102	0.00088	13.73%
2	0.00352	0.00352	0.00%
3	0.01014	0.0089	12.23%
4	0.01246	0.01288	-3.37%
5	0.01066	0.00882	17.26%
6	0.01078	0.01084	-0.56%
7	0.01366	0.01312	3.95%
8	0.03194	0.03006	5.89%
9	0.00134	0.00114	14.93%
10	0.01012	0.00954	5.73%
11	0.0015	0.00142	5.33%
12	0.00054	0.00048	11.11%
13	0.00184	0.0016	13.04%
14	0.00024	0.00036	-50.00%

`cspell`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.006144	0.005752	6.38%
2	0.00492	0.00464	5.69%
3	0.00169	0.00178	-5.33%
4	0.1296	0.12676	2.19%
5	0.0029	0.00324	-11.72%
6	0.00184	0.00158	14.13%
7	0.00726	0.00698	3.86%
8	0.00216	0.00232	-7.41%
9	0.00928	0.00904	2.59%
10	0.00644	0.0057	11.49%
11	0.00368	0.00344	6.52%
12	0.00518	0.00514	0.77%
13	0.00398	0.0037	7.04%
14	0.00782	0.00718	8.18%
15	0.01128	0.01054	6.56%
16	0.00412	0.00372	9.71%
17	0.01284	0.0125	2.65%
18	0.00336	0.00322	4.17%
19	0.01114	0.01094	1.80%
20	0.01172	0.01138	2.90%
21	0.02116	0.0201	5.01%
22	0.0216	0.02092	3.15%
23	0.00282	0.00264	6.38%
24	0.00314	0.00296	5.73%
25	0.03718	0.0358	3.71%
26	0.04404	0.04026	8.58%

`sourcecred`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)

1	0.00202	0.00186	7.92%
2	0.0015	0.00148	1.33%
3	0.0007	0.00084	-20.00%
4	0.00404	0.00374	7.43%
5	0.00614	0.00558	9.12%
6	0.00136	0.0014	-2.94%
7	0.0026	0.00232	10.77%
8	0.00276	0.0027	2.17%
9	0.00276	0.0023	16.67%
10	0.00106	0.00088	16.98%
11	0.00312	0.0033	-5.77%
12	0.00306	0.0029	5.23%
13	0.00326	0.00282	13.50%
14	0.00348	0.00328	5.75%
15	0.00526	0.00492	6.46%
16	0.00276	0.00246	10.87%
17	0.0092	0.00918	0.22%
18	0.13482	0.12382	8.16%
19	0.00842	0.00762	9.50%
20	0.01174	0.0109	7.16%
21	0.00452	0.00422	6.64%
22	0.0049	0.00474	3.27%
23	0.00592	0.00608	-2.70%
24	0.00382	0.00358	6.28%
25	0.00318	0.00352	-10.69%
26	0.00244	0.00234	4.10%
27	0.00322	0.00454	-40.99%
28	0.01198	0.00982	18.03%
29	0.01544	0.01232	20.21%

`bit`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	6.00E-05	4.00E-05	3.33%
2	0.00016	8.00E-05	5.00%
3	0.00038	8.00E-05	7.89%
4	0.00016	0.00018	-1.25%
5	0.00012	0.0001	1.67%
6	8.00E-05	0	10.00%
7	4.00E-05	2.00E-05	5.00%
8	4.00E-05	6.00E-05	-5.00%

`vscode-psl`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.00048	0.00036	25.00%
2	0.0004	0.00038	5.00%
3	0.00052	0.00052	0.00%
4	0.00188	0.0018	4.26%
5	0.00542	0.00438	19.19%
6	0.00148	0.00154	-4.05%
7	0.0023	0.00292	-26.96%
8	0.00078	0.00078	0.00%
9	0.00226	0.00202	10.62%
10	0.00093	0.00085	8.60%
11	0.00058	0.00058	0.00%
12	0.0007	0.00074	-5.71%

