Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors

Nathan Bell and Michael Garland

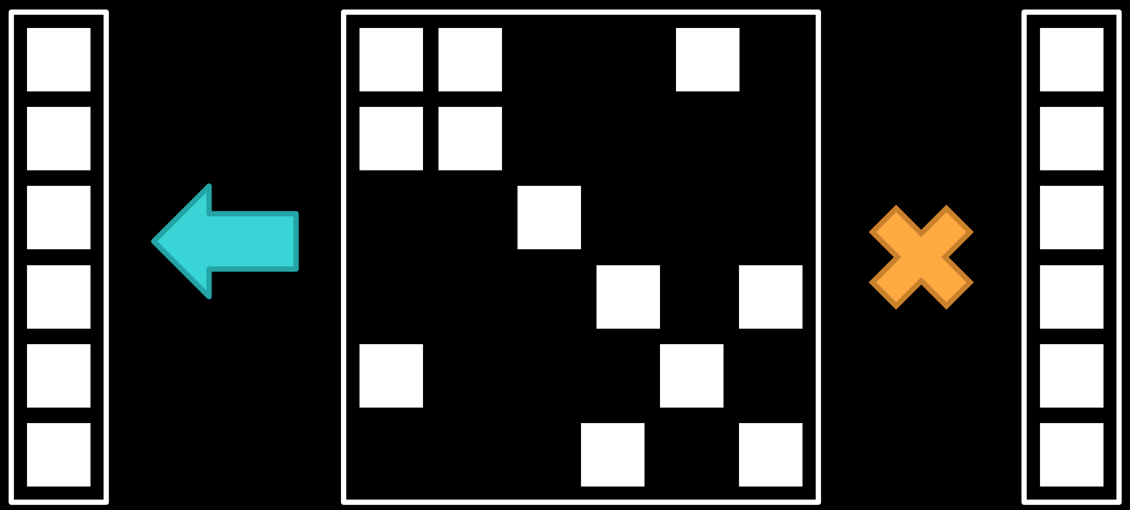
NVIDIA Research

Overview

- GPUs deliver high SpMV performance
  - 10+ GFLOP/s on unstructured matrices
  - 140+ GByte/s memory bandwidth
- No one-size-fits-all approach
  - Match method to matrix structure
- Exploit structure when possible
  - Fast methods for regular portion
  - Robust methods for irregular portion

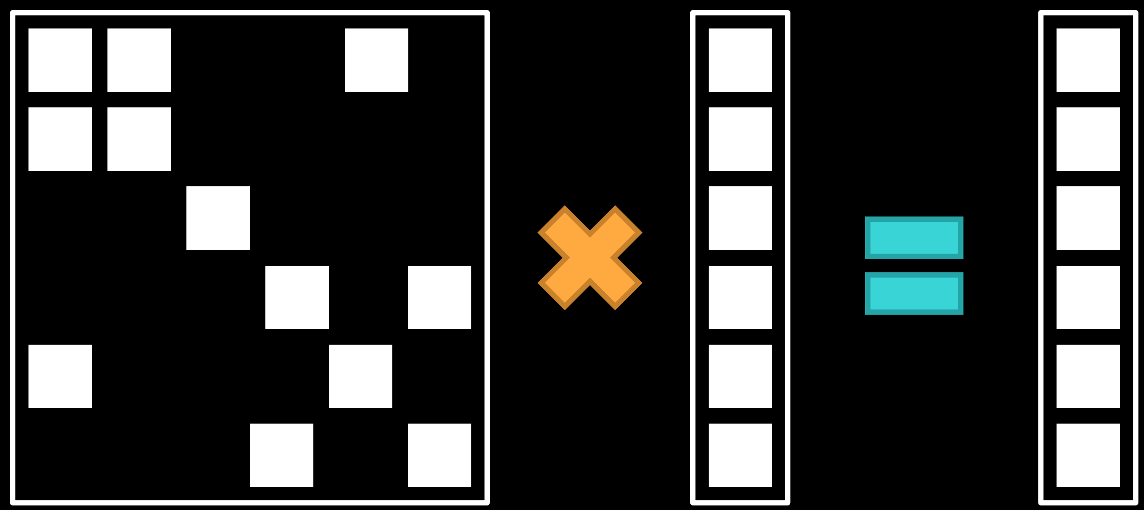
Characteristics of SpMV

- Memory bound
  - FLOP : Byte ratio is very low
- Generally irregular & unstructured
  - Unlike dense matrix operations (BLAS)



