Block Size and Performance:

The block size of 32 performed relatively poorly compared to the CPU implementation. The CPU implementation outperformed both the GPU ELL implementation and the GPU CSR implementation by a factor of around 2 when a block size of 32 was used. This was most likely caused by a number of different factors. A block size of 32 underutilizes the GPU cores which leads to poor performance. In addition with a block size of only 32 the kernel has a high kernel launch overhead relative to work done. A block size of 64 was the sweet spot for performance on both ELL and CSR kernels. When the block size was 64 the ELL and CSR kernels achieved speedups of 3.15x and 2.95x respectively.

Block sizes of 96 and 128 exhibited significantly degraded performance compared to the optimal block size of 64, with execution times regressing to approximately 0.50ms for CSR and 0.44ms for ELL representing a performance loss of roughly 7× relative to block size 64. This performance degradation can be attributed to several architectural limitations. Larger block sizes increase register pressure, as each thread requires a certain number of registers, and with more threads per block, the total register demand can exceed what is available per streaming multiprocessor.

GPU vs CPU Performance Comparison

Speedup Analysis (Block Size 64 - Best Performance)

Implementation	Time (s)	Speedup vs CPU CSR
CPU CSR	0.0224	1.00×
GPU CSR	0.0076	**2.95×**
GPU ELL	0.0071	**3.15×**

All Block Sizes vs CPU

Block Size	CSR Speedup	ELL Speedup
32	0.42×	0.43×
64	**2.95×**	**3.15×**
96	0.44×	0.50×
128	0.44×	0.50×

CSR vs ELL:

The ELL format consistently outperformed the CSR format across all block sizes tested, though the performance advantage was relatively modest compared to the dramatic differences observed between formats in CPU implementations. At the optimal block size of 64, ELL achieved a kernel execution time of 0.071ms compared to CSR's 0.076ms, a 6.6% performance improvement. This advantage stems from ELL's more regular memory access patterns, which enable better memory coalescing on GPU architectures. In the ELL format, threads within a warp access consecutive elements in the padded column structure, allowing the GPU to combine multiple memory requests into fewer transactions. This results in more efficient utilization of memory bandwidth, as evidenced by the bandwidth measurements showing ELL achieving 7.40 GB/s compared to CSR's 6.96 GB/s at block size 64.

The bandwidth measurements reveal interesting behavior across different block sizes. At block size 32, ELL demonstrated substantially superior bandwidth utilization (6.87 GB/s) compared to CSR (1.29 GB/s), suggesting that smaller block sizes particularly benefit from ELL's regular structure. Conversely, at block sizes 96 and 128, CSR achieved better bandwidth (4.97 GB/s) than ELL (1.90 GB/s), indicating that the padding overhead in ELL becomes more problematic at larger block sizes. Despite these variations in bandwidth, ELL maintained consistently faster kernel execution times across all configurations, suggesting that factors beyond raw bandwidth such as memory access latency, cache utilization, and reduced thread divergence contribute to its performance advantage.

Host to Device Transfer Bandwidth (GB/s)

Block Size	CSR Bandwidth	ELL Bandwidth	Better Format
32	1.290	6.866	**ELL (5.3× faster)**
64	6.964	7.396	**ELL (1.06× faster)**
96	4.971	1.901	CSR (2.6× faster)
128	4.971	1.901	CSR (2.6× faster)

Other Findings:

GPU Overhead Dominates for Single Operations: A critical finding from this analysis is that GPU memory allocation and data transfer overhead significantly impacts the practical performance of GPU-accelerated SpMV for single operations. The GPU allocation phase alone required 191-251ms across different block sizes, which is orders of magnitude larger than the actual kernel execution time of 0.071-0.527ms. When accounting for the complete pipeline—including loading (595ms), conversion (27ms), GPU allocation (251ms), kernel execution (7ms), and storage (22ms) the total GPU-based solution required approximately 0.922 seconds. In stark contrast, the single-threaded CPU CSR implementation completed in just 0.022 seconds, and the optimized 20-thread CPU parallel implementation finished in an impressive 0.055 seconds. This means that for single SpMV operations, the CPU implementation is actually 12-16× faster than the GPU solution when overhead is included.

When GPUs Become Advantageous: These results highlight an important principle in GPU computing: GPUs excel at throughput rather than latency. The GPU would become the superior choice in scenarios where the SpMV kernel is executed repeatedly, such as in iterative solvers (Conjugate Gradient, GMRES, BiCGSTAB) where thousands of matrix-vector products are computed. In such cases, the allocation overhead is paid only once at the beginning, and the per-iteration kernel time of 0.071ms would represent the true cost. For iterative methods requiring 1,000 iterations, the GPU would complete the SpMV operations in approximately 71ms compared to the CPU's 22,000ms (single-threaded) or 55,000ms (20-thread parallel), demonstrating clear GPU superiority for high-throughput scenarios.