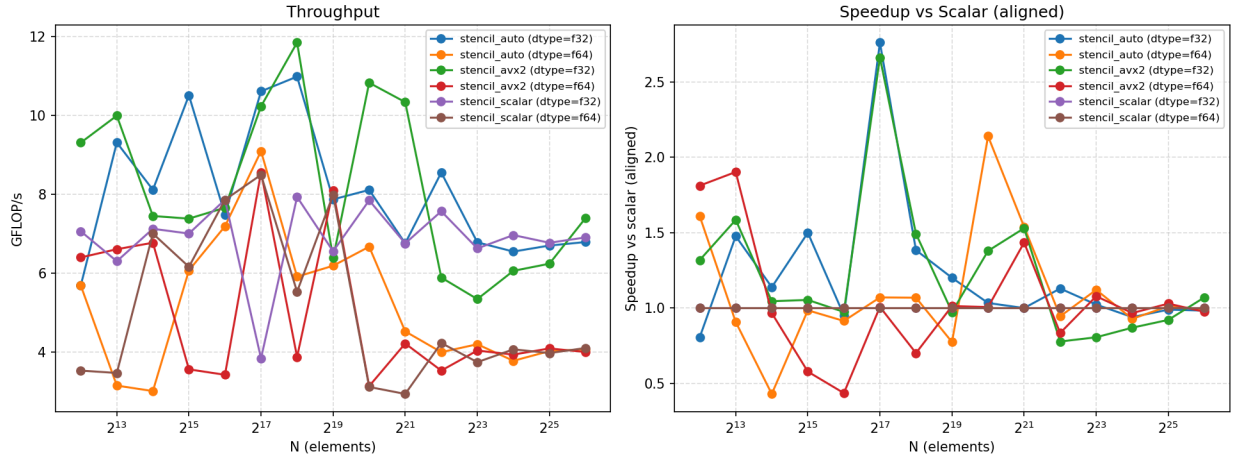
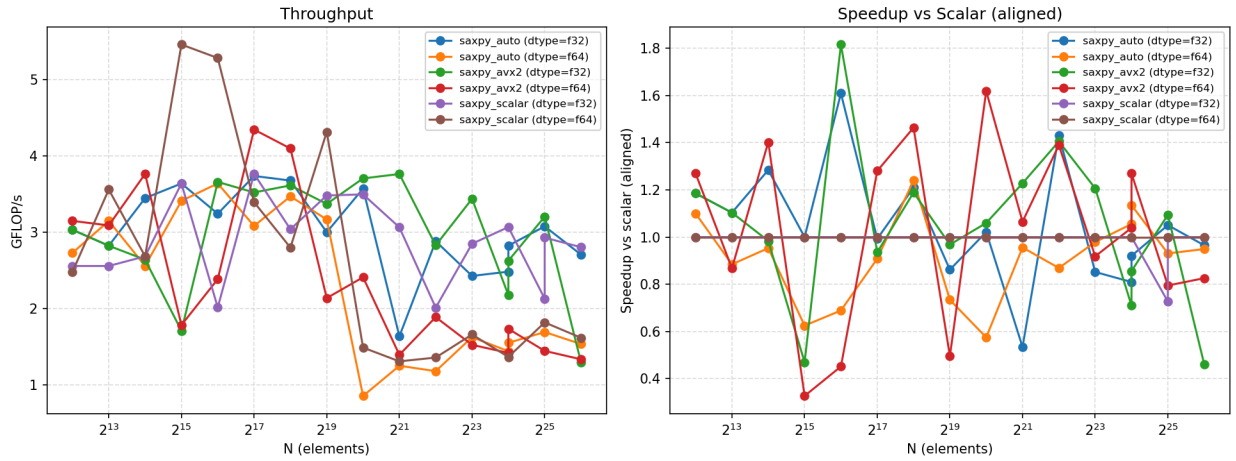
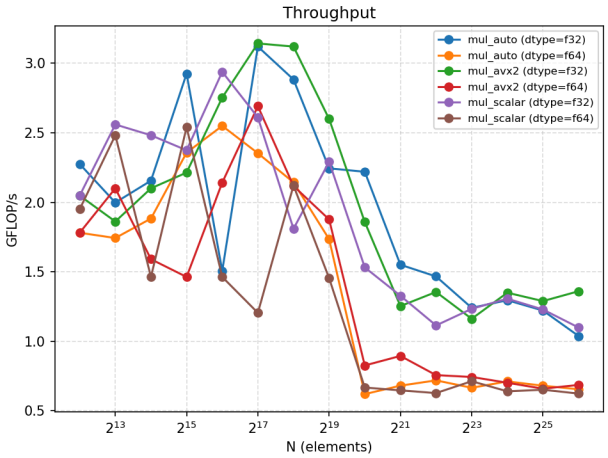
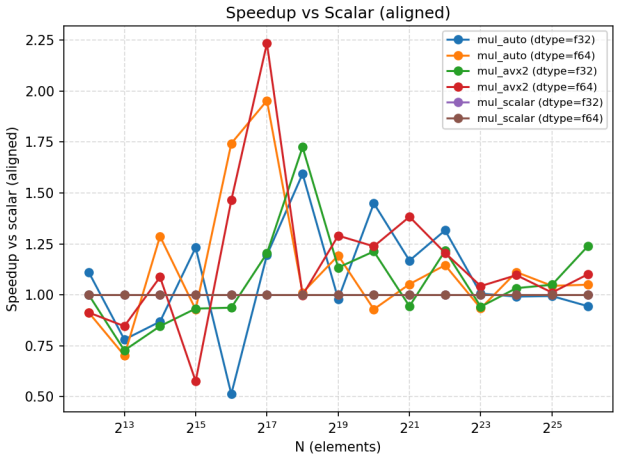
1. Baseline