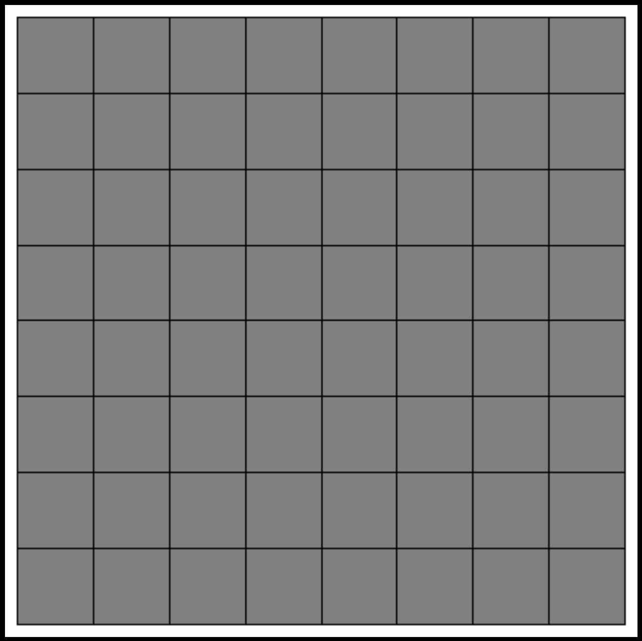
Solving Sparse Linear Systems

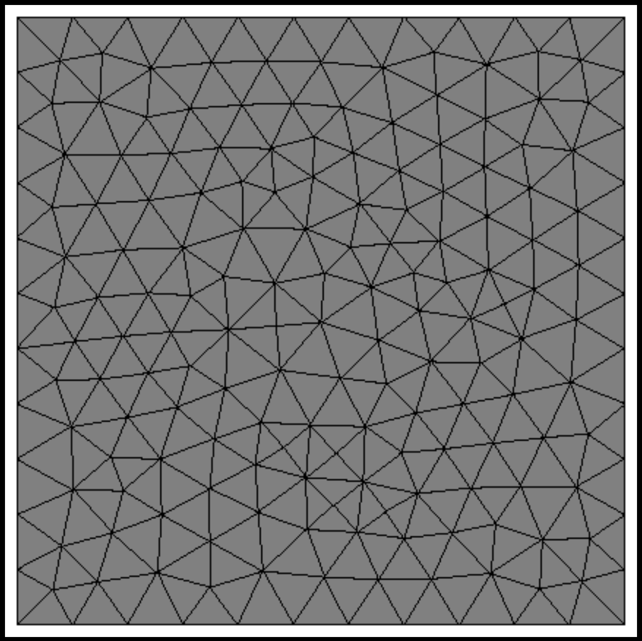
- Iterative methods
  - CG, GMRES, BiCGstab, etc.
  - Require 100s or 1000s of SpMV operations



Finite-Element Methods

- Discretized on structured or unstructured meshes
  - Determines matrix sparsity structure

