Analysis of GCC's optimization feature

Author: Jorge Andrés Pietra Santa Ochoa

October 27th, 2019

1 Abstract

When we refer to Compiler Optimization, we refer to the process through which a compiler tries to optimize of maximize some of the aspects of a computer program. Commonly, the goal is to minimize a program's execution time, memory requirements and even the power consumption of said computer program. This is all done throught the analysis and identification of a programming language's patterns in order to create a binary executable that is easier for the CPU to read, interpret and process.

2 Introduction

More often than not, programming language engineers make performance one of the most important aspects to take into account when creating and designing a compiler so that the programs created by the language are as fast as they can be. One of the most common ways to know just how fast the compiler is is throguh the process of "benchmarking", which is basically comparing the results of one product, service or process (in this case we're benchmarking a process) using some key "comparators" that will make it evident for whoever is analyzing the results which one is better. In this specific case of analysis, we will be comparing the execution times of the binaries created by GCC when no optimizations are done, compared to the time taken by the file created when using optimization level 3.

3 Objective

In this document, we will analyze the assembly code generated by GCC with no optimizations and the code generated when we use the optimization parameter on level 3. The results might not be drastically different, but we should be able to observe at least some difference between the two.

4 Development

To be as transparent as possible, here are the specs of the machine that will be used to run these tests:

Computer Model	Macbook Pro 15 Mid-2018
Operating System	macOS 10.15 Catalina
Procesor Number	8750H
#Cores	6
#Threads	12
Base Speed	2.20 GHz
Turbo Speed	4.10 GHz

Table 1: Machine specs

The machine we're using for these tests is what the industry considers "high-end", so the time it will take for our computer to execute the files could be considerably faster than the average machine, so it might be a good idea to run these tests on your computer to observe the difference it might make on your specific setup.

The code we will be analyzing in this document is fairly simple, but it should give us an insigth of how much of a difference optimization can really make. Given the following C function:

```
int proc(int a[]) {
        int sum = 0, i;
        for (i = 0; i < 1000000; i++)
                sum += a[i];
        return sum;
}
```

We will initialize a one-million-element array for the proc function to go through. We will later compile and execute the files generated by GCC.

First, let us take a look to both the unoptimized and optimized assembly codes generated by GCC, and see what the differences are:

5 Unoptimized Assembly Code:

```
./nonOpt:     file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:
__text:
100000ed0:   55          pushq   %rbp
100000ed1:   48 89 e5            movq    %rsp, %rbp
100000ed4:   48 89 7d f8         movq    %rdi, -8(%rbp)
100000ed8:   c7 45 f4 00 00 00 00    movl    $0, -12(%rbp)
100000edf:   c7 45 f0 00 00 00 00    movl    $0, -16(%rbp)
100000ee6:   81 7d f0 40 42 0f 00    cmpl    $1000000, -16(%rbp)
100000eed:   0f 8d 1f 00 00 00       jge     31 <_proc+0x42>
100000ef3:   48 8b 45 f8         movq    -8(%rbp), %rax
100000ef7:   48 63 4d f0         movslq  -16(%rbp), %rcx
100000efb:   8b 14 88            movl    (%rax,%rcx,4), %edx
100000efe:   03 55 f4            addl    -12(%rbp), %edx
```

```
100000f01:  89 55 f4          movl    %edx, -12(%rbp)
100000f04:  8b 45 f0          movl    -16(%rbp), %eax
100000f07:  83 c0 01          addl    $1, %eax
100000f0a:  89 45 f0          movl    %eax, -16(%rbp)
100000f0d:  e9 d4 ff ff ff    jmp     -44 <_proc+0x16>
100000f12:  8b 45 f4          movl    -12(%rbp), %eax
100000f15:  5d                popq    %rbp
100000f16:  c3                retq
100000f17:  66 0f 1f 84 00 00 00 00 00        nopw    (%rax,%rax)
100000f20:  55                pushq   %rbp
100000f21:  48 89 e5          movq    %rsp, %rbp
100000f24:  48 83 ec 20       subq    $32, %rsp
100000f28:  c7 45 fc 00 00 00 00    movl    $0, -4(%rbp)
100000f2f:  bf 00 09 3d 00    movl    $4000000, %edi
100000f34:  e8 2b 00 00 00    callq   43 <dyld_stub_binder+0x100000f64>
100000f39:  48 89 45 f0       movq    %rax, -16(%rbp)
100000f3d:  48 8b 7d f0       movq    -16(%rbp), %rdi
100000f41:  e8 8a ff ff ff    callq   -118 <_proc>
100000f46:  48 8d 3d 47 00 00 00    leaq    71(%rip), %rdi
100000f4d:  89 c6             movl    %eax, %esi
100000f4f:  b0 00             movb    $0, %al
100000f51:  e8 14 00 00 00    callq   20 <dyld_stub_binder+0x100000f6a>
100000f56:  31 f6             xorl    %esi, %esi
100000f58:  89 45 ec          movl    %eax, -20(%rbp)
100000f5b:  89 f0             movl    %esi, %eax
100000f5d:  48 83 c4 20       addq    $32, %rsp
100000f61:  5d                popq    %rbp
100000f62:  c3                retq

_proc:
100000ed0:  55                pushq   %rbp
100000ed1:  48 89 e5          movq    %rsp, %rbp
100000ed4:  48 89 7d f8       movq    %rdi, -8(%rbp)
100000ed8:  c7 45 f4 00 00 00 00    movl    $0, -12(%rbp)
100000edf:  c7 45 f0 00 00 00 00    movl    $0, -16(%rbp)
100000ee6:  81 7d f0 40 42 0f 00    cmpl    $1000000, -16(%rbp)
100000eed:  0f 8d 1f 00 00 00       jge     31 <_proc+0x42>
100000ef3:  48 8b 45 f8       movq    -8(%rbp), %rax
100000ef7:  48 63 4d f0       movslq  -16(%rbp), %rcx
100000efb:  8b 14 88          movl    (%rax,%rcx,4), %edx
100000efe:  03 55 f4          addl    -12(%rbp), %edx
100000f01:  89 55 f4          movl    %edx, -12(%rbp)
100000f04:  8b 45 f0          movl    -16(%rbp), %eax
100000f07:  83 c0 01          addl    $1, %eax
100000f0a:  89 45 f0          movl    %eax, -16(%rbp)
100000f0d:  e9 d4 ff ff ff    jmp     -44 <_proc+0x16>
100000f12:  8b 45 f4          movl    -12(%rbp), %eax
100000f15:  5d                popq    %rbp
100000f16:  c3                retq
100000f17:  66 0f 1f 84 00 00 00 00 00        nopw    (%rax,%rax)
```

```
_main:
100000f20:  55        pushq   %rbp
100000f21:  48 89 e5          movq    %rsp, %rbp
100000f24:  48 83 ec 20       subq    $32, %rsp
100000f28:  c7 45 fc 00 00 00 00      movl    $0, -4(%rbp)
100000f2f:  bf 00 09 3d 00    movl    $4000000, %edi
100000f34:  e8 2b 00 00 00    callq   43 <dyld_stub_binder+0x100000f64>
100000f39:  48 89 45 f0       movq    %rax, -16(%rbp)
100000f3d:  48 8b 7d f0       movq    -16(%rbp), %rdi
100000f41:  e8 8a ff ff ff    callq   -118 <_proc>
100000f46:  48 8d 3d 47 00 00 00      leaq    71(%rip), %rdi
100000f4d:  89 c6     movl    %eax, %esi
100000f4f:  b0 00     movb    $0, %al
100000f51:  e8 14 00 00 00    callq   20 <dyld_stub_binder+0x100000f6a>
100000f56:  31 f6     xorl    %esi, %esi
100000f58:  89 45 ec          movl    %eax, -20(%rbp)
100000f5b:  89 f0     movl    %esi, %eax
100000f5d:  48 83 c4 20       addq    $32, %rsp
100000f61:  5d        popq    %rbp
100000f62:  c3        retq
Disassembly of section __TEXT,__stubs:
__stubs:
100000f64:  ff 25 96 10 00 00         jmpq    *4246(%rip)
100000f6a:  ff 25 98 10 00 00         jmpq    *4248(%rip)
Disassembly of section __TEXT,__stub_helper:
__stub_helper:
100000f70:  4c 8d 1d 99 10 00 00      leaq    4249(%rip), %r11
100000f77:  41 53     pushq   %r11
100000f79:  ff 25 81 00 00 00         jmpq    *129(%rip)
100000f7f:  90        nop
100000f80:  68 00 00 00 00    pushq   $0
100000f85:  e9 e6 ff ff ff    jmp     -26 <__stub_helper>
100000f8a:  68 0e 00 00 00    pushq   $14
100000f8f:  e9 dc ff ff ff    jmp     -36 <__stub_helper>
```