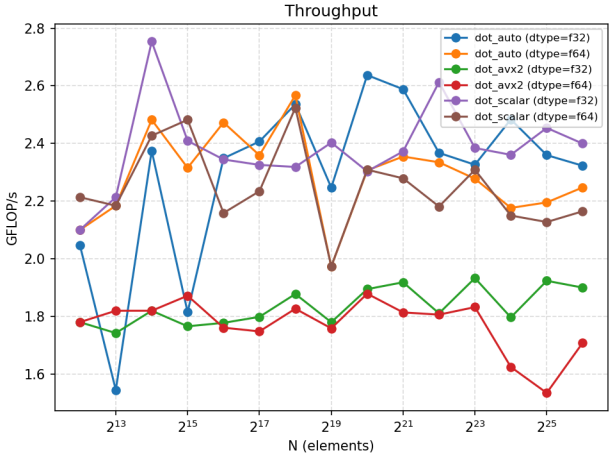
The scalar and vectorized versions for each kernel were ran across numerous sizes with the throughput in GFLOPs and the speedup comparison plotted below. The general trends seem to show a decrease in throughput as the number of elements increases as well as a peak speedup around the middle of the set for certain configurations, with a lot of variability between each configuration.

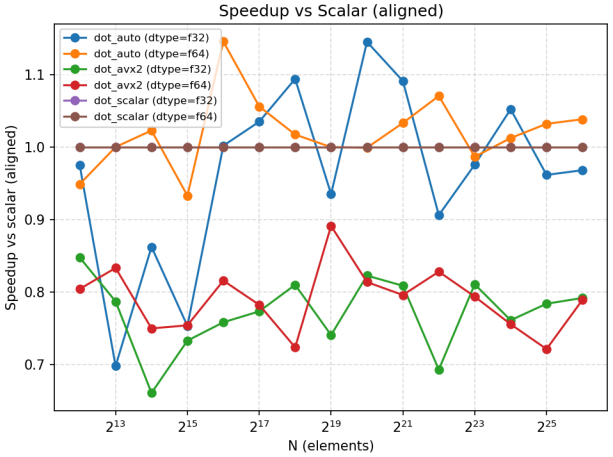








