COMPARISON OF OPTIMIZATIONS RUN ON QED

ARUN DEBRAY
AUGUST 9, 2013

This document contains a summary of the data collected when I ran tests of various optimizations along with Quick Error Detection on the `bzip2` testing file. Most of it has been automatically generated, and then annotated.

The tests were conducted in the following manner:

(1) The QED executable was made with the command `opt $(PRE_PASSES) -load Hello.dylib $(OPT) -hello -QED-Mode=15 $(POST_PASSES)`. Here, `PRE_PASSES` was a Makefile variable that contained the list of passes to be executed before QED, and `POST_PASSES` was a variable for the passes conducted after QED. Finally, `OPT` held the optimization level for the test. For each trial, the passes and optimization levels are given. Note that `opt` does not always run its passes in order; thus, I eventually switched to executing the pre-QED passes, the QED pass, and the post-QED pass over three separate calls to `opt`. The trials for which I did this are indicated below.

(2) At the same time, a non-QED executable was made with the command `opt $(PRE_PASSES) $(POST_PASSES)`. This executable was used to compare the times given by the QED and non-QED incarnations of `bzip2`.

(3) The programs were tested on the input of `input.graphic`. There are wildly differing numbers of tests, because I wanted extra data about some of the programs. To make the tests most fair, I used my computer lightly while they were run, and in particular never ran two tests simultaneously.

The passes specified are referenced at `http://llvm.org/docs/Passes.html`.

Here are the results of the trials. Some trials are excluded, a few because they were not very informative, and a few because they caused introduced errors in QED. The results are also displayed in the graph of Figure 1, which might be useful.

1. `baseline`. The objective of this test was to serve as a control; no optimizations are made, except for QED. This baseline was useful to establish how siginficant other optimizations are.

   This test was run at optimization level `-O0`. No passes were run before the QED pass. No passes were run after the QED pass.

   See Table 1 for the data for this test.

TABLE 1. Timings for the `baseline` test.

Trial	QED	Normal
1	7m 33.149s	1m 22.563s
2	7m 35.996s	1m 26.941s
Mean	7m 34.572s	1m 24.752s

2. `o3`. The objective of this test was to determine how significant the `-O3` pass was as an improvement from the baseline. It turned out to be sigificantly better, though later optimizations ran better at `-O0` than `-O3`. To see the precise list and order of passes called by this trial, it's possible to run `llvm-as < /dev/null | opt -O3 -disable-output -debug-pass=Arguments`.

   This test was run at optimization level `-O3`. No passes were run before the QED pass. No passes were run after the QED pass.

   See Table 2 for the data for this test.

TABLE 2. Timings for the `o3` test.

Trial	QED	Normal
1	5m 33.981s	0m 47.457s

3. `loop_vectorize`. I had wondered whether loop vectorization would be useful for QED, since EDDI generates instructions that could possibly be combined into vectors. However, this didn't do anything useful.

   This test was run at optimization level `-O0`. No passes were run before the QED pass. The following pass was run after the QED pass:

   ```
   -loop-vectorize
   ```

   See Table 3 for the data for this test.

TABLE 3. Timings for the `loop-vectorize` test.

Trial	QED	Normal
1	7m 37.808s	1m 22.432s

4. `std_link`. Here, I wanted to see how effective `-std-link-opts` was. It isn't mentioned in the list of passes, but I thought it might be useful because `bzip2` had a linking phase. The effect was small but helpful, so this pass was incorporated into some later trials (including the ones that seem to do the best). To see the list of passes this used, it is possible to call `llvm-as` and `opt` as in the `o3` trial, above.

   This test was run at optimization level `-O0`. The following pass was run before the QED pass:

   ```
   -std-link-opts
   ```

   No passes were run after the QED pass.

   See Table 4 for the data for this test.

TABLE 4. Timings for the `std-link` test.

Trial	QED	Normal
1	6m 13.948s	0m 42.558s
2	6m 21.420s	0m 43.633s
3	6m 20.817s	0m 41.588s
4	6m 21.900s	0m 44.052s
Mean	6m 19.521s	0m 42.958s

5. `slow_vectorize`. For this test, I wanted to know if vectorizing basic blocks would be helpful. It turned out to be fairly useless.

   This test was run at optimization level `-O0`. No passes were run before the QED pass. The following pass was run after the QED pass:

   ```
   -bb-vectorize
   ```

   See Table 5 for the data for this test.

TABLE 5. Timings for the `slow-vectorize` test.

