Lab 7 Daniel Hjelm:

Task 1:

Since N is 200, we can try a unroll factor of 4, since 200/4 = 50.

When we did this it actually became slower, let’s try 8.

With 8 it’s still slower.

But with 20 it’s faster!  
  
So with a unroll factor of 20 it was worth it.

Let’s see if the compiler can do a better job at unrolling.

Without any flag: 2.660 wall seconds.

With O2: 0.248 wall seconds.

With -funroll-loops: 0.249 wall seconds.

With O3: 0.259 wall seconds.

So unrolling makes the code faster, but not by much.

Task -2:

Here we can skip the variable counter and the if-condition if we use unroll with a unroll factor of 4. This reduced the time by half! 0.580 🡪 0.245

If we try to let the compiler do it for us (-funroll-loops): 0.6 , so it became slower.

This is probably since it can’t really unroll the if-statement.

Task-3:

When using loop fusion we get the following results:

Doing 400 iterations with N=232 using apply\_stencil function version 1 took 0.770 wall seconds.

Doing 400 iterations with N=232 using apply\_stencil function version 2 took 0.446 wall seconds.

Doing 400 iterations with N=232 using apply\_stencil function version 3 took 0.771 wall seconds.

Checking correctness by comparing 1 and 2: max\_abs\_diff = 2.442e-15

Checking correctness by comparing 1 and 3: max\_abs\_diff = 0

OK, results seem correct!

So it becomes almost twice as fast when using loop fusion.

When we change Stencil\_sz from 20 to 4 there is almost no change in speed:

Doing 400 iterations with N=200 using apply\_stencil function version 1 took 0.077 wall seconds.

Doing 400 iterations with N=200 using apply\_stencil function version 2 took 0.066 wall seconds.

Doing 400 iterations with N=200 using apply\_stencil function version 3 took 0.087 wall seconds.

Checking correctness by comparing 1 and 2: max\_abs\_diff = 8.882e-16

Checking correctness by comparing 1 and 3: max\_abs\_diff = 0

OK, results seem correct!

There almost no different between the loop fused and the original.

Task-4:

We should really try to avoid using if-statements inside loops since they can really slow down the program.

We were able to reduce the code so we did not have any if statements and no variable counter. Results:

f\_std tests took 0.923 wall seconds.

f\_opt tests took 0.231 wall seconds.

Checking correctness: abs\_diff = 0

OK, result seems correct.

Task-5:

Without any change:

f\_std tests took 1.327 wall seconds.

f\_opt tests took 1.115 wall seconds.

If we change 1000 to 500 to get distribution up to 0.5 instead of 1 we get:

f\_std tests took 0.412 wall seconds.

f\_opt tests took 0.327 wall seconds.

This is a lot faster, so branch-prediction is good when the condition is always (or almost always meet/non-meet. But branch-prediction is not as fast as removing the branches. This we see since in Task-4 we could reduce the time to 0.231 which is faster than 0.327

Task-6:

Kompilering: gcc -O2 -g -Wall -ftree-vectorize -o matvec matvec.c:

Ref: time = 1.237129

Auto-vec: time = 0.345224

e\_sum: 0.000000

OK

Kompilering: gcc -O2 -g -Wall -ftree-vectorize -ffast-math -o matvec matvec.c

Ref: time = 0.931815

Auto-vec: time = 0.168403

e\_sum: 0.000000

OK

Kompilering: gcc -O2 -g -Wall -ftree-vectorize -ffast-math -o -march=native matvec matvec.c

Ref: time = 0.171669

Auto-vec: time = 0.156568

e\_sum: 0.000000

OK

Task-7: