GPU SPMV Performance Evaluation: CSR vs. ELL

1. Overview

This report evaluates two GPU-accelerated Sparse Matrix–Vector Multiplication (SpMV) implementations:

- CSR-based SpMV (row-compressed format)
- ELL-based SpMV (Ellpack format)

Both implementations were integrated into the provided CUDA codebase, and the missing functions were fully implemented:

1. `convert_csr_to_ell()`
2. `allocate_csr_gpu()`
3. `allocate_ell_gpu()`
4. `spmv_kernel()` (CSR kernel)
5. `spmv_kernel_ell()` (ELL kernel)

The matrix used was `cant.mtx`, and all experiments were executed with `MAX_ITER = 100` iterations for stable timing.

CUDA version note: The Talapas cluster required switching CUDA from the default version to **CUDA 11.2.0**. Because of this, we updated the environment setup in `spmv.srun`

This resolved compatibility issues and ensured that both CSR and ELL kernels compiled correctly.

2. Implementation Summary

2.1 CSR GPU Implementation

The CSR GPU kernel assigns one thread block per matrix row:

- Threads iterate over the nonzeros of the row using a strided loop.
- Partial sums are stored in `sdata[]` in shared memory.
- A parallel reduction computes the final row result.

GPU execution parameters:

```
blocks  = GRID_SIZE
threads = BLOCK_SIZE
shared  = threads * sizeof(double)
```

2.2 ELL GPU Implementation

The ELL format stores exactly `max_nnz_per_row` values per row. Each GPU block computes a single row:

- Threads iterate across the padded column slots.
- Zeros represent padded entries and are skipped.
- Shared-memory reduction is performed identically to CSR.

Because ELL uses column-major storage (`index = k*m + row`), memory accesses are more regular but sometimes include padded zeros.

2.3 Converting CSR to ELL

We implement:

1. Compute the maximum nonzeros in any row
2. Allocate `m * max_nnz` slots
3. Copy each row's columns/values into ELL's padded layout

This ensures GPU-friendly, uniform row lengths.

2.4 GPU Memory Allocation

Both `allocate_csr_gpu` and `allocate_ell_gpu` use:

- `cudaMallocHost()` for pinned CPU memory
- `cudaMemcpy()` for fast transfer to device

This significantly improves PCIe bandwidth.

3. Experiment Design

We varied two execution parameters passed via `GRID_SIZE` and `BLOCK_SIZE`:

$$\text{Block sizes} = \{32,\ 64,\ 128,\ 256,\ 512\}$$

$$\text{Grid sizes} = \{3903,\ 7806,\ 15612,\ 31225,\ 62451\}$$

These correspond to:

- different numbers of thread blocks,
- different numbers of GPU threads per block.

For each configuration, we recorded:

- CSR execution time
- ELL execution time
- Speedup: `ELL/CSR`

4. Results and Analysis

4.1 Execution Time Differences

ELL is consistently slower than CSR across all parameter combinations. This is expected because:

1. ELL includes padded entries, increasing total operations
2. CSR has perfectly compact nonzero storage
3. CSR memory accesses are less wasteful

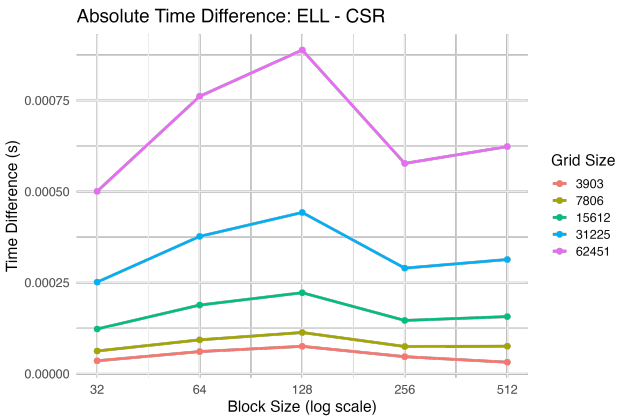

Figure 1: Execution time difference (ELL – CSR).

4.2 CSR vs ELL per Grid Size

Execution times rise smoothly with block size. CSR remains faster in every case, especially for large matrices.

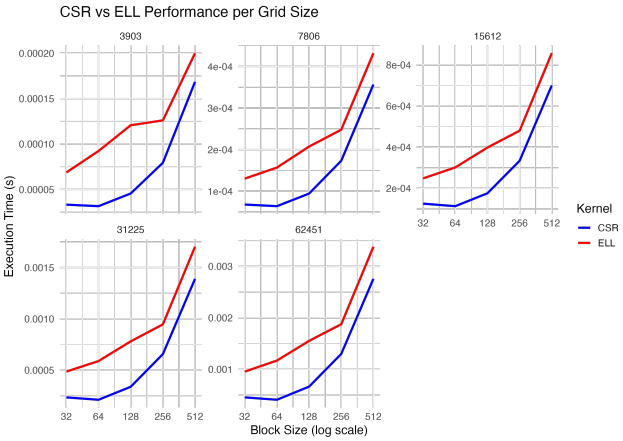

Figure 2: CSR vs ELL execution time for each grid size.

4.3 Speedup Behavior

Speedup is defined as:

$$\text{Speedup} = \frac{\text{ELL time}}{\text{CSR time}}$$

Values above 1.0 mean CSR is faster.

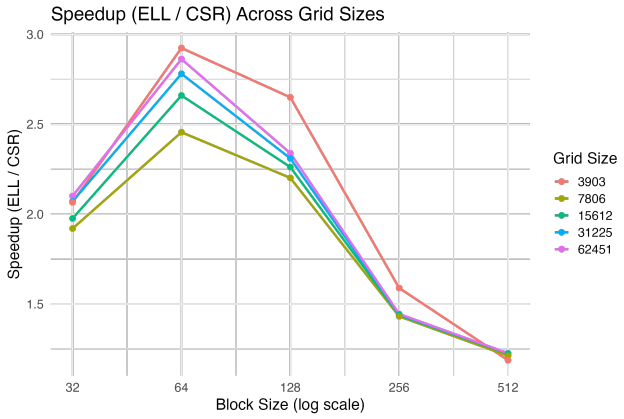

Figure 3: Speedup (ELL / CSR). Lower is better for ELL.

Observations:

- ELL is worst at small block sizes (32–64)
- Speedup peaks near 64 threads per block
- For large block sizes (256–512), ELL performance declines sharply

This behavior matches ELL's sensitivity to padded elements.

5. Conclusions

- CSR clearly outperforms ELL for SpMV on GPUs in all tests.
- ELL suffers from padded zero entries and performs more work.
- CSR scales smoothly with block size and grid size.
- ELL does not gain performance from larger blocks; padding overhead grows.
- Switching CUDA to 11.2.0 was necessary for successful compilation and execution.

Overall, CSR is the preferred GPU format for sparse matrix–vector multiplication in this assignment.