6 Optimized Assembly Code:

```
./opt3:     file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,__text:
__text:
100000de0:  55        pushq   %rbp
100000de1:  48 89 e5          movq    %rsp, %rbp
100000de4:  66 0f ef c0       pxor    %xmm0, %xmm0
100000de8:  b8 24 00 00 00    movl    $36, %eax
100000ded:  66 0f ef c9       pxor    %xmm1, %xmm1
100000df1:  66 2e 0f 1f 84 00 00 00 00 00     nopw    %cs:(%rax,%rax)
```

```
100000dfb:  0f 1f 44 00 00   nopl    (%rax,%rax)
100000e00:  f3 0f 6f 94 87 70 ff ff ff       movdqu  -144(%rdi,%rax,4), %xmm
100000e09:  66 0f fe d0      paddd   %xmm0, %xmm2
100000e0d:  f3 0f 6f 44 87 80        movdqu  -128(%rdi,%rax,4), %xmm0
100000e13:  66 0f fe c1      paddd   %xmm1, %xmm0
100000e17:  f3 0f 6f 4c 87 90        movdqu  -112(%rdi,%rax,4), %xmm1
100000e1d:  f3 0f 6f 5c 87 a0        movdqu  -96(%rdi,%rax,4), %xmm3
100000e23:  f3 0f 6f 64 87 b0        movdqu  -80(%rdi,%rax,4), %xmm4
100000e29:  66 0f fe e1      paddd   %xmm1, %xmm4
100000e2d:  66 0f fe e2      paddd   %xmm2, %xmm4
100000e31:  f3 0f 6f 54 87 c0        movdqu  -64(%rdi,%rax,4), %xmm2
100000e37:  66 0f fe d3      paddd   %xmm3, %xmm2
100000e3b:  66 0f fe d0      paddd   %xmm0, %xmm2
100000e3f:  f3 0f 6f 4c 87 d0        movdqu  -48(%rdi,%rax,4), %xmm1
100000e45:  f3 0f 6f 5c 87 e0        movdqu  -32(%rdi,%rax,4), %xmm3
100000e4b:  f3 0f 6f 44 87 f0        movdqu  -16(%rdi,%rax,4), %xmm0
100000e51:  66 0f fe c1      paddd   %xmm1, %xmm0
100000e55:  66 0f fe c4      paddd   %xmm4, %xmm0
100000e59:  f3 0f 6f 0c 87   movdqu  (%rdi,%rax,4), %xmm1
100000e5e:  66 0f fe cb      paddd   %xmm3, %xmm1
100000e62:  66 0f fe ca      paddd   %xmm2, %xmm1
100000e66:  48 83 c0 28      addq    $40, %rax
100000e6a:  48 3d 64 42 0f 00        cmpq    $1000036, %rax
100000e70:  75 8e    jne     -114 <_proc+0x20>
100000e72:  66 0f fe c8      paddd   %xmm0, %xmm1
100000e76:  66 0f 70 c1 4e   pshufd  $78, %xmm1, %xmm0
100000e7b:  66 0f fe c1      paddd   %xmm1, %xmm0
100000e7f:  66 0f 70 c8 e5   pshufd  $229, %xmm0, %xmm1
100000e84:  66 0f fe c8      paddd   %xmm0, %xmm1
100000e88:  66 0f 7e c8      movd    %xmm1, %eax
100000e8c:  5d       popq    %rbp
100000e8d:  c3       retq
100000e8e:  66 90    nop
100000e90:  55       pushq   %rbp
100000e91:  48 89 e5         movq    %rsp, %rbp
100000e94:  bf 00 09 3d 00   movl    $4000000, %edi
100000e99:  e8 c4 00 00 00   callq   196 <dyld_stub_binder+0x100000f62>
100000e9e:  66 0f ef c0      pxor    %xmm0, %xmm0
100000ea2:  b9 1d 00 00 00   movl    $29, %ecx
