Problem 4

I chose an algorithm to find the rank of a matrix (https://en.wikipedia.org/wiki/Rank_(linear_algebra)). The algorithm is a relatively basic one based on the commonplace Guass-Jordan elimination method. More about it can be read here: https://cp-algorithms.com/linear_algebra/rank-matrix.html (this link also contains a C++ implementation, and I effectively rewrote their implementation in C).

Part 1:

Data for the optimization levels + input matrix size vs. execution time was taken, and results can be seen in the chart below. Unsurprisingly, the program runtime scaled with the matrix size, and it wasn't until the matrix hit 5000 x 5000 that time differences became significant. The program was compiled and executed on an M1 Pro Max Macbook Pro with 32 GB of RAM.

It's worth noting that even though there was a large difference between the unoptimized program and the -O1 optimized program, there wasn't a large difference for subsequent optimizations. The most notable difference would be how the -Oz was noticeably slower than the O1, O2, Os, O3, and Ofast optimizations. This makes me think that the vast majority of the major optimizations that could be made were included in -O1 optimization.

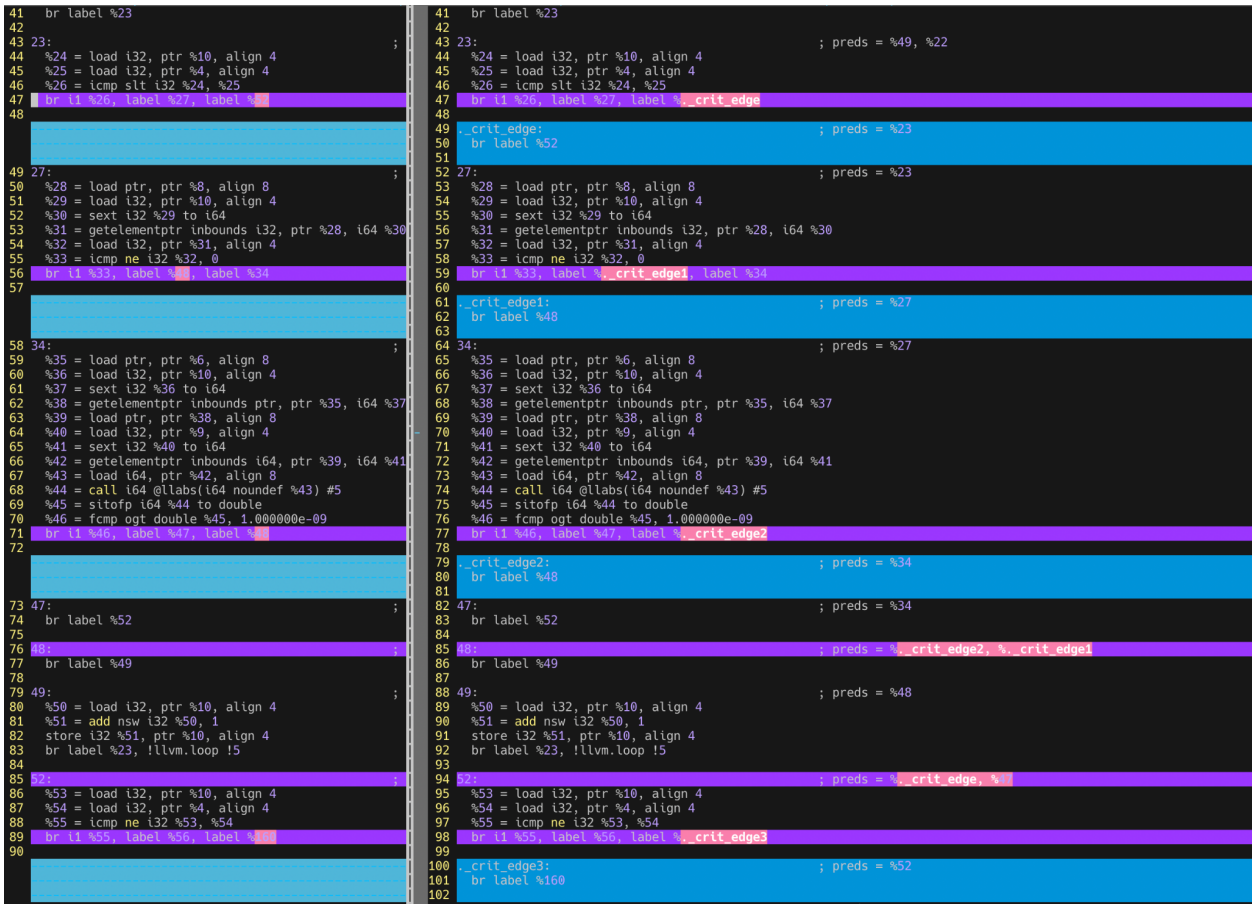
In terms of program size, there does not seem to be a big difference regardless of the optimization. The largest program was actually the -Ofast one, and the smallest one was tied between the O1, O2, and Os programs. One thing that I find interesting is that the -Oz compiled program was actually larger than the Os program. This is surprising because, from my understanding, the -Oz optimization attempts to optimize size more aggressively than the -Os optimization. I think this might be because the program is so small in relation to a real program that would actually benefit from size optimization.