13	0.0003	0.00028	6.67%
14	0.00026	0.00034	-30.77%
15	0.00048	0.00048	0.00%
16	0.0003	0.00034	-13.33%
17	0.00035	0.00037	-5.71%
18	0.00032	0.00034	-6.25%
19	0.00018	0.0003	-66.67%
20	0.0003	0.00038	-26.67%
21	0.00024	0.00022	8.33%
22	0.0003	0.0003	0.00%
23	0.0004	0.0004	0.00%
24	0.0009	0.00078	13.33%
25	0.00046	0.00044	4.35%
26	0.00044	0.00042	4.55%
27	0.0003	0.0002	33.33%
28	0.00016	0.00016	0.00%
29	0.00024	0.00016	33.33%
30	0.00058	0.00062	-6.90%
31	0.00046	0.00042	8.70%
32	0.00052	0.0004	23.08%
33	0.00064	0.0007	-9.38%
34	0.00044	0.0003	31.82%
35	0.0003	0.00034	-13.33%
36	0.00032	0.0003	6.25%
37	0.0002	0.00022	-10.00%
38	0.00032	0.00018	43.75%
39	0.0003	0.00032	-6.67%
40	0.00026	0.00016	38.46%
41	0.00058	0.00036	37.93%
42	0.00172	0.00172	0.00%
43	0.00028	0.00026	7.14%
44	0.00024	6.00E05	75.00%
45	0.00022	0.00016	27.27%
46	0.00053	0.00049	7.55%
47	0.00036	0.00028	22.22%
48	0.00088	0.00058	34.09%
49	0.00034	0.00022	35.29%
50	0.00032	0.00032	0.00%
51	0.00046	0.00036	21.74%
52	0.00026	0.00032	-23.08%
53	0.00038	0.00036	5.26%
54	0.00038	0.00036	5.26%
55	0.00022	0.00024	-9.09%
56	0.00052	0.00054	-3.85%
57	0.00078	0.00076	2.56%
58	0.00034	0.00044	-29.41%
59	0.0006	0.00061	-1.67%
60	0.00029	0.00034	-17.24%
61	0.00036	0.00027	25.00%
62	0.00029	0.00041	-41.38%
63	0.00054	0.00054	0.00%
64	0.00049	0.00054	-10.20%
65	0.00048	0.0004	16.67%
66	0.00035	0.00048	-37.14%
67	0.00017	0.00018	-5.88%
68	0.00041	0.00043	-4.88%
69	0.00051	0.00052	-1.96%
70	0.0003	0.00019	36.67%
71	0.00033	0.0004	-21.21%
72	0.0004	0.00038	5.00%

73	0.00028	0.00038	-35.71%
74	0.0005	0.00038	24.00%
75	0.00052	0.00044	15.38%
76	0.00038	0.00054	-42.11%
77	0.00038	0.0003	21.05%
78	0.00054	0.00052	3.70%
79	0.0059	0.0058	1.69%
80	0.01094	0.00968	11.52%
81	0.00074	0.0009	-21.62%
82	0.0003	0.00046	-53.33%
83	0.0008	0.0006	25.00%
84	0.02322	0.02066	11.02%
85	0.00046	0.00044	4.35%
86	0.00048	0.00034	29.17%
87	0.00076	0.00056	26.32%
88	0.00038	0.00046	-21.05%
89	0.00046	0.0003	34.78%
90	0.00102	0.00114	-11.76%
91	0.00068	0.00072	-5.88%
92	0.00042	0.0003	28.57%
93	0.00044	0.0005	-13.64%
94	0.00046	0.00052	-13.04%
95	0.00046	0.00038	17.39%
96	0.00038	0.00034	10.53%
97	0.00046	0.00036	21.74%
98	0.00026	0.00032	-23.08%
99	0.0005	0.00058	-16.00%
100	0.00076	0.00078	-2.63%
101	0.00028	0.00034	-21.43%
102	0.00018	0.00022	-22.22%
103	0.00052	0.0005	3.85%
104	0.0004	0.00028	30.00%
105	0.00034	0.0004	-17.65%
106	0.0002	0.00022	-10.00%
107	0.00046	0.00042	8.70%
108	0.00042	0.00048	-14.29%
109	0.00062	0.00052	16.13%
110	0.0006	0.00048	20.00%
111	0.0006	0.0006	0.00%
112	0.00032	0.0002	37.50%
113	0.0002	0.00026	-30.00%

`gatsby`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.00726	0.00669	7.85%
2	0.00423	0.00425	-0.47%
3	0.01399	0.01339	4.29%
4	0.01955	0.0176	9.97%
5	0.00367	0.00343	6.54%
6	0.01048	0.01049	-0.10%
7	0.00788	0.007	11.17%
8	0.00668	0.00706	-5.69%
9	0.00543	0.00473	12.89%
10	0.00571	0.00485	15.06%
11	0.00316	0.00301	4.75%
12	0.00453	0.00481	-6.18%
13	0.0038	0.00405	-6.58%
14	0.00153	0.00141	7.84%