100000ea7:  66 0f ef c9      pxor    %xmm1, %xmm1
100000eab:  eb 1a    jmp     26 <_main+0x37>
100000ead:  0f 1f 00         nopl    (%rax)
100000eb0:  f3 0f 6f 44 88 f0        movdqu  -16(%rax,%rcx,4), %xmm0
100000eb6:  f3 0f 6f 0c 88   movdqu  (%rax,%rcx,4), %xmm1
100000ebb:  66 0f fe c2      paddd   %xmm2, %xmm0
100000ebf:  66 0f fe cb      paddd   %xmm3, %xmm1
100000ec3:  48 83 c1 20      addq    $32, %rcx
100000ec7:  f3 0f 6f 5c 88 90        movdqu  -112(%rax,%rcx,4), %xmm3
100000ecd:  66 0f fe d8      paddd   %xmm0, %xmm3
100000ed1:  f3 0f 6f 44 88 a0        movdqu  -96(%rax,%rcx,4), %xmm0
```

```
100000ed7:  66 0f fe c1       paddd   %xmm1, %xmm0
100000edb:  f3 0f 6f 4c 88 b0 movdqu  -80(%rax,%rcx,4), %xmm1
100000ee1:  f3 0f 6f 64 88 c0 movdqu  -64(%rax,%rcx,4), %xmm4
100000ee7:  f3 0f 6f 54 88 d0 movdqu  -48(%rax,%rcx,4), %xmm2
100000eed:  66 0f fe d1       paddd   %xmm1, %xmm2
100000ef1:  66 0f fe d3       paddd   %xmm3, %xmm2
100000ef5:  f3 0f 6f 5c 88 e0 movdqu  -32(%rax,%rcx,4), %xmm3
100000efb:  66 0f fe dc       paddd   %xmm4, %xmm3
100000eff:  66 0f fe d8       paddd   %xmm0, %xmm3
100000f03:  48 81 f9 3d 42 0f 00 cmpq    $999997, %rcx
100000f0a:  75 a4    jne      -92 <_main+0x20>
100000f0c:  66 0f fe da       paddd   %xmm2, %xmm3
100000f10:  66 0f 70 c3 4e    pshufd  $78, %xmm3, %xmm0
100000f15:  66 0f fe c3       paddd   %xmm3, %xmm0
100000f19:  66 0f 70 c8 e5    pshufd  $229, %xmm0, %xmm1
100000f1e:  66 0f fe c8       paddd   %xmm0, %xmm1
100000f22:  66 0f 7e ce       movd    %xmm1, %esi
100000f26:  03 b0 e4 08 3d 00 addl    3999972(%rax), %esi
100000f2c:  03 b0 e8 08 3d 00 addl    3999976(%rax), %esi
100000f32:  03 b0 ec 08 3d 00 addl    3999980(%rax), %esi
100000f38:  03 b0 f0 08 3d 00 addl    3999984(%rax), %esi
100000f3e:  03 b0 f4 08 3d 00 addl    3999988(%rax), %esi
100000f44:  03 b0 f8 08 3d 00 addl    3999992(%rax), %esi
100000f4a:  03 b0 fc 08 3d 00 addl    3999996(%rax), %esi
100000f50:  48 8d 3d 3d 00 00 00 leaq    61(%rip), %rdi
100000f57:  31 c0    xorl    %eax, %eax
100000f59:  e8 0a 00 00 00    callq   10 <dyld_stub_binder+0x100000f68>
100000f5e:  31 c0    xorl    %eax, %eax
100000f60:  5d       popq    %rbp
100000f61:  c3       retq

_proc:
100000de0:  55       pushq   %rbp
100000de1:  48 89 e5          movq    %rsp, %rbp
100000de4:  66 0f ef c0       pxor    %xmm0, %xmm0
100000de8:  b8 24 00 00 00    movl    $36, %eax
100000ded:  66 0f ef c9       pxor    %xmm1, %xmm1
100000df1:  66 2e 0f 1f 84 00 00 00 00 00    nopw    %cs:(%rax,%rax)