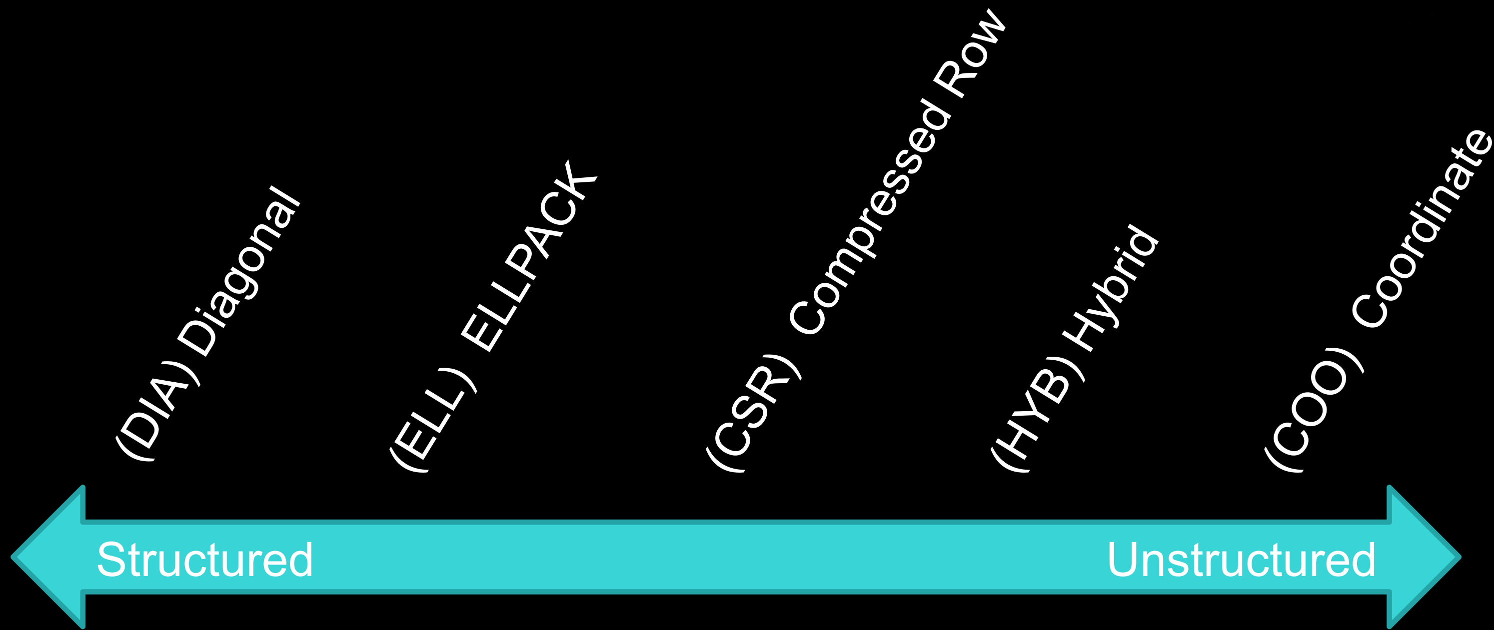


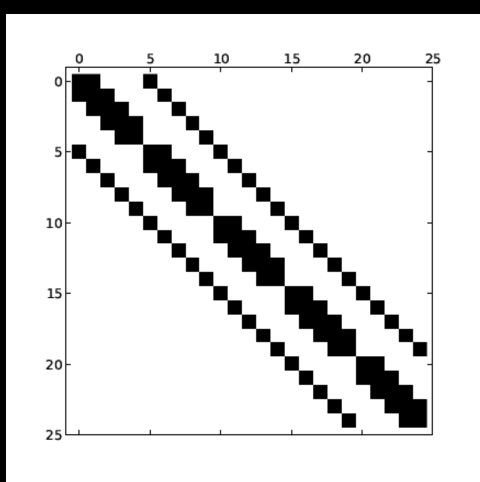


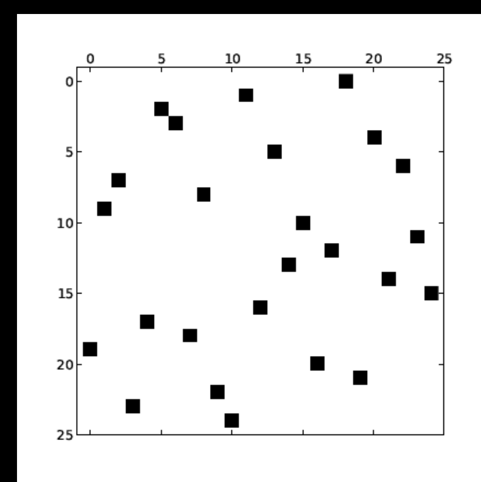
Objectives

- Expose sufficient parallelism
  - Develop 1000s of independent threads
- Minimize execution path divergence
  - SIMD utilization
- Minimize memory access divergence
  - Memory coalescing