15	0.01211	0.01334	-10.16%
16	0.34281	0.34835	-1.62%
17	0.06548	0.06393	2.37%
18	0.00627	0.00654	-4.31%
19	0.37569	0.37197	0.99%
20	0.04824	0.04773	1.06%
21	0.00617	0.00589	4.54%
22	0.00042	0.00024	42.86%
23	0.00046	0.00022	52.17%
24	0.00303	0.00286	5.61%
25	0.00423	0.00407	3.78%
26	0.00533	0.00555	-4.13%
27	0.01812	0.01812	0.00%
28	0.00083	0.00058	30.12%
29	0.00116	0.00112	3.45%
30	0.00101	0.0008	20.79%
31	0.00047	0.00034	27.66%
32	1.32068	1.31097	0.74%
33	0.01526	0.02069	-35.58%
34	0.00865	0.01069	-23.58%
35	0.00321	0.00291	9.35%
36	0.66681	0.56394	15.43%
37	0.29577	0.26601	10.06%
38	0.30846	0.26978	12.54%
39	0.0132	0.01254	5.00%
40	0.28989	0.2667	8.00%
41	0.28491	0.26135	8.27%
42	0.30076	0.28594	4.93%
43	0.27458	0.25635	6.64%

`jamserve`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.14963	0.14808	1.04%
2	0.00459	0.00453	1.31%
3	0.0797	0.07858	1.41%
4	0.07877	0.07728	1.89%
5	0.00495	0.00479	3.23%
6	0.00607	0.00591	2.64%
7	0.00317	0.00295	6.94%
8	0.00479	0.00473	1.25%
9	0.01196	0.01129	5.60%
10	0.11253	0.11112	1.25%
11	0.05648	0.05587	1.08%
12	1.01603	1.00316	1.27%
13	0.00467	0.00463	0.86%
14	0.0056	0.00565	-0.89%
15	0.00473	0.00469	0.85%
16	0.00359	0.00367	-2.23%
17	0.04489	0.04477	0.27%
18	0.00766	0.00751	1.96%
19	0.07852	0.07623	2.92%
20	0.07754	0.07568	2.40%
21	0.13414	0.13073	2.54%
22	0.00462	0.00457	1.08%
23	0.00462	0.00477	-3.25%
24	0.00636	0.00626	1.57%
25	0.00548	0.00546	0.36%
26	0.06377	0.06253	1.94%

27	0.01379	0.0135	2.10%
28	0.00556	0.00547	1.62%
29	0.00477	0.00459	3.77%
30	0.00794	0.00844	-6.30%
31	0.0377	0.04122	-9.34%
32	0.70936	0.70788	0.21%
33	0.00501	0.00494	1.40%
34	0.00488	0.00486	0.41%
35	0.1474	0.1473	0.07%
36	0.00441	0.00455	-3.17%
37	0.07741	0.07921	-2.33%
38	0.07647	0.0779	-1.87%
39	0.00463	0.00488	-5.40%
40	0.00487	0.00491	-0.82%
41	0.00531	0.00562	-5.84%
42	0.00302	0.00284	5.96%
43	0.00468	0.00474	-1.28%
44	0.00711	0.00697	1.97%
45	0.0167	0.0172	-2.99%
46	0.12041	0.12213	-1.43%
47	0.09123	0.09176	-0.58%
48	3.6676	3.68028	-0.35%
49	0.00469	0.00472	-0.64%
50	0.00594	0.00588	1.01%
51	0.00482	0.0048	0.41%
52	0.00346	0.00336	2.89%
53	0.04363	0.04429	-1.51%
54	0.00745	0.00758	-1.74%
55	0.07604	0.07777	-2.28%
56	0.07532	0.07679	-1.95%
57	0.13348	0.13605	-1.93%
58	0.00492	0.00491	0.20%
59	0.00499	0.00501	-0.40%
60	0.00458	0.00479	-4.59%
61	0.00471	0.0047	0.21%
62	0.00513	0.00496	3.31%
63	0.00475	0.00481	-1.26%
64	0.00467	0.00457	2.14%
65	0.00477	0.00478	-0.21%
66	0.00691	0.00695	-0.58%
67	0.00489	0.00495	-1.23%
68	0.06234	0.06375	-2.26%
69	0.01433	0.01437	-0.28%
70	0.00553	0.00565	-2.17%
71	0.00494	0.00498	-0.81%
72	0.00795	0.00729	8.30%
73	0.02065	0.01944	5.86%
74	2.1263	2.09582	1.43%
75	0.01727	0.0148	14.30%
76	0.00032	0.00035	-9.38%
77	0.00465	0.00498	-7.10%
78	0.00398	0.00434	-9.05%
79	0.01052	0.00913	13.21%
80	0.00695	0.00743	-6.91%
81	0.00549	0.00635	-15.66%
82	0.00372	0.00373	-0.27%
83	0.05681	0.0576	-1.39%
84	0.00975	0.00935	4.10%
85	0.00925	0.008	13.51%
86	0.00586	0.00627	-7.00%

87	0.00993	0.00984	0.91%
88	0.00566	0.00575	-1.59%
89	0.0064	0.00693	-8.28%
90	0.00589	0.00536	9.00%
91	0.00509	0.00481	5.50%
92	0.00406	0.00409	-0.74%
93	0.00316	0.00265	16.14%
94	0.0028	0.00297	-6.07%
95	0.0075	0.0072	4.00%
96	0.00419	0.00422	-0.72%
97	0.03853	0.03738	2.98%
98	0.00453	0.00475	-4.86%
99	0.02008	0.01984	1.20%
100	0.02177	0.02148	1.33%
101	0.00279	0.00303	-8.60%
102	0.00976	0.00952	2.46%
103	0.0719	0.06924	3.70%
104	0.33542	0.32725	2.44%
105	0.00408	0.00371	9.07%
106	0.05043	0.04456	11.64%
107	0.00834	0.00833	0.12%
108	0.02073	0.02034	1.88%
109	0.02083	0.02086	-0.14%
110	0.06601	0.06564	0.56%
111	0.02034	0.01977	2.80%
112	0.01389	0.01395	-0.43%
113	1.03906	1.03726	0.17%
114	0.02523	0.02214	12.25%
115	0.01685	0.01682	0.18%
116	1.11931	1.11702	0.20%
117	0.02401	0.02432	-1.29%
118	0.00733	0.00707	3.55%
119	0.08389	0.08261	1.53%
120	0.22085	0.2184	1.11%
121	0.00484	0.00476	1.65%
122	0.11726	0.1197	-2.08%
123	0.11763	0.1186	-0.82%
124	0.00562	0.00563	-0.18%
125	0.00333	0.00338	-1.50%
126	0.01237	0.01251	-1.13%
127	0.1498	0.15405	-2.84%
128	1.47914	1.47506	0.28%
129	0.00465	0.00463	0.43%
130	0.00703	0.00712	-1.28%
131	0.00364	0.00359	1.37%
132	0.04454	0.04443	0.25%
133	0.00797	0.00778	2.38%
134	0.11712	0.11685	0.23%
135	0.11675	0.11624	0.44%
136	0.18211	0.18261	-0.27%
137	0.09482	0.09606	-1.31%
138	0.0143	0.01456	-1.82%
139	0.00493	0.00481	2.43%
140	0.05549	0.05514	0.63%
141	0.00517	0.00507	1.93%
142	0.05858	0.0576	1.67%
143	0.00656	0.00737	-12.35%
144	0.19988	0.19581	2.04%
145	0.00648	0.00551	14.97%
146	0.13489	0.13171	2.36%