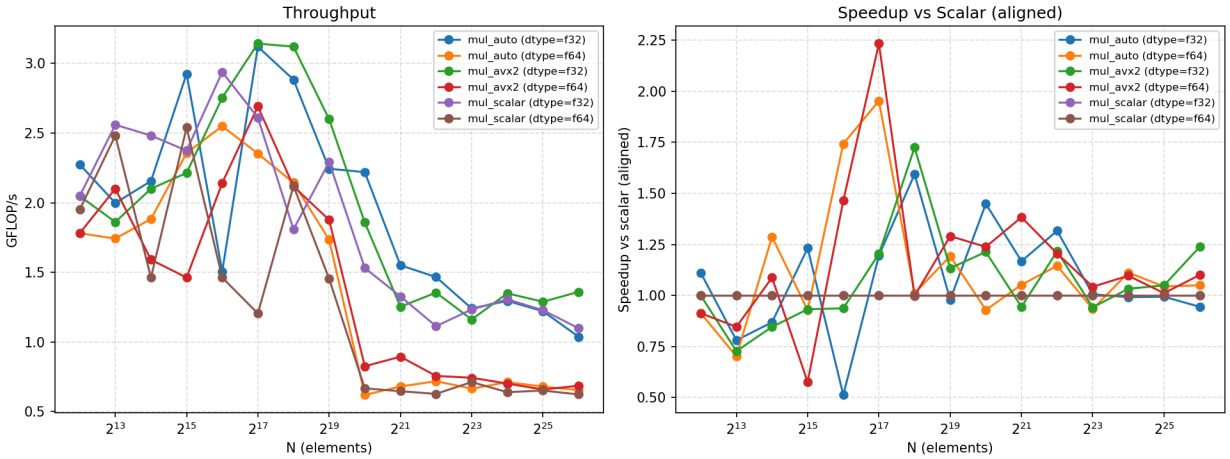




2. Vectorization verification

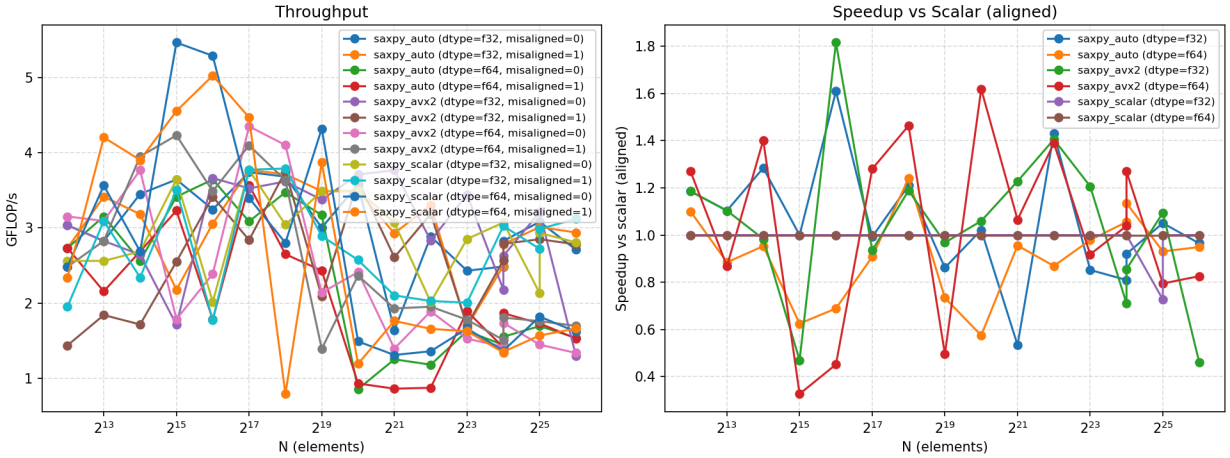
3. Locality sweep

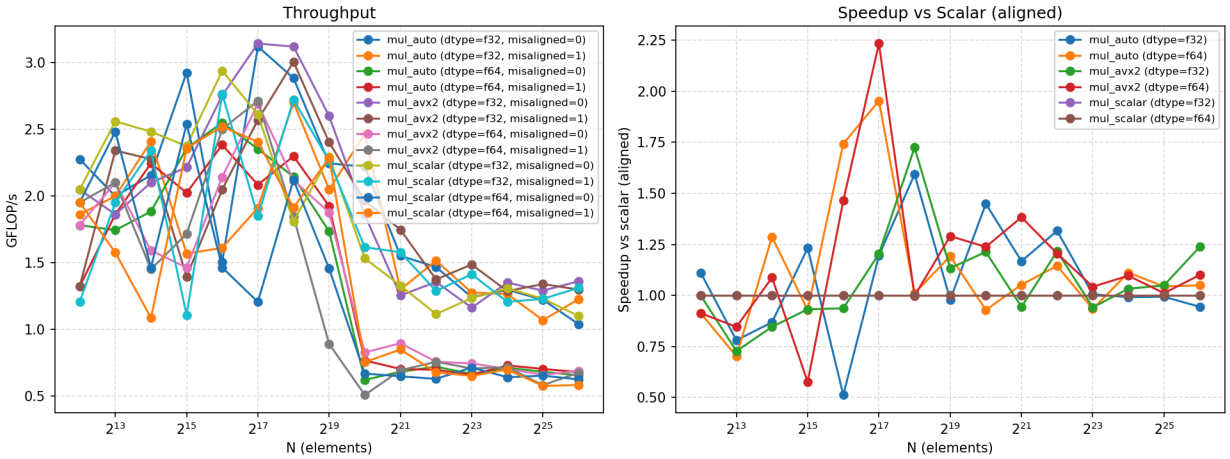
Taking a closer look at elementwise multiply, we can see the strongest decrease in throughput after around 2^20 elements. Additionally, speedup starts off slower than default and increases to its max around 2^17 elements. It then settles back towards the default, but remains at a slight speedup or at least on par with the default.