Sparse Matrix Formats

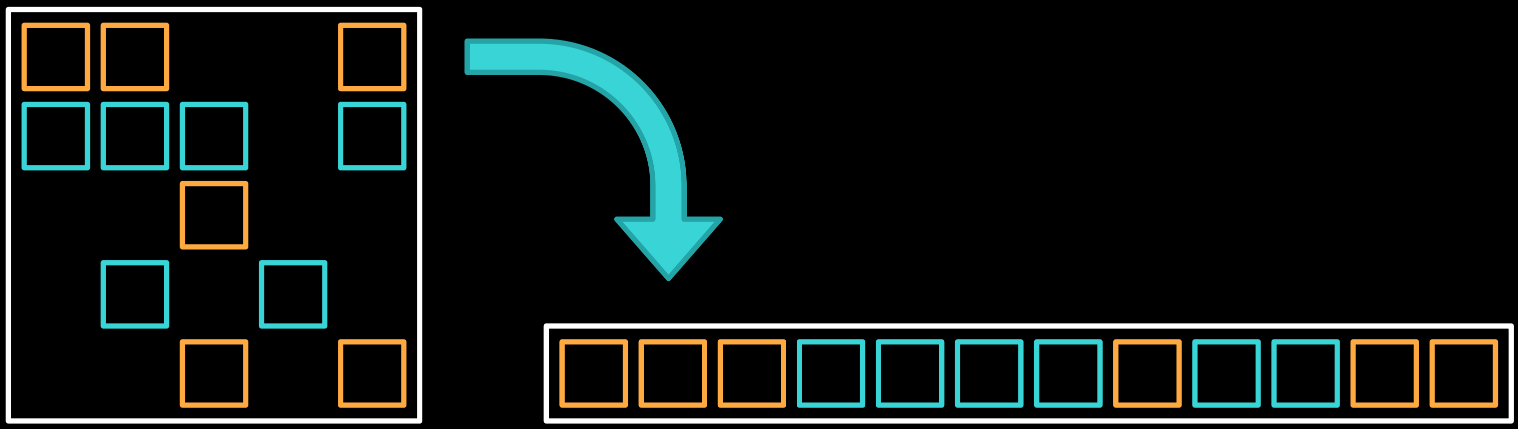






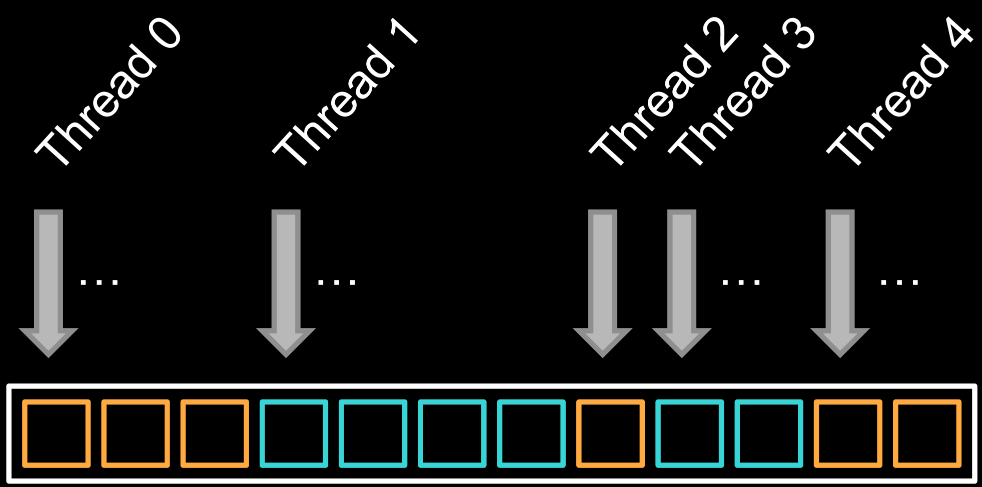
Compressed Sparse Row (CSR)