Trial	QED	Normal
1	7m 32.263s	1m 22.628s
2	7m 31.799s	1m 23.979s
Mean	7m 32.031s	1m 23.303s

6. `comp_and_link`. With this test, both `-std-compile-opts` and `-std-link-opts` were run before QED. I did this because they were both useful and wanted to see how they interacted. Even though they contain many of the same passes, the effect was significant; this trial is among the best I found, and is probably the simplest, yet it doesn't use `-O3`.

   This test was run at optimization level `-O0`. The following passes were run before the QED pass:

`-std-compile-opts`
`-std-link-opts`

No passes were run after the QED pass.

See Table 6 for the data for this test.

TABLE 6. Timings for the `comp-and-link` test.

Trial	QED	Normal
1	4m 21.131s	0m 37.843s
2	4m 25.365s	0m 38.874s
Mean	4m 23.248s	0m 38.359s

7. `o3_std`. This trial ran the above one at `-O3`. Surprisingly, it did worse, though this was a theme among the very good optimizations.

   This test was run at optimization level `-O3`. The following passes were run before the QED pass:

   `-std-compile-opts`
   `-std-link-opts`

   No passes were run after the QED pass.

   See Table 7 for the data for this test.

TABLE 7. Timings for the `o3-std` test.

Trial	QED	Normal
1	4m 41.835s	0m 39.794s
2	4m 33.637s	0m 39.835s
Mean	4m 37.736s	0m 39.814s

8. `speed1`. Though this trial is the same as the previous one, I listed it separately as an attempt to narrow in on the best optimization. This was the first step, followed by `speed2`.

   This test was run at optimization level `-O3`. The following passes were run before the QED pass:

   `-std-compile-opts` `-std-link-opts`

   No passes were run after the QED pass.

   See Table 8 for the data for this test.

TABLE 8. Timings for the `speed1` test.

Trial	QED	Normal
1	4m 34.987s	0m 39.634s
2	4m 35.892s	0m 42.006s
3	4m 33.069s	0m 39.374s
4	4m 35.646s	0m 39.359s
5	4m 32.139s	0m 39.911s
6	4m 34.760s	0m 43.348s
7	4m 32.562s	0m 39.832s
8	4m 30.371s	0m 40.590s
9	4m 35.180s	0m 41.545s
10	4m 42.876s	0m 39.160s
Mean	4m 34.748s	0m 40.476s

9. `speed2`. This test added several loop optimizations to `-std-compile-opts` and `-std-link-opts`. Interestingly, it did very well, and is probably the one optimization I would recommend using. Additionally, running `-instcombine` has the side effect of eliminating indirect function calls, allowing global CFCSS to work properly in more programs (though this was irrelevant for `bzip2`, which has no indirect calls).

   This test was run at optimization level `-O0`. The following passes were run before the QED pass:

   ```
   -std-compile-opts
   -std-link-opts
   -reassociate
   -lcssa
   -loop-simplify
   -indvars
   -loop-reduce
   -loop-rotate
   -loop-unswitch
   -loop-deletion
   -loop-unroll
   -instcombine
   ```

   No passes were run after the QED pass.

   See Table 9 for the data for this test. As with most of the optimizations that did well, I ran a lot of trials on this one to try to understand how variable it could be.

TABLE 9. Timings for the `speed2` test.

Trial	QED	Normal
1	4m 24.520s	0m 37.711s
2	4m 20.882s	0m 37.740s
3	4m 22.627s	0m 38.572s
4	4m 21.071s	0m 44.469s
5	4m 20.909s	0m 38.891s
6	4m 15.010s	0m 37.355s
7	4m 14.087s	0m 37.355s
8	4m 14.089s	0m 37.319s
9	4m 19.916s	0m 37.501s
10	4m 17.847s	0m 37.360s
11	4m 20.884s	0m 37.966s
12	4m 23.149s	0m 38.710s
13	4m 21.099s	0m 38.664s
14	4m 14.930s	0m 37.042s
15	4m 28.927s	0m 38.892s
16	4m 24.520s	0m 37.983s
17	4m 24.047s	0m 38.244s
18	4m 24.794s	0m 41.454s
19	4m 26.157s	0m 40.551s
20	4m 30.200s	0m 39.774s
21	4m 21.842s	0m 38.910s
22	4m 22.710s	0m 37.995s
23	4m 25.247s	0m 39.163s
24	4m 22.723s	0m 40.395s
Mean	4m 21.758s	0m 38.751s

10. `loops`. This trial consisted of several loop optimizations called before QED. It did reasonably well on its own, and was incorporated into `speed2`, above.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

```
-lcssa
-loop-simplify
-indvars
-loop-reduce
-loop-rotate
-licm
-loop-unswitch
-loop-deletion
-loop-unroll
-instcombine
```

No passes were run after the QED pass.