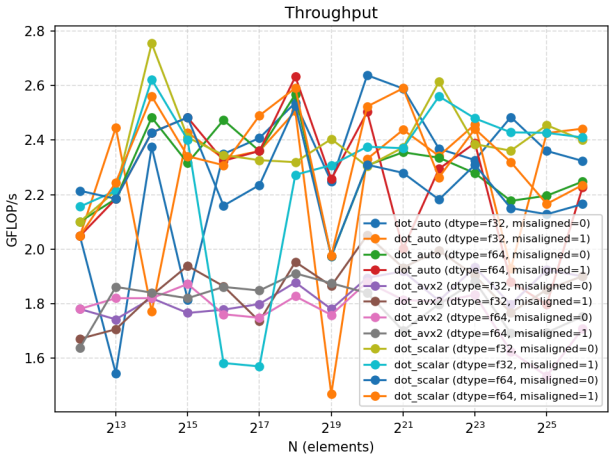


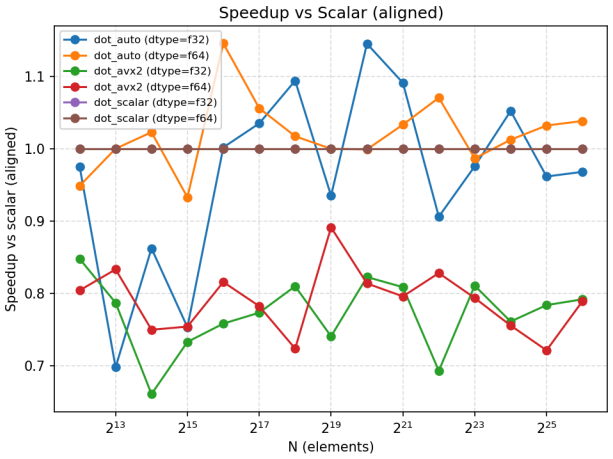
4. Alignment & tail-handling

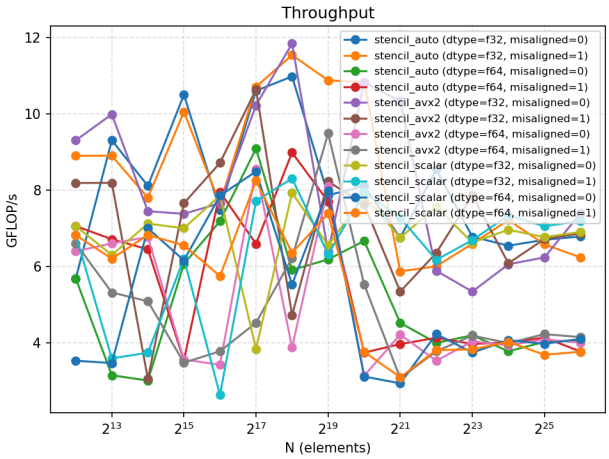
Below we can see comparisons between aligned and misaligned data. When comparing similar cases and looking solely at the alignment, we can see overall decreases in GFLOP/s levels.