- Rows laid out in sequence
- Inconvenient for fine-grained parallelism



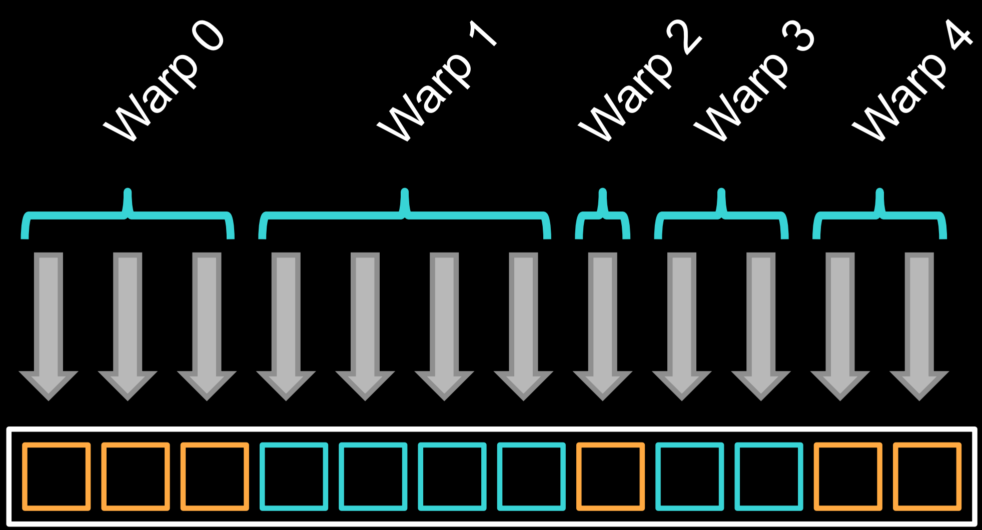
CSR (scalar) kernel

- One thread per row
  - Poor memory coalescing
  - Unaligned memory access



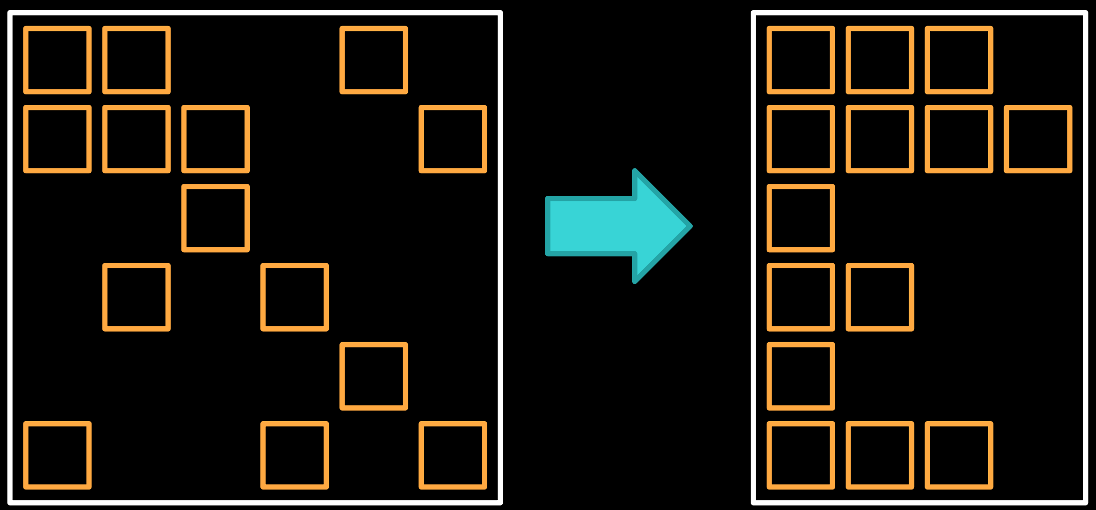
CSR (vector) kernel

- One SIMD vector or *warp* per row
  - Partial memory coalescing
  - Unaligned memory access