Time	No optimizations (O0)	O1	O2	Os	Oz	O3	Ofast
10 x 10	0.387 sec	0.089 sec	0.238 sec	0.100 sec	0.450 sec	0.347 sec	0.269 sec
50 x 50	0.130 sec	0.103 sec	0.231 sec	0.107 sec	0.247 sec	0.095 sec	0.240 sec
500 x 500	0.386 sec	0.266 sec	0.129 sec	0.229 sec	0.142 sec	0.244 sec	0.130 sec
5000 x 5000	2 min 17.245 sec	46.277 sec	46.037 sec	46.265 sec	52.741 sec	46.411 sec	46.036 sec
10000 x 10000	17 min 39.286 sec	5 min 32.531 sec	5 min 31.481 sec	5 min 32.281 sec	6 min 35.099 sec	5 min 33.932 sec	5 min 33.565 sec

Optimization	Program size
No optimizations (O0)	33604 bytes
O1	33569 bytes
O2	33569 bytes
Os	33569 bytes
Oz	33601 bytes
O3	33633 bytes
Ofast	33636 bytes

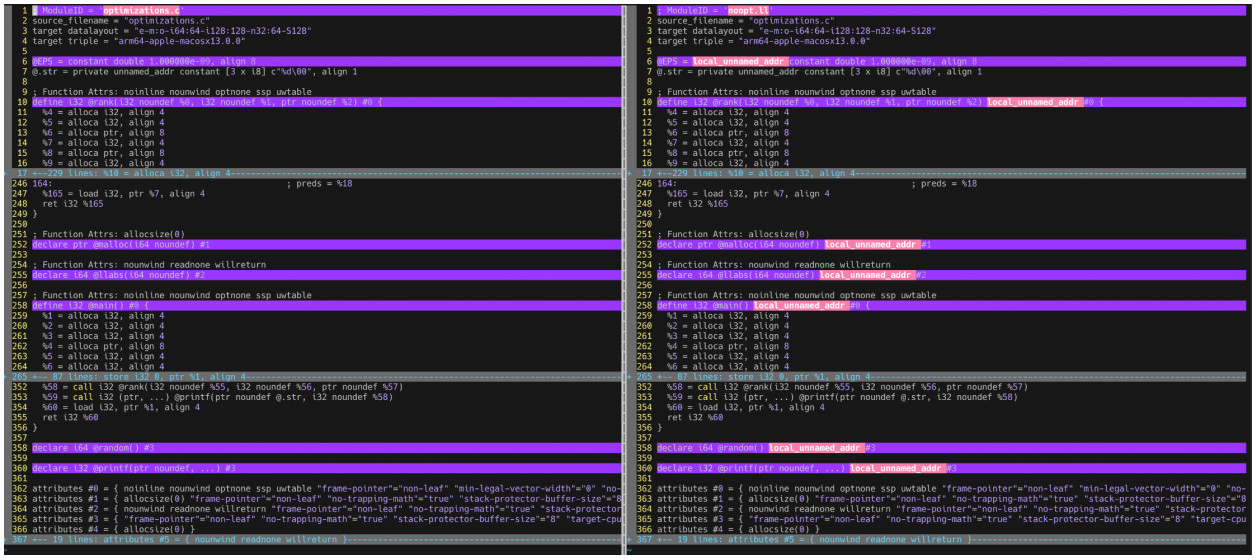
Part 2:

1. --type-promotion + --break-crit-edges made an effect. Examining the difference between the unoptimized and optimized code with vimdiff, it seems like the majority of the differences seem to be these "._crit_edgeX" labels, as seen below in the screenshot. Interestingly, these crit_edge labels don't appear wit just the --break-crit-edges optimization flag used. Both the --type-promotion and --break-crit-edge flags need to be set.



2. --global-opt also made an effect. This time it seems like it affected function definitions. From the vimdiff output, it seems like the tag "local_unnamed_addr" has been added

before the "#X" portion.



3. --strip had a minor effect. It seems like a lot of the "@.str"s were simply replaced with "@0".