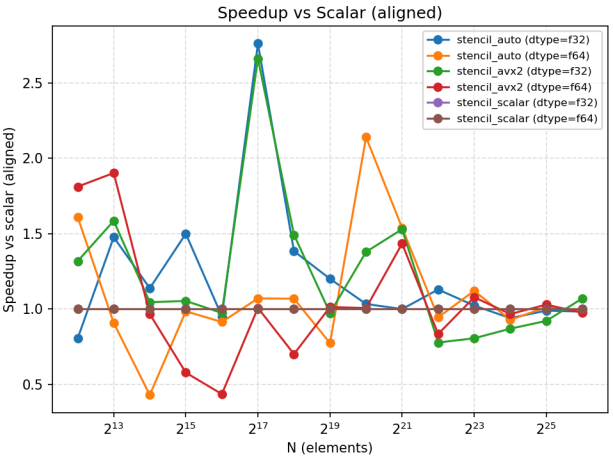





Throughput
GFLOP/s
N (elements)
dot_auto (dtype=f32, misaligned=0)
dot_auto (dtype=f32, misaligned=1)
dot_auto (dtype=f64, misaligned=0)
dot_auto (dtype=f64, misaligned=1)
dot_avx2 (dtype=f32, misaligned=0)
dot_avx2 (dtype=f32, misaligned=1)
dot_avx2 (dtype=f64, misaligned=0)
dot_avx2 (dtype=f64, misaligned=1)
dot_scalar (dtype=f32, misaligned=0)
dot_scalar (dtype=f32, misaligned=1)
dot_scalar (dtype=f64, misaligned=0)
dot_scalar (dtype=f64, misaligned=1)

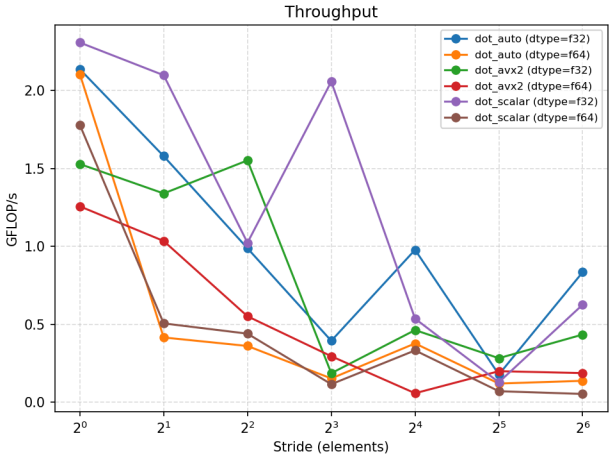

Speedup vs Scalar (aligned)
Speedup vs scalar (aligned)
N (elements)
dot_auto (dtype=f32)
dot_auto (dtype=f64)
dot_avx2 (dtype=f32)
dot_avx2 (dtype=f64)
dot_scalar (dtype=f32)
dot_scalar (dtype=f64)

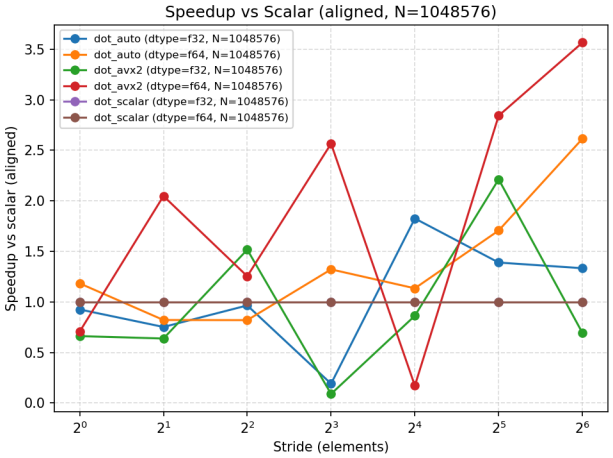

Throughput
GFLOP/s
N (elements)
stencil_auto (dtype=f32, misaligned=0)
stencil_auto (dtype=f32, misaligned=1)
stencil_auto (dtype=f64, misaligned=0)
stencil_auto (dtype=f64, misaligned=1)
stencil_avx2 (dtype=f32, misaligned=0)
stencil_avx2 (dtype=f32, misaligned=1)
stencil_avx2 (dtype=f64, misaligned=0)
stencil_avx2 (dtype=f64, misaligned=1)
stencil_scalar (dtype=f32, misaligned=0)
stencil_scalar (dtype=f32, misaligned=1)
stencil_scalar (dtype=f64, misaligned=0)
stencil_scalar (dtype=f64, misaligned=1)