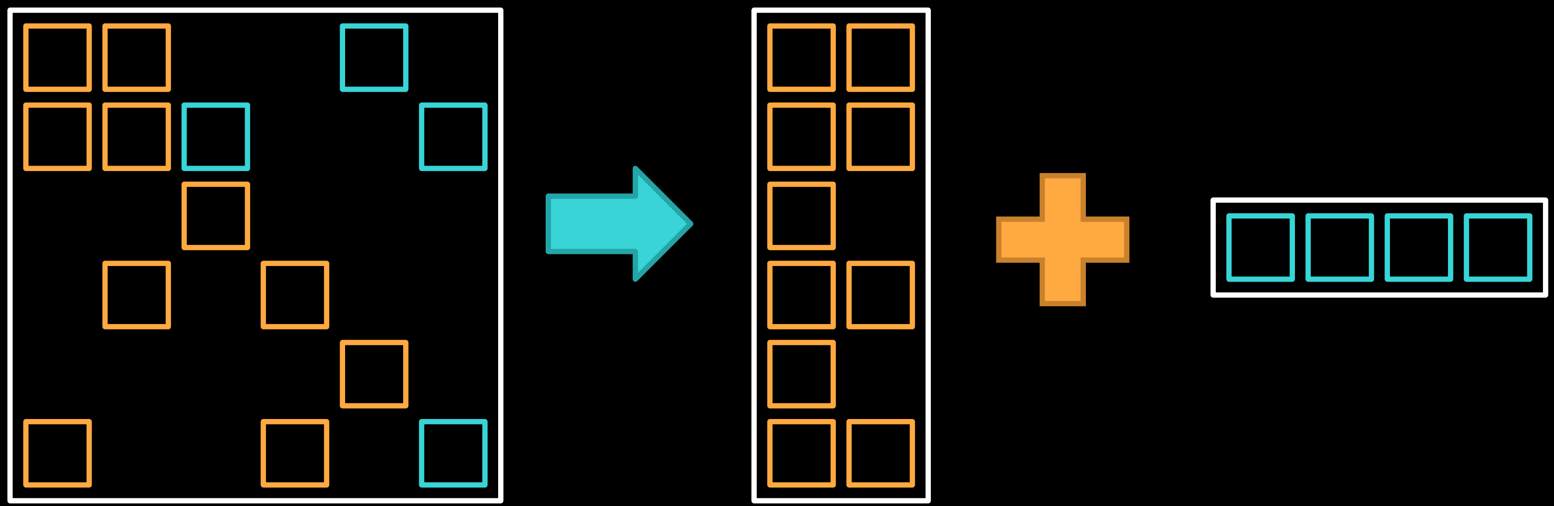
ELLPACK (ELL)

- Storage for K nonzeros per row
  - Pad rows with fewer than K nonzeros
  - Inefficient when row length varies



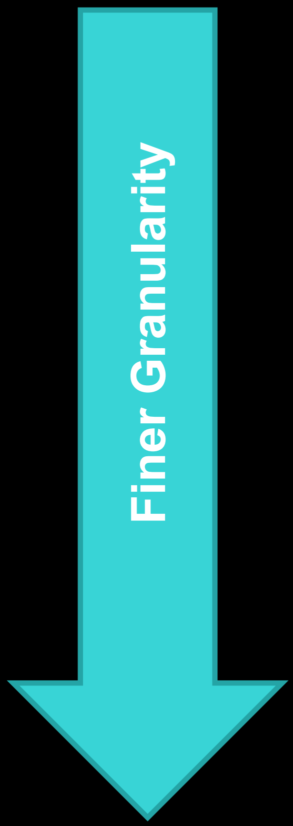
Hybrid Format

- ELL handles *typical* entries
- COO handles *exceptional* entries
  - Implemented with segmented reduction

