Assignment 6

Naga Nithin Manne & Dipti Sengupta
MPCS 51300, University of Chicago

Compile Time vs. Speed

We are going to investigate the following three optimization techniques on a few simple test programs

- Inlining
- InstCombine
- All Available Optimizations

Note: The resolution of the times are approximately 1 ms. It was not possible to get higher precision in Python without linking to Windows APIs.

Inlining

Test Program

```
extern int arg(int);

def int add(int $x, int $y) { return $x + $y; }

def int run() {
    int $n = arg(0);
    int $i = 0;
    int $j = 0;
    int $s = 0;
    while ($i < $n) {
        $j = 0;
        while ($j < $n) {
            $s = add($s, $i);
            $j = $j + 1;
        }
        $i = $i + 1;
    }
}
```

Function Inlining can be used to improve the performance of the call to *add*, which is done n^2 times in the test program. From the below graph, we can see that this optimization is very quick and offers performance improvements for even small values of n.

Time taken for Optimization: 0.0009965896606445312

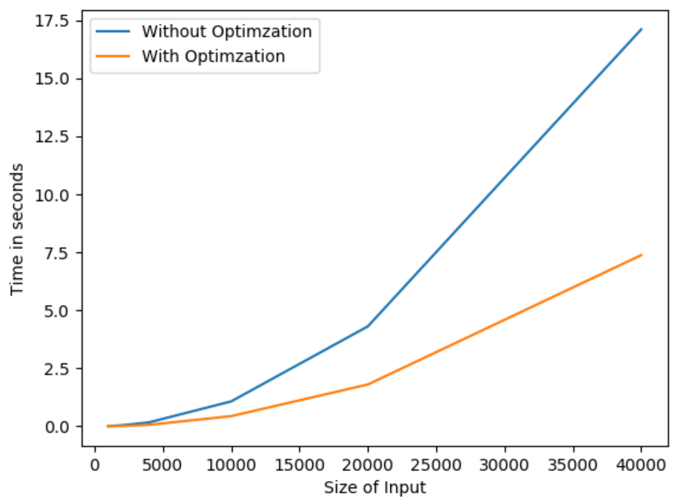

Figure 1: Inlining

InstCombine

Test Program

```
extern int arg(int);

def int run() {
    int $n = arg(0);
    int $i = 0;
    int $j = 0;
    int $s = 0;
    while ($i < $n) {
        $j = 0;
        while ($j < $n) {
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $s = $s + 1;
            $j = $j + 1;
        }
        $i = $i + 1;
    }
}
```

InstCombine can be used to combine multiple instructions like the additions above into a single instruction which can reduce the clock cycles needed to execute the program. Here, the cost of optimization is just high enough that its is more efficient to not perform optimization for smaller input sizes.

Time taken for Optimization: 0.0019943714141845703

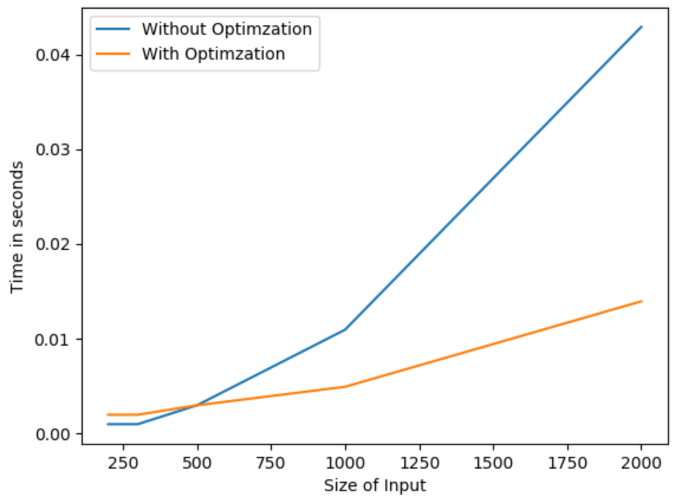

Figure 2: InstCombine

All Available Optimizations

Test Program

```
extern int arg(int);

def int fib(int $n) {
    if ($n == 0 || $n == 1) return $n;
    else return fib($n - 1) + fib($n - 2);
}

def int run() {
    int $n = arg(0);
    print fib($n);
}
```

In this test code, the fibonacci sequence cannot be optimized since the recursion algorithm has no possible optimizations. So in this scenario, the performance improvement due to optimization is minor as from the graph below.

Time taken for Optimization: 0.001962900161743164

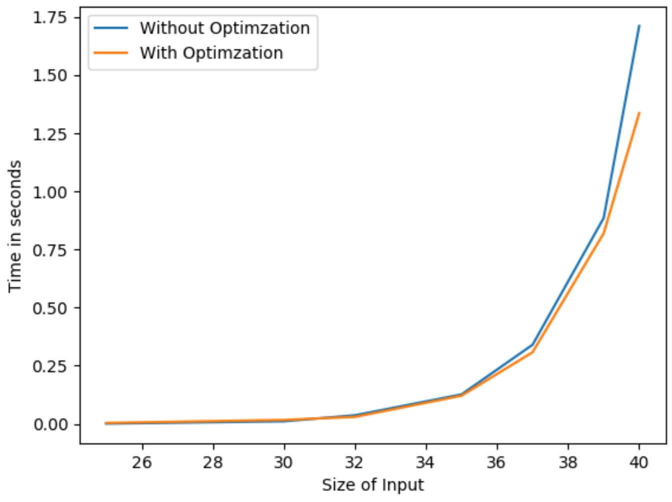

Figure 3: AllOptimzations