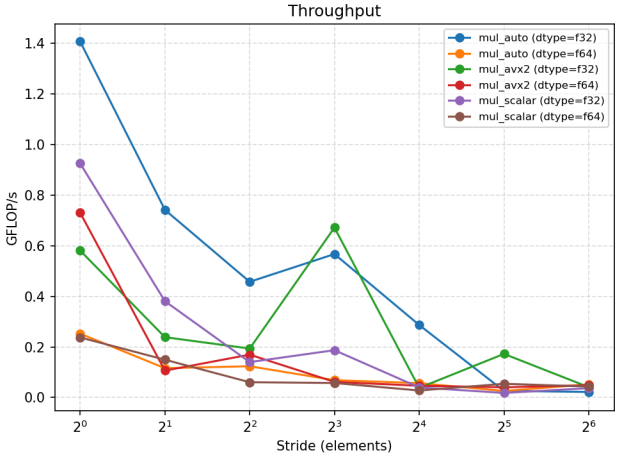

Speedup vs Scalar (aligned)
Speedup vs scalar (aligned)
N (elements)
stencil_auto (dtype=f32)
stencil_auto (dtype=f64)
stencil_avx2 (dtype=f32)
stencil_avx2 (dtype=f64)
stencil_scalar (dtype=f32)
stencil_scalar (dtype=f64)

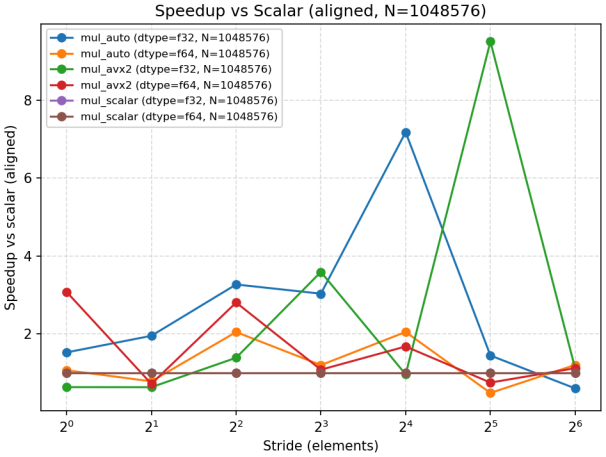
5. Stride / gather effects

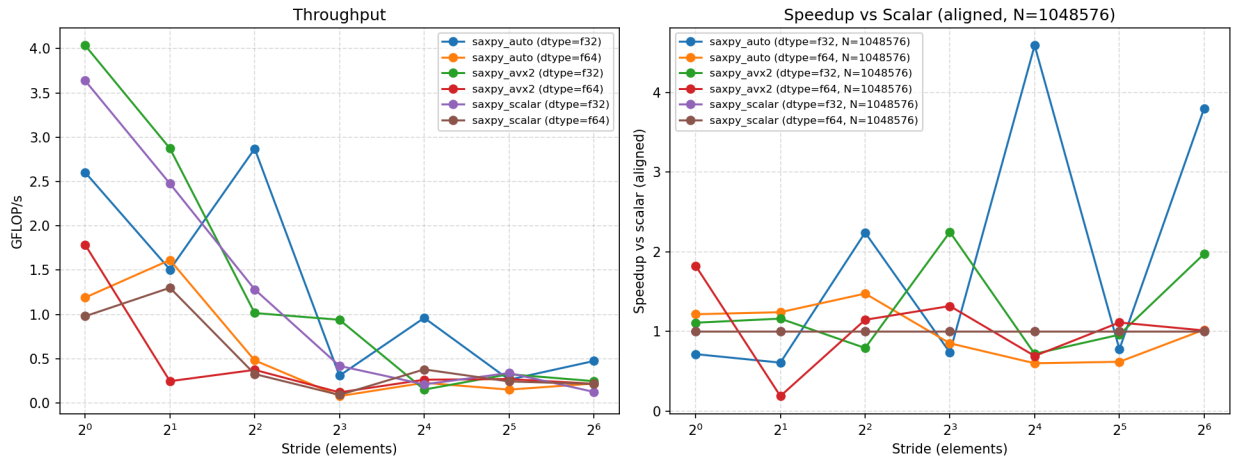
Looking at varied stride levels, we can see that in general, as stride increases, GFLOP/s decrease, but the speedup benefit vs the default grows for higher stride values.