147	0.72975	0.72315	0.90%
148	0.14993	0.1454	3.02%
149	0.08435	0.08456	-0.25%
150	0.17697	0.17312	2.18%
151	0.00051	0.00051	0.00%
152	0.00407	0.00378	7.13%
153	0.02151	0.02073	3.63%
154	0.00569	0.00596	-4.75%
155	0.01083	0.01084	-0.09%
156	0.01071	0.01057	1.31%
157	0.00612	0.00605	1.14%
158	0.00346	0.00305	11.85%
159	0.01054	0.00999	5.22%
160	0.0537	0.05165	3.82%
161	0.19407	0.1904	1.89%
162	0.0037	0.00353	4.59%
163	0.04764	0.04691	1.53%
164	0.00785	0.00783	0.25%
165	0.01101	0.0108	1.91%
166	0.01097	0.01115	-1.64%
167	0.04604	0.04538	1.43%
168	0.0147	0.01465	0.34%
169	0.01395	0.01345	3.58%
170	0.00015	0.00012	20.00%
171	0.01977	0.02026	-2.48%
172	0.00565	0.00505	10.62%
173	0.01103	0.01076	2.45%
174	0.01181	0.0117	0.93%
175	0.00634	0.00701	-10.57%
176	0.00286	0.00288	-0.70%
177	0.01009	0.01134	-12.39%
178	0.0603	0.05997	0.55%
179	0.1914	0.19153	-0.07%
180	0.00371	0.00399	-7.55%
181	0.04807	0.04766	0.85%
182	0.0081	0.00814	-0.49%
183	0.01264	0.01131	10.52%
184	0.01208	0.01189	1.57%
185	0.05415	0.05323	1.70%
186	0.01527	0.01488	2.55%
187	0.01454	0.01388	4.54%
188	0.00013	0.0001	23.08%
189	0.00512	0.00513	-0.20%
190	0.00507	0.005	1.38%
191	0.02048	0.01973	3.66%
192	0.00519	0.00535	-3.08%
193	0.01055	0.01054	0.09%
194	0.01094	0.01091	0.27%
195	0.00484	0.00502	-3.72%
196	0.0057	0.00633	-11.05%
197	0.00355	0.00321	9.58%
198	0.01159	0.01265	-9.15%
199	0.05648	0.05486	2.87%
200	0.00496	0.00471	5.04%
201	0.0132	0.01271	3.71%
202	0.19266	0.19095	0.89%
203	0.00526	0.00504	4.18%
204	0.00582	0.00608	-4.47%
205	0.00579	0.00576	0.52%
206	0.00362	0.00361	0.28%

207	0.04698	0.04615	1.77%
208	0.00801	0.00775	3.25%
209	0.01024	0.0103	-0.59%
210	0.01068	0.01078	-0.94%
211	0.05124	0.04934	3.71%
212	0.01482	0.01483	-0.07%
213	0.01364	0.01324	2.93%
214	0.00735	0.00696	5.31%
215	0.00536	0.00544	-1.49%
216	0.70728	0.71411	-0.97%
217	0.04058	0.04024	0.84%
218	0.02611	0.02585	1.00%
219	0.00967	0.0094	2.79%
220	0.00668	0.00705	-5.54%
221	0.01085	0.01103	-1.66%
222	0.00917	0.00875	4.58%
223	0.64946	0.64474	0.73%
224	0.34565	0.34589	-0.07%
225	0.00437	0.00447	-2.29%
226	0.18285	0.18266	0.10%
227	0.18417	0.18393	0.13%
228	0.00548	0.00549	-0.18%
229	0.00329	0.0031	5.78%
230	0.00975	0.01013	-3.90%
231	0.19004	0.19506	-2.64%
232	2.28606	2.28213	0.17%
233	0.00371	0.00355	4.31%
234	0.04422	0.04467	-1.02%
235	0.00755	0.00771	-2.12%
236	0.18435	0.18476	-0.22%
237	0.18168	0.18258	-0.50%
238	0.25349	0.25278	0.28%
239	0.14033	0.14211	-1.27%
240	0.01411	0.01405	0.43%
241	0.00015	0.00012	20.00%
242	0.71719	0.7069	1.43%
243	0.09707	0.098	-0.96%
244	0.23496	0.23654	-0.67%
245	0.01131	0.0108	4.51%
246	0.01163	0.01229	-5.67%
247	0.00622	0.00567	8.84%
248	0.0034	0.00351	-3.24%
249	0.01124	0.01293	-15.04%
250	0.02734	0.02702	1.17%
251	0.05665	0.05604	1.08%
252	0.0086	0.00885	-2.91%
253	0.01179	0.01189	-0.85%
254	0.01252	0.012	4.15%
255	0.02072	0.01965	5.16%
256	0.01514	0.01614	-6.61%
257	0.01524	0.01491	2.17%
258	0.70046	0.69607	0.63%
259	0.10425	0.10396	0.28%
260	0.09634	0.09605	0.30%
261	0.09605	0.09444	1.68%
262	0.28925	0.28533	1.36%
263	0.10382	0.10259	1.18%
264	0.01805	0.01743	3.43%
265	0.338035	0.34109	-0.90%
266	0.22847	0.231385	-1.28%

267	0.39316	0.39762	-1.13%
268	0.23162	0.23235	-0.32%
269	0.39092	0.39255	-0.42%
270	0.334665	0.329125	1.66%
271	0.34095	0.342075	-0.33%
272	0.307885	0.311185	-1.07%

`get`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.4172	0.41896	-0.42%
2	0.39756	0.38404	3.40%
3	0.404	0.40034	0.91%