100000dfb:  0f 1f 44 00 00    nopl    (%rax,%rax)
100000e00:  f3 0f 6f 94 87 70 ff ff ff       movdqu  -144(%rdi,%rax,4), %xmm
100000e09:  66 0f fe d0       paddd   %xmm0, %xmm2
100000e0d:  f3 0f 6f 44 87 80 movdqu  -128(%rdi,%rax,4), %xmm0
100000e13:  66 0f fe c1       paddd   %xmm1, %xmm0
100000e17:  f3 0f 6f 4c 87 90 movdqu  -112(%rdi,%rax,4), %xmm1
100000e1d:  f3 0f 6f 5c 87 a0 movdqu  -96(%rdi,%rax,4), %xmm3
100000e23:  f3 0f 6f 64 87 b0 movdqu  -80(%rdi,%rax,4), %xmm4
100000e29:  66 0f fe e1       paddd   %xmm1, %xmm4
100000e2d:  66 0f fe e2       paddd   %xmm2, %xmm4
100000e31:  f3 0f 6f 54 87 c0 movdqu  -64(%rdi,%rax,4), %xmm2
100000e37:  66 0f fe d3       paddd   %xmm3, %xmm2
```

```
100000e3b:  66 0f fe d0       paddd   %xmm0, %xmm2
100000e3f:  f3 0f 6f 4c 87 d0 movdqu  -48(%rdi,%rax,4), %xmm1
100000e45:  f3 0f 6f 5c 87 e0 movdqu  -32(%rdi,%rax,4), %xmm3
100000e4b:  f3 0f 6f 44 87 f0 movdqu  -16(%rdi,%rax,4), %xmm0
100000e51:  66 0f fe c1       paddd   %xmm1, %xmm0
100000e55:  66 0f fe c4       paddd   %xmm4, %xmm0
100000e59:  f3 0f 6f 0c 87    movdqu  (%rdi,%rax,4), %xmm1
100000e5e:  66 0f fe cb       paddd   %xmm3, %xmm1
100000e62:  66 0f fe ca       paddd   %xmm2, %xmm1
100000e66:  48 83 c0 28       addq    $40, %rax
100000e6a:  48 3d 64 42 0f 00 cmpq    $1000036, %rax
100000e70:  75 8e     jne     -114 <_proc+0x20>
100000e72:  66 0f fe c8       paddd   %xmm0, %xmm1
100000e76:  66 0f 70 c1 4e    pshufd  $78, %xmm1, %xmm0
100000e7b:  66 0f fe c1       paddd   %xmm1, %xmm0
100000e7f:  66 0f 70 c8 e5    pshufd  $229, %xmm0, %xmm1
100000e84:  66 0f fe c8       paddd   %xmm0, %xmm1
100000e88:  66 0f 7e c8       movd    %xmm1, %eax
100000e8c:  5d        popq    %rbp
100000e8d:  c3        retq
100000e8e:  66 90     nop

_main:
100000e90:  55        pushq   %rbp
100000e91:  48 89 e5          movq    %rsp, %rbp
100000e94:  bf 00 09 3d 00    movl    $4000000, %edi
100000e99:  e8 c4 00 00 00    callq   196 <dyld_stub_binder+0x100000f62>
100000e9e:  66 0f ef c0       pxor    %xmm0, %xmm0
100000ea2:  b9 1d 00 00 00    movl    $29, %ecx
100000ea7:  66 0f ef c9       pxor    %xmm1, %xmm1
100000eab:  eb 1a     jmp     26 <_main+0x37>
100000ead:  0f 1f 00          nopl    (%rax)
100000eb0:  f3 0f 6f 44 88 f0 movdqu  -16(%rax,%rcx,4), %xmm0
100000eb6:  f3 0f 6f 0c 88    movdqu  (%rax,%rcx,4), %xmm1
100000ebb:  66 0f fe c2       paddd   %xmm2, %xmm0
100000ebf:  66 0f fe cb       paddd   %xmm3, %xmm1
100000ec3:  48 83 c1 20       addq    $32, %rcx