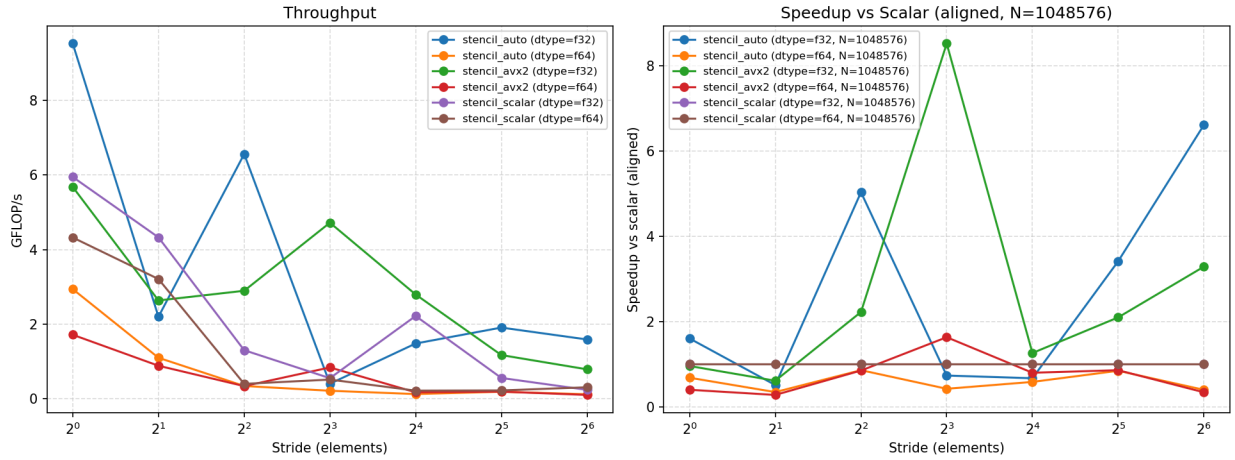






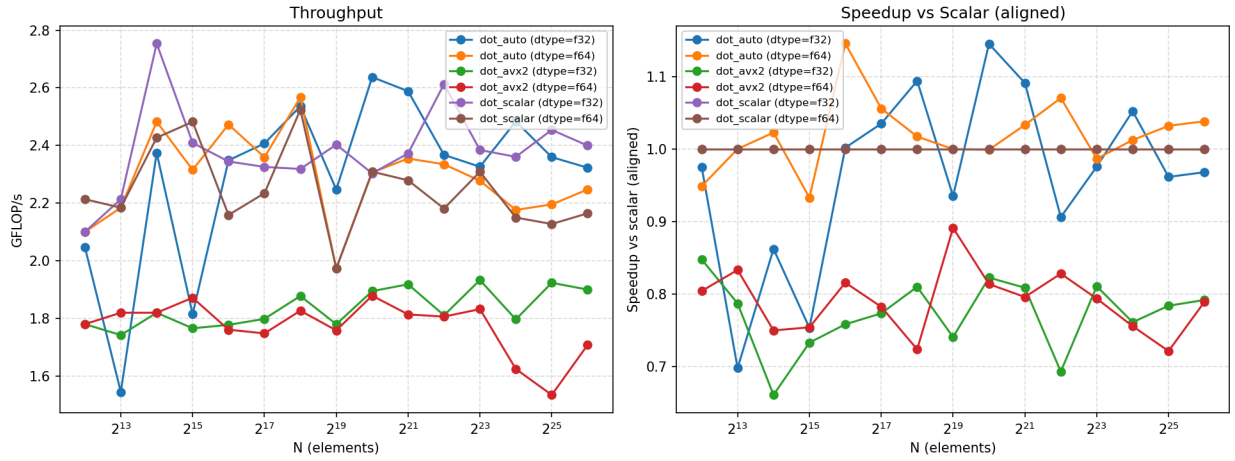






6. Data type comparison

Again, looking at a specific case (dot product), we can see that using f64 data results in lower throughput than the corresponding f32 data, however, the general speedups compared to the default (f64, scalar) shows mixed performance depending on the number of elements.



7. Roofline analysis

Looking at the roofline, we can see performance tends to generally track linearly, showing that this data is likely still in the bandwidth bound region. The exception to this may be for the dot calculations, which taper off as even though intensity is increasing, performance is at a plateau, indicating it may be compute bound.