`cucumber-js`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.0013	0.00122	6.15%
2	0.00114	0.0011	3.51%
3	0.00092	0.00082	10.87%
4	0.00122	0.00104	14.75%
5	0.00096	0.00106	-10.42%
6	0.00136	0.0013	4.41%
7	0.00854	0.00852	0.23%
8	0.0039	0.00372	4.62%
9	0.0027	0.00256	5.19%
10	0.0008	0.0008	0.00%
11	0.00186	0.00182	2.15%
12	0.00238	0.00222	6.72%
13	0.00322	0.00266	17.39%
14	0.00186	0.00198	-6.45%
15	0.00154	0.00166	-7.79%
16	0.00174	0.00158	9.20%
17	0.00032	0.00012	62.50%

`sapper`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.16942	0.17248	-1.81%
2	0.18172	0.18168	0.02%
3	0.21678	0.04306	80.14%
4	0.00016	0.0001	37.50%

`svelte`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.0163	0.0152	6.75%

`reflect`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.01154	0.0121	-4.85%
2	1.1151	1.11358	0.14%
3	0.0093	0.00862	7.31%

`mattermost-redux`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.05376	0.04902	8.82%
2	0.0557	0.0506	9.16%
3	0.1011	0.09444	6.59%
4	0.07544	0.0696	7.74%
5	0.1077	0.10012	7.04%
6	0.06994	0.0646	7.64%

`enquirer`

Test	Avg. Time original (s)	Avg. Time with reorder (s)	Avg. speedup (%)
1	0.00444	0.00448	-0.90%
2	0.02286	0.02302	-0.70%
3	0.00084	0.00084	0.00%
4	0.00074	0.00048	35.14%
5	0.00206	0.00216	-4.85%
6	0.00052	0.0007	-34.62%
7	0.0006	0.0006	0.00%
8	0.00088	0.00084	4.55%
9	0.00034	0.00052	-52.94%
10	0.001	0.0011	-10.00%
11	0.00048	0.00036	25.00%
12	0.00046	0.00048	-4.35%
13	0.00044	0.00028	36.36%
14	0.0005	0.00042	16.00%
15	0.00048	0.0004	16.67%
16	0.0021	0.00176	16.19%
17	0.00036	0.00062	-72.22%
18	0.00042	0.00048	-14.29%
19	0.00042	0.00042	0.00%
20	0.00186	0.00186	0.00%
21	0.0014	0.00116	17.14%
22	0.00098	0.00106	-8.16%
23	0.00084	0.00088	-4.76%
24	0.00058	0.00082	-41.38%
25	0.00098	0.00094	4.08%
26	0.00137	0.00154	-12.41%
27	0.00062	0.00106	-70.97%
28	0.00158	0.00146	7.59%
29	0.00058	0.0007	-20.69%
30	0.00172	0.00172	0.00%
31	0.001	0.00108	-8.00%
32	0.0016	0.00168	-5.00%
33	0.0014	0.00154	-10.00%
34	0.00054	0.00046	14.81%
35	0.00076	0.0007	7.89%
36	0.00094	0.00098	-4.26%
37	0.00062	0.0005	19.35%
38	0.00056	0.0006	-7.14%
39	0.05984	0.05982	0.03%
40	0.00064	0.00076	-18.75%
41	0.00056	0.00044	21.43%
42	0.00064	0.00082	-28.13%
43	0.00064	0.00066	-3.13%
44	0.00042	0.00026	38.10%
45	0.00044	0.00062	-40.91%
46	0.00144	0.00116	19.44%

47	0.02598	0.026	-0.08%
48	0.00038	0.0005	-31.58%
49	0.00044	0.00042	4.55%
50	0.0009	0.00066	26.67%
51	0.00136	0.00122	10.29%
52	0.0005	0.00034	32.00%
53	0.00072	0.0007	2.78%
54	0.00098	0.00078	20.41%
55	0.00154	0.00128	16.88%
56	0.00106	0.00114	-7.55%
57	0.00134	0.00132	1.49%
58	0.0011	0.00102	7.27%
59	0.00092	0.0008	13.04%
60	0.00066	0.00056	15.15%
61	0.0014	0.0014	0.00%
62	0.00138	0.00096	30.43%
63	0.00076	0.00066	13.16%
64	0.00156	0.00136	12.82%
65	0.01088	0.0109	-0.18%
66	0.00622	0.00656	-5.47%
67	0.00042	0.00056	-33.33%
68	0.00144	0.00132	8.33%
69	0.00048	0.0004	16.67%
70	0.00032	0.00048	-50.00%
71	0.00074	0.00048	35.14%