100000ec7:  f3 0f 6f 5c 88 90 movdqu  -112(%rax,%rcx,4), %xmm3
100000ecd:  66 0f fe d8       paddd   %xmm0, %xmm3
100000ed1:  f3 0f 6f 44 88 a0 movdqu  -96(%rax,%rcx,4), %xmm0
100000ed7:  66 0f fe c1       paddd   %xmm1, %xmm0
100000edb:  f3 0f 6f 4c 88 b0 movdqu  -80(%rax,%rcx,4), %xmm1
100000ee1:  f3 0f 6f 64 88 c0 movdqu  -64(%rax,%rcx,4), %xmm4
100000ee7:  f3 0f 6f 54 88 d0 movdqu  -48(%rax,%rcx,4), %xmm2
100000eed:  66 0f fe d1       paddd   %xmm1, %xmm2
100000ef1:  66 0f fe d3       paddd   %xmm3, %xmm2
100000ef5:  f3 0f 6f 5c 88 e0 movdqu  -32(%rax,%rcx,4), %xmm3
100000efb:  66 0f fe dc       paddd   %xmm4, %xmm3
100000eff:  66 0f fe d8       paddd   %xmm0, %xmm3
100000f03:  48 81 f9 3d 42 0f 00 cmpq    $999997, %rcx
```

```
100000f0a:  75 a4     jne      -92 <_main+0x20>
100000f0c:  66 0f fe da        paddd    %xmm2, %xmm3
100000f10:  66 0f 70 c3 4e     pshufd   $78, %xmm3, %xmm0
100000f15:  66 0f fe c3        paddd    %xmm3, %xmm0
100000f19:  66 0f 70 c8 e5     pshufd   $229, %xmm0, %xmm1
100000f1e:  66 0f fe c8        paddd    %xmm0, %xmm1
100000f22:  66 0f 7e ce        movd     %xmm1, %esi
100000f26:  03 b0 e4 08 3d 00          addl     3999972(%rax), %esi
100000f2c:  03 b0 e8 08 3d 00          addl     3999976(%rax), %esi
100000f32:  03 b0 ec 08 3d 00          addl     3999980(%rax), %esi
100000f38:  03 b0 f0 08 3d 00          addl     3999984(%rax), %esi
100000f3e:  03 b0 f4 08 3d 00          addl     3999988(%rax), %esi
100000f44:  03 b0 f8 08 3d 00          addl     3999992(%rax), %esi
100000f4a:  03 b0 fc 08 3d 00          addl     3999996(%rax), %esi
100000f50:  48 8d 3d 3d 00 00 00       leaq     61(%rip), %rdi
100000f57:  31 c0     xorl     %eax, %eax
100000f59:  e8 0a 00 00 00     callq    10 <dyld_stub_binder+0x100000f68>
100000f5e:  31 c0     xorl     %eax, %eax
100000f60:  5d        popq     %rbp
100000f61:  c3        retq
Disassembly of section __TEXT,__stubs:
__stubs:
100000f62:  ff 25 98 10 00 00          jmpq     *4248(%rip)
100000f68:  ff 25 9a 10 00 00          jmpq     *4250(%rip)
Disassembly of section __TEXT,__stub_helper:
__stub_helper:
100000f70:  4c 8d 1d 99 10 00 00       leaq     4249(%rip), %r11
100000f77:  41 53     pushq    %r11
100000f79:  ff 25 81 00 00 00          jmpq     *129(%rip)
100000f7f:  90        nop
100000f80:  68 00 00 00 00     pushq    $0
100000f85:  e9 e6 ff ff ff     jmp      -26 <__stub_helper>
100000f8a:  68 0e 00 00 00     pushq    $14
100000f8f:  e9 dc ff ff ff     jmp      -36 <__stub_helper>
```