See Table 10 for the data for this test.

TABLE 10. Timings for the `loops` test.

Trial	QED	Normal
1	6m 2.643s	1m 12.875s
2	6m 6.015s	1m 11.041s
3	6m 10.777s	1m 11.445s
4	6m 48.533s	1m 23.690s
5	6m 10.483s	1m 15.760s
Mean	6m 15.690s	1m 14.962s

11. `move_to_post`. For this trial, I attempted to run some optimizations after QED. These were chosen carefully so as to be unlikely to interfere with QED, but since it didn't help very much relative to `speed2`, I didn't end up using them.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    -vectorize-slp
    -vectorize-slp-aggressive
    ```

    The following passes were run after the QED pass:

    ```
    -mem2reg
    -block-placement
    ```

    See Table 11 for the data for this test.

12. `forced_timing`. This is a version of `speed2` in which `opt` was run in three separate passes, after I discovered that passes aren't always executed in order. Thus, since no passes were run after QED, I can claim that this trial handled it correctly, and introduced no errors through optimization. Additionally, it did about as well as `speed2`, which is good; I can get considerable gain while also preserving correctness.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    ```

TABLE 11. Timings for the `move-to-post` test.

Trial	QED	Normal
1	4m 20.703s	0m 38.601s
2	4m 19.790s	0m 38.784s
3	4m 17.924s	0m 39.992s
4	4m 32.185s	0m 38.648s
5	4m 30.344s	0m 43.435s
6	4m 23.165s	0m 39.949s
7	4m 30.730s	0m 39.795s
8	4m 27.861s	0m 39.589s
Mean	4m 25.338s	0m 39.849s

```
-lcssa
-loop-simplify
-indvars
-loop-reduce
-loop-rotate
-loop-unswitch
-loop-deletion
-loop-unroll
-instcombine
```

No passes were run after the QED pass.

See Table 12 for the data for this test.

TABLE 12. Timings for the `forced-timing` test.

Trial	QED	Normal
1	4m 21.294s	0m 39.391s
2	4m 22.968s	0m 43.345s
3	4m 29.991s	0m 41.852s
4	4m 29.680s	0m 40.064s
5	4m 25.807s	0m 39.195s
Mean	4m 25.948s	0m 40.769s

13. `block_placement`. Here, I tried isolating the effect of `-block-placement` to see if it did anything useful. This trial compiled the program in three separate phases, so that the passes run before QED didn't unintentionally get called afterwards. However, there was not a significant effect relative to `speed2`, so I didn't end up using this.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    The following pass was run after the QED pass:

`-block-placement`

See Table 13 for the data for this test.

TABLE 13. Timings for the `block-placement` test.

Trial	QED	Normal
1	4m 25.052s	0m 38.601s
2	4m 29.842s	0m 38.475s
3	4m 29.962s	0m 38.541s
4	4m 37.293s	0m 38.209s
5	4m 28.901s	0m 39.368s
Mean	4m 30.210s	0m 38.639s

14. `arg_promotion`. Similarly to the above, I tried running `-arg-promotion` after QED to see if it would help. This did well, but not better than `speed2`, which it was based upon.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    `-std-compile-opts`
    `-std-link-opts`
    `-reassociate`
    `-lcssa`
    `-loop-simplify`
    `-indvars`
    `-loop-reduce`
    `-loop-rotate`
    `-loop-unswitch`
    `-loop-deletion`
    `-loop-unroll`
    `-instcombine`

    The following pass was run after the QED pass:

    `-argpromotion`

    See Table 14 for the data for this test.

TABLE 14. Timings for the `arg-promotion` test.

Trial	QED	Normal
1	4m 22.264s	0m 40.940s
2	4m 22.674s	0m 39.033s
3	4m 22.271s	0m 38.695s
4	4m 25.876s	0m 38.823s
5	4m 21.268s	0m 38.900s
6	4m 21.039s	0m 38.205s
7	4m 15.418s	0m 40.082s
8	4m 12.403s	0m 36.952s
9	4m 18.950s	0m 37.846s
10	4m 20.882s	0m 42.624s
11	4m 21.252s	0m 39.384s
12	4m 18.849s	0m 40.330s
13	4m 21.913s	0m 38.579s
14	4m 19.209s	0m 37.231s
15	4m 15.863s	0m 37.083s
Mean	4m 20.009s	0m 38.980s

15. `simplify_libcalls`. Finally, this test tried executing `-simplify-libcalls` after `speed2`. The effect was once again not very strong.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    The following pass was run after the QED pass:

    ```
    -simplify-libcalls
    ```

    See Table 15 for the data for this test.

