Assignment 6: Compile Time vs. Speed

Note:

The test programs for this assignment are **func_test.py** and **loop_test.py** located in the root folder. The **-jit** option already worked in the previous version.

1. Experiment 1 - Function call testing

1.1 Source code

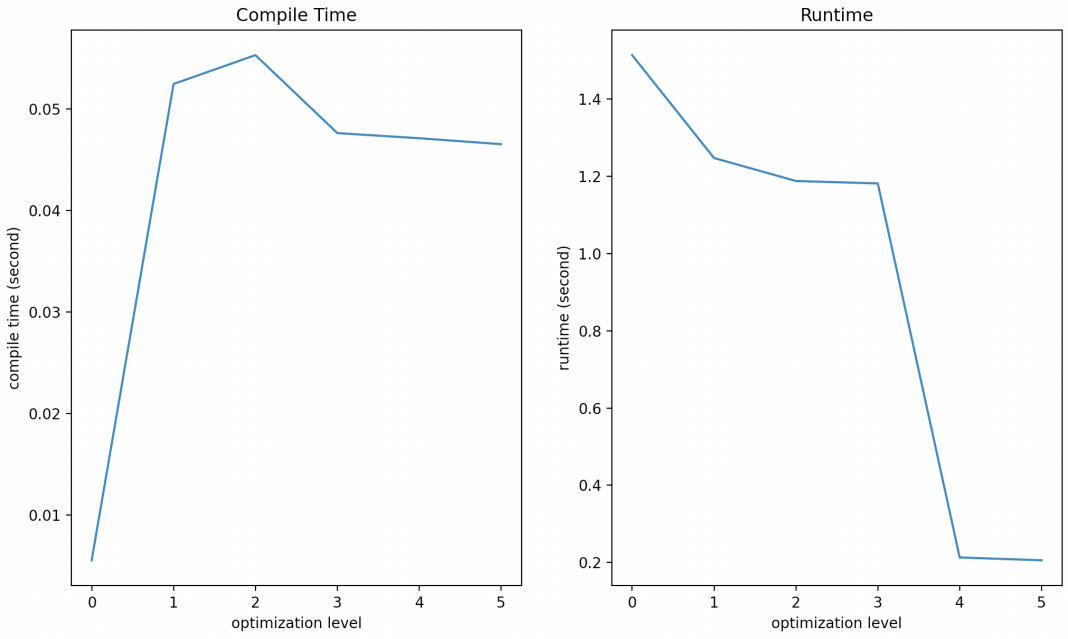
The source code below is defined in **test/func_test.ek**. There is a simple function test_func() which is called 2 * 10^9 times from the main function.

test > func_test.ek

```
1   def int test_func(int $n) {
2     int $ret = $n + 1;
3     return $ret;
4   }
5
6   def int run () {
7     int $res = 0;
8     int $i = 0;
9     while ($i < 2000000000) {
10      $res = test_func($i);
11      $i = $i + 1;
12    }
13    return 0;
14  }
```

1.2 Result

The result below is the average compile time and runtime for the source code above, we run the program on each optimization level 30 times to get the average result.



1.3 Discussion

	Optimizations
Level 0	No optimizations
Level 1	Add basic alias analysis pass
Level 2	Add function attrs pass
Level 3	Add function inlining pass
Level 4	Add licm pass
Level 5	Add all other optimization passes

There are 6 different types of pipelines, on each level, we added a new pass on the top of the previous level.

Based on the results, we can see the **licm pass** reduces the runtime by **1s**, the **basic alias analysis pass**, **function attrs pass** and **function inlining pass** also improve the performance by **0.2s**. The licm pass performs loop invariant code motion on the while loop. The function attrs pass and function inlining pass implemented a bottom-up traversal of the call graph, deduce the function attributes and inlining the functions to callees.

As for the compile time, adding these passes resulted **~0.05 s** increase on the compile time which is not much compared with the performance improvement.

2. Experiment 2 - Loop testing

1.1 Source code

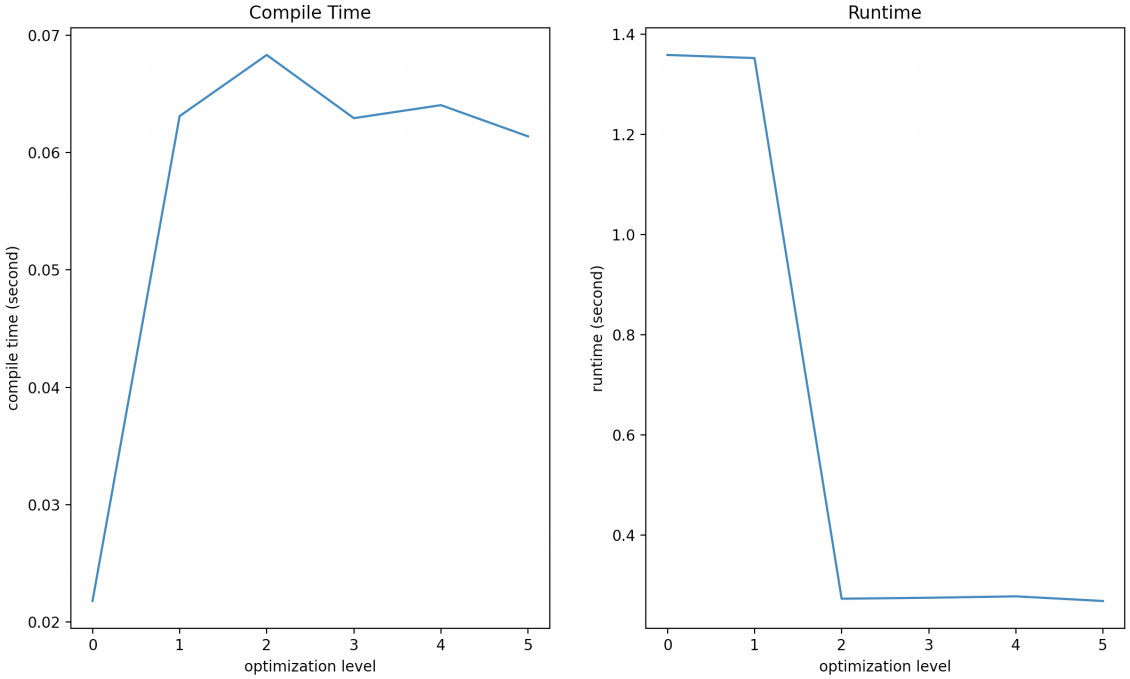
The source code defined in **test/loop_test.ek** is a simple while loop, it increases $i 10^9 times.

test > loop_test.ek

```
1  def int run () {
2    int $i = 0;
3
4    while ($i < 1000000000) {
5      $i = $i + 1;
6    }
7
8    return 0;
9  }
10
```

1.2 Result

The result below is the average compile time and runtime for the source code above, we run the program on each optimization level 30 times to get the average result.



1.3 Discussion

	Optimizations
Level 0	No optimizations
Level 1	Add basic alias analysis pass
Level 2	Add licm pass
Level 3	Add dead arg eliminations pass
Level 4	Add constant mergepass
Level 5	Add all other optimization passes

This is a simple example, the performance is improved by **licm pass** significantly, it reduces the runtime by **1.3s.** The licm pass performs loop invariant code motion, for this while loop, it reduces the time complexity from O(n) to O(1). But the compile time is increased by 0.6s due to the addition of optimization passes.