1.

a. 100
b. Since the addition can be pipelined, at steady state, a functional unit can process 1 addition each cycle. And since there are 4 functional units, the peak performance is 4 additions per cycle.
c. Even though the addition can be pipelined and there are a total of 4 functional units, since we are performing a dependent chain of computation, we can't really take advantage of them. As the result, it will take 200 cycles to complete the sum.
d. The time it takes to compute the vector sum was calculated to be 200 cycles. Since the peak performance was calculated to be 4 additions per cycle, theoretically 800 independent float point additions can be performed. The code above only has a efficiency of 0.5 addition per cycle, so it only has 12.5% of the peak performance.
e. Since the latency is 2 cycle, 2 chains of independent addition can fill in the pipeline. And since we have 4 functional units, we need a total of 8 independent addtions.

2.

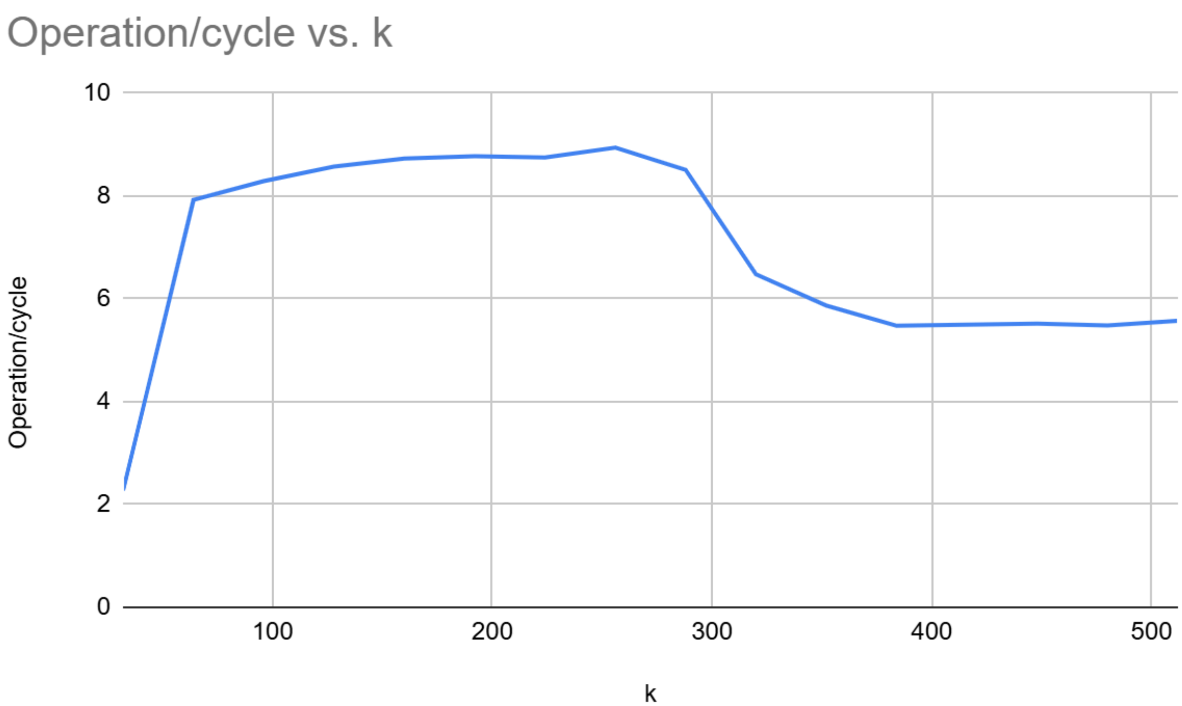
a. Since we can only fire one instruction at a cycle, there are a total of 2 operations, and each operation takes one cycle, it takes 2 cycles to complete one max. Therefore 0.5 max per cycle.
b. Since the functional unit is pipelined, we can fill the pipeline with 2 independent chain of instructions. Then, we can perform 2 max in 4 cycles, with is still 0.5 max per cycle.
c. For each max operation, we need to perform 2 instructions: a GEQ and a BLEND. The GEQ instruction takes in 2 inputs and produces 1 output, the BLEND instructions takes in 3 inputs and produces 1 output, but the output of BLEND can safely overwrite the output of GEQ, so the 1 max operation uses 3 registers. Furthermore, since the code is always comparing each element in the in array with 0, the subsequent max only needs 1 input. With 16 registers, we can perform 7 max operations at the same time (uses 15 registers). But since we only have 1 functional unit, it only takes 2 independent chains of max operation to fill the pipeline. The pseudocode for the kernel is then:

```
For i between 0 and len:
        input_1 = in[i]
        input_2 = in[i + 1]
        mask_1 = GEQ(input_1, 0)
        mask_2 = GEQ(input_2, 0)
        out[i] = BLEND(input_1, 0, mask_1)
        out[i + 1] = BLEND(input_2, 0, mask_1)
```

This assumes the length of the in array is divisible by 2.

3.

a. If the FMA takes 6 cycles, then we need at least 6 independent chains of instructions to fill the pipeline. This means the matrix C must have at least 6 entries, so M x N could be either 3 x 2 or 6 x 1.
b. With 4 functional units, we need 24 chains of instructions to fill the pipeline. For a matrix C with 24 entries, it can have shape of 6 x 4, 8 x 3, 12 x 2, or 24 x 1.
c. With a SIMD FMA that has a latency of 5 cycles and throughput of 2 instructions per cycle, we need at least a total of 10 independent instructions to fill the pipeline. The size of C however, depends on how many inputs 1 FMA can take. For a FMA that takes 4 inputs, C must have at least 40 entries. For the simplicity of SIMD instructions, we can let C have the shape 10x4.



4. Around 5 hours.