TABLE 15. Timings for the `simplify-libcalls` test.

Trial	QED	Normal
1	4m 21.020s	0m 39.181s
2	4m 15.405s	0m 37.683s
3	4m 21.295s	0m 37.510s
4	4m 19.222s	0m 39.457s
5	4m 23.616s	0m 38.606s
6	4m 37.569s	0m 46.803s
7	4m 41.392s	0m 44.564s
8	4m 21.271s	0m 43.794s
9	4m 26.729s	0m 39.350s
10	4m 22.492s	0m 39.670s
Mean	4m 25.001s	0m 40.662s

16. `function_attrs`. Another test of a flag after QED that didn't make all that much of a difference.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    The following pass was run after the QED pass:

    ```
    -functionattrs
    ```

    See Table 16 for the data for this test.

17. `dead_post`. This test tried to eliminate dead code after QED. I don't believe it interfered with QED, but since it didn't have much of an effect I didn't check completely and just didn't use it. Stripping symbols

TABLE 16. Timings for the `function-attrs` test.

Trial	QED	Normal
1	4m 17.810s	0m 38.378s
2	4m 25.996s	0m 40.421s
3	4m 22.552s	0m 41.252s
4	4m 21.741s	0m 37.941s
5	4m 33.686s	0m 37.471s
Mean	4m 24.357s	0m 39.093s

and debug information might be useful if it were necessary to worry about the size of the program, but in the unlikely event that is necessary it could be better done with `-Os`, which actually caused a speedup.

This test was run at optimization level `-O0`. No passes were run before the QED pass. The following passes were run after the QED pass:

```
-deadargelim
-dse
-deadtypeelim
-strip-dead-debug-info
-strip-dead-prototypes
```

See Table 17 for the data for this test.

TABLE 17. Timings for the `dead-post` test.

Trial	QED	Normal
1	7m 30.812s	1m 26.030s

18. `codegenprepare`. Another flag that seemed like it might be useful for running after `speed2` and QED. It did well, though not necessarily better than `speed2`.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    The following pass was run after the QED pass:

    ```
    -codegenprepare
    ```

    See Table 18 for the data for this test.
19. `readonly`. In this test, the CFCSS and EDDI check functions were marked as readonly. I thought this might allow other optimizatons or the code generator to make them more efficient, but it didn't seem to have much of an effect.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    ```

TABLE 18. Timings for the `codegenprepare` test.

Trial	QED	Normal
1	4m 19.015s	0m 40.857s
2	4m 21.914s	0m 41.271s
3	4m 23.861s	0m 39.456s
4	4m 24.408s	0m 42.168s
5	4m 26.405s	0m 39.678s
Mean	4m 23.121s	0m 40.686s

```
-indvars
-loop-reduce
-loop-rotate
-loop-unswitch
-loop-deletion
-loop-unroll
-instcombine
```

The following pass was run after the QED pass:

```
-simplify-libcalls
```

See Table 19 for the data for this test.

TABLE 19. Timings for the `readonly` test.

Trial	QED	Normal
1	4m 21.771s	0m 44.479s
2	4m 18.678s	0m 38.463s
3	4m 28.956s	0m 37.792s
4	4m 26.036s	0m 41.521s
5	4m 27.647s	0m 40.218s
Mean	4m 24.618s	0m 40.495s

20. `fastcc`. In this trial, the check functions were called using the `fastcc` calling convention, rather than the C calling convention. I guessed that this would be helpful because of how often these functions are called, and it did seem to make a little bit of difference. Notice that the optimization passes are listed as the same as before, but the change was in the QED pass.

    This test was run at optimization level `-O0`. The following passes were run before the QED pass:

    ```
    -std-compile-opts
    -std-link-opts
    -reassociate
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    The following pass was run after the QED pass:

    ```
    -simplify-libcalls
    ```

    See Table 20 for the data for this test.
21. `std_compile`. This test, one of the earlier ones, ran just `-std-compile-opts`, gaining a significant speedup.