7 Test results

Before conducting the tests, we first calculated the estimated run times using the following formula:

$$\frac{I * CPI}{f}$$

Where I stands for the number of executed instructions, CPI is the average number of Clocks per Instruction, and f is our Clock Frequency. This will give us a time in millisecons.

For the CPIs, we weren't able to find a table that contained the specific CPIs for the Coffee Lake 14nm++ architecture from Intel, but since Coffee Lake is built on the same 14nm process as Kaby Lake, and Kaby Lake is built

on the same 14 nm process as Sylake, we took the table for the enthusiast-grade Skylake-X architecture and figured they would perform similarly given the 14 nm process situation and performance gains reported by reviewers and benchmarks available online.

Having said that, using the CPIs from Skylake-X for our calculations, we obtained an estimated time of 1.2195 ms for the execution of our non-optimized file, whereas for the optimized file, we obtained 4.3488 ms. This is to be expected, since the optimization process opts for a different approach than usual: instead of repeating the same instruction several times, we instead go for more instructions but each one is repeated less times. The estimated latency is not necessarily a definitive equivalent of what is real-world performance, since the CPI is based on averages, and different instructions take different times to execute, so we corroborate our results with the time command provided by bash:

```
time ./nonOpt
0.37 real         0.00 user         0.00 sys


time ./opt3
0.10 real         0.00 user         0.00 sys


time ./nonOpt
0.31 real         0.00 user         0.00 sys


time ./opt3
0.09 real         0.00 user         0.00 sys


time ./nonOpt
0.08 real         0.00 user         0.00 sys


time ./opt3
0.07 real         0.00 user         0.00 sys


time ./nonOpt
0.07 real         0.00 user         0.00 sys


time ./opt3
0.06 real         0.00 user         0.00 sys
```

We ran each one four times: two with from a cold start (that is after leaving the computer idle for some time at below-base speeds –30 minutes was our standard), and two with an already warm CPU, that is, running at the full 4.1 GHz Turbo Frequency that our processor allows us to use.

8 Conclusions

We can observe from our results that even though our estimates were a bit off, perhaps because we were not using the official CPIs for the Coffee Lake processors, but rather the CPI table for an enthusiast-grade lineup of CPUs, we can actually observe a difference between the optimized and unoptimized versions of our files.

From a cold start, the execution times differ vastly, with the unoptimized version taking anywhere from 0.30 to 0.40 seconds to execute, while the optimized version took only around 0.10 seconds. This is a performance gain of nearly 70%. However, when running on a warm CPU, things weren't as drastic as with a cold CPU; the unoptimized program ran only about 0.01 seconds slower than the optimized version, which is still a performance gain, but maybe not as much as we would expect. However, this was using a fairly simple function with a quite simple task. This 0.01 second difference could add up and translate to potentially huge gains when dealing with huge programs.

9 References:

Fog, A. (2019). Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. https://www.agner.org/optimize/instruction_tables.pdf