| Cases | 4x4 | 8x8 | 16x16 | 32x32 |
|---|---|---|---|---|
| 1B | 1.710880 | 0.031840 | 0.029632 | 0.032864 |
| 2B | 1.694048 | 0.024160 | 0.022016 | 0.024000 |
| 3B | 0.012544 | 2.019872 | 0.030560 | 0.032000 |
| 4B | 0.009536 | 0.014816 | 2.088224 | 0.265408 |
| 1T | 0.066368 | 0.052704 | .05262 | .05743 |
| 2T | 0.061248 | 0.053472 | .05081 | .05852 |
| 3T | 0.071360 | 0.053664 | .05345 | .05781 |
| 4T | 0.254848 | 0.103104 | .09389 | .09604 |

All times in (ms)

Themes I noticed:

- 4x4 blocks are usually slower than 8x8, 16x16, or 32x32
- At some point having bigger block sizes doesn't help with speedup, since the added block width means there is a lot of "dead" space on the edges
- The basic one is pretty fast *for small matrices with edges that are easily divisible by the nxn tiles (less comparisons and FLOPS)
- The Basic one has a lot of outliers (>1.0 ms) that get larger as the file size increases. This is not ever seen in the Tiled computations
- Since 128 is divisible by all of those numbers, but 100 is not, there is no noticeable speedup between Case 1 and Case 2, except a small amount for the exited if statements (skipped rows and columns)
- Case 3 is not square but is slightly bigger than the first two cases, this means there is a lot of unused edges in the bigger blocks.
- As the square matrices are smaller than the rectangular matrices it's hard to directly compare them; however, it appears that they do not go significantly faster than the rectangular matrices because the ones marginally bigger than the square matrices are roughly the same time as the next ones below it (case 3–rectangular–to case 2–square)