TABLE 20. Timings for the `fastcc` test.

Trial	QED	Normal
1	4m 18.611s	0m 37.572s
2	4m 21.918s	0m 39.365s
3	4m 24.725s	0m 38.704s
4	4m 13.229s	0m 37.050s
5	4m 13.366s	0m 36.955s
6	4m 16.011s	0m 37.174s
7	4m 21.226s	0m 37.560s
8	4m 16.209s	0m 38.516s
9	4m 14.108s	0m 37.383s
10	4m 14.847s	0m 37.341s
11	4m 14.022s	0m 37.318s
12	4m 14.308s	0m 37.320s
13	4m 14.050s	0m 37.265s
14	4m 14.562s	0m 37.274s
15	4m 13.863s	0m 37.283s
16	4m 14.149s	0m 37.335s
17	4m 15.270s	0m 37.233s
18	4m 15.519s	0m 37.267s
19	4m 23.647s	0m 38.241s
20	4m 18.102s	0m 37.601s
21	4m 20.833s	0m 37.171s
22	4m 21.532s	0m 38.555s
23	4m 17.324s	0m 38.777s
24	4m 12.952s	0m 37.016s
25	4m 15.202s	0m 37.706s
26	4m 14.592s	0m 37.422s
27	4m 19.916s	0m 37.386s
Mean	4m 16.818s	0m 37.622s

This test was run at optimization level `-O0`. The following pass was run before the QED pass:

```
-std-compile-opts
```

No passes were run after the QED pass.

See Table 21 for the data for this test.

TABLE 21. Timings for the `std-compile` test.

Trial	QED	Normal
1	5m 19.263s	0m 42.531s
2	5m 16.757s	0m 42.337s
3	5m 16.807s	0m 42.642s
4	5m 20.587s	0m 43.424s
5	5m 27.870s	0m 43.234s
6	5m 23.166s	0m 44.227s
Mean	5m 20.742s	0m 43.066s

22. `std_post`. Here, I tried running `-std-compile-opts` after QED. This didn't make much of a difference.

    This test was run at optimization level `-O0`. No passes were run before the QED pass. The following pass was run after the QED pass:

    ```
    -std-compile-opts
    ```

    See Table 22 for the data for this test.

TABLE 22. Timings for the `std-post` test.

Trial	QED	Normal
1	5m 7.325s	0m 42.499s

23. `o3_loops`. This optimization is identical to the `loops` test, but run at a higher optimization level. It did reasonably well, but was beaten by other optimizations later on.

    This test was run at optimization level `-O3`. The following passes were run before the QED pass:

    ```
    -lcssa
    -loop-simplify
    -indvars
    -loop-reduce
    -loop-rotate
    -loop-unswitch
    -loop-deletion
    -loop-unroll
    -instcombine
    ```

    No passes were run after the QED pass.

    See Table 23 for the data for this test.

TABLE 23. Timings for the `o3-loops` test.

Trial	QED	Normal
1	5m 37.408s	0m 46.652s
2	5m 37.316s	0m 45.756s
3	5m 38.521s	0m 44.758s
4	5m 37.175s	0m 45.076s
Mean	5m 37.605s	0m 45.560s

24. `post_bb`. Similarly to other post-QED trials, I tried to make some improvements after running QED. However, `-globalopt` merges duplicate globals and thus is incompatible with EDDI. This set should not be used.

    This test was run at optimization level `-O0`. No passes were run before the QED pass. The following passes were run after the QED pass:

    ```
    -block-placement
    -bb-vectorize
    -globalopt
    -mem2reg
    ```

    See Table 24 for the data for this test.

TABLE 24. Timings for the `post-bb` test.

Trial	QED	Normal
1	6m 46.208s	0m 45.294s

25. `vectorize`. For this test, I wanted to understand the effect of `-bb-vectorize`. It turned out to be not terribly different than just regular `-O3`.

    This test was run at optimization level `-O3`. No passes were run before the QED pass. The following pass was run after the QED pass:

    ```
    -bb-vectorize
    ```

    See Table 25 for the data for this test.

In Figure 1, the trials above are plotted. There are several clear groups: some trials had little to no effect, and are located in the upper-right corner, near the baseline trial. Other optimizations had some effects, but

TABLE 25. Timings for the `vectorize` test.

Trial	QED	Normal
1	5m 38.306s	0m 46.172s

not great ones, such as `loops` and `std_link`. However, when combined, some of these optimizations led to much better ones, which are expanded in Figure 2 below. Many of the best optimizations varied significantly over the range of Figure 2, so it's not clear that any single one is better than the others.

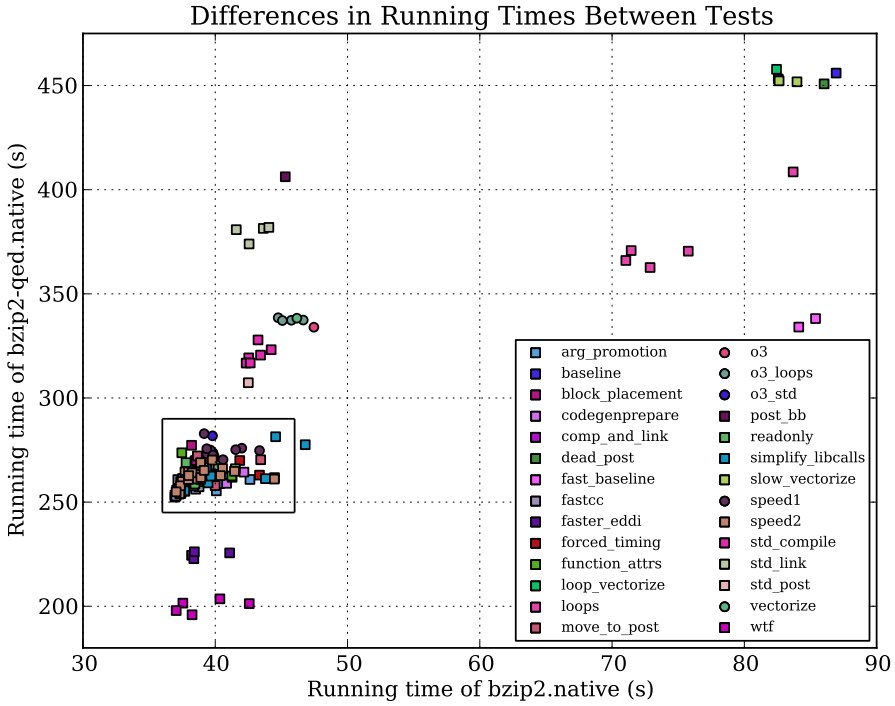

FIGURE 1. A plot of the gains due to different optimizations. Tests are as above; circular points correspond to using `-O3`, squares to `-O0`, and triangles to other options.

The best optimizations were given by `speed2` and slight modifications, though many of the things I tried after that didn't cause a significant improvement. Thus, I suggest doing the following:

- Setting up the list of passes from `speed2` before QED is added. Specifically, the following passes are used:

```
-std-compile-opts      -loop-simplify      -loop-unswitch
-std-link-opts         -indvars            -loop-deletion
-reassociate           -loop-reduce        -loop-unroll
-lcssa                 -loop-rotate        -instcombine
```

  `-instcombine` has the additional effect of cleaning up indirect calls in other test cases, which is a bonus.
- Adding `-simplify-libcalls` after QED. This has a small but nonzero effect, and doesn't appear to cause issues with QED itself.
- Setting the calling conventions to the check functions within QED. This can be done within `Hello.cpp` by adding the calling convention to calls to the check functions and to the functions themselves.

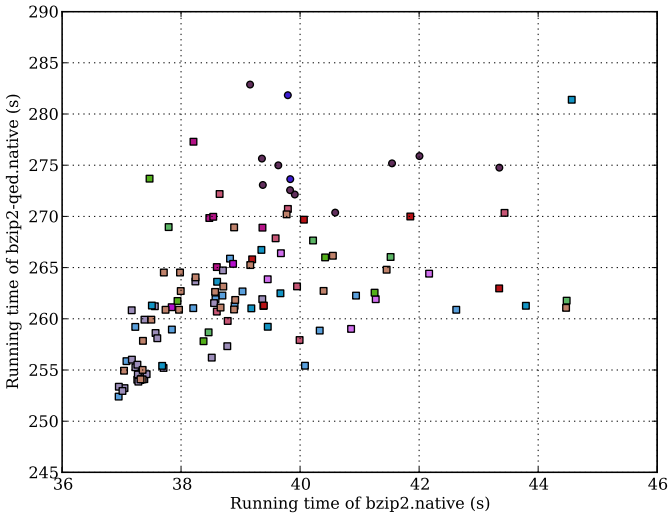

FIGURE 2. A zoomed-in version of the box in the lower-left-hand corner of Figure 1. Colors and meanings are the same as above.