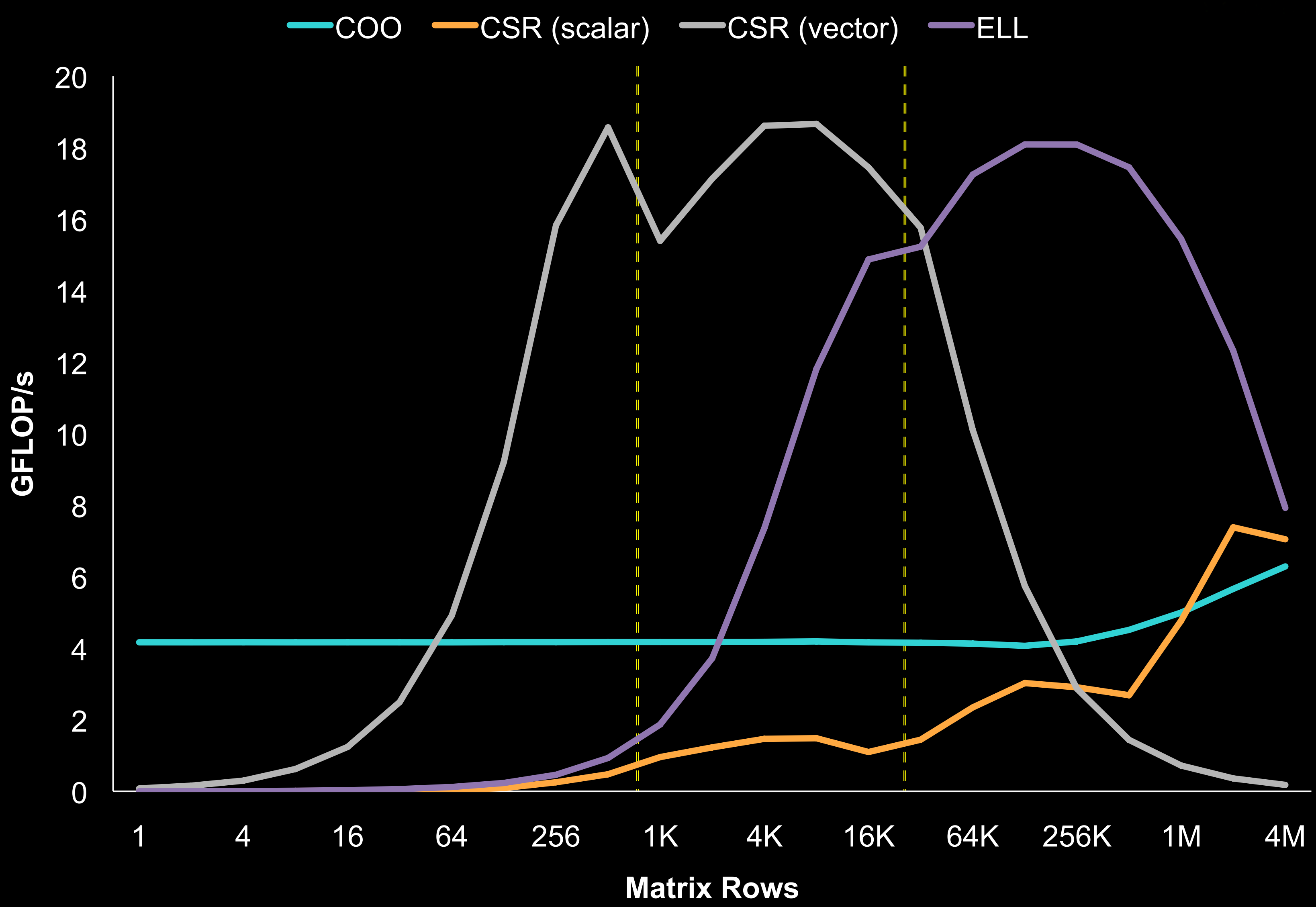


Exposing Parallelism

- DIA, ELL & CSR (scalar)
  - One thread per row
- CSR (vector)
  - One warp per row
- COO
  - One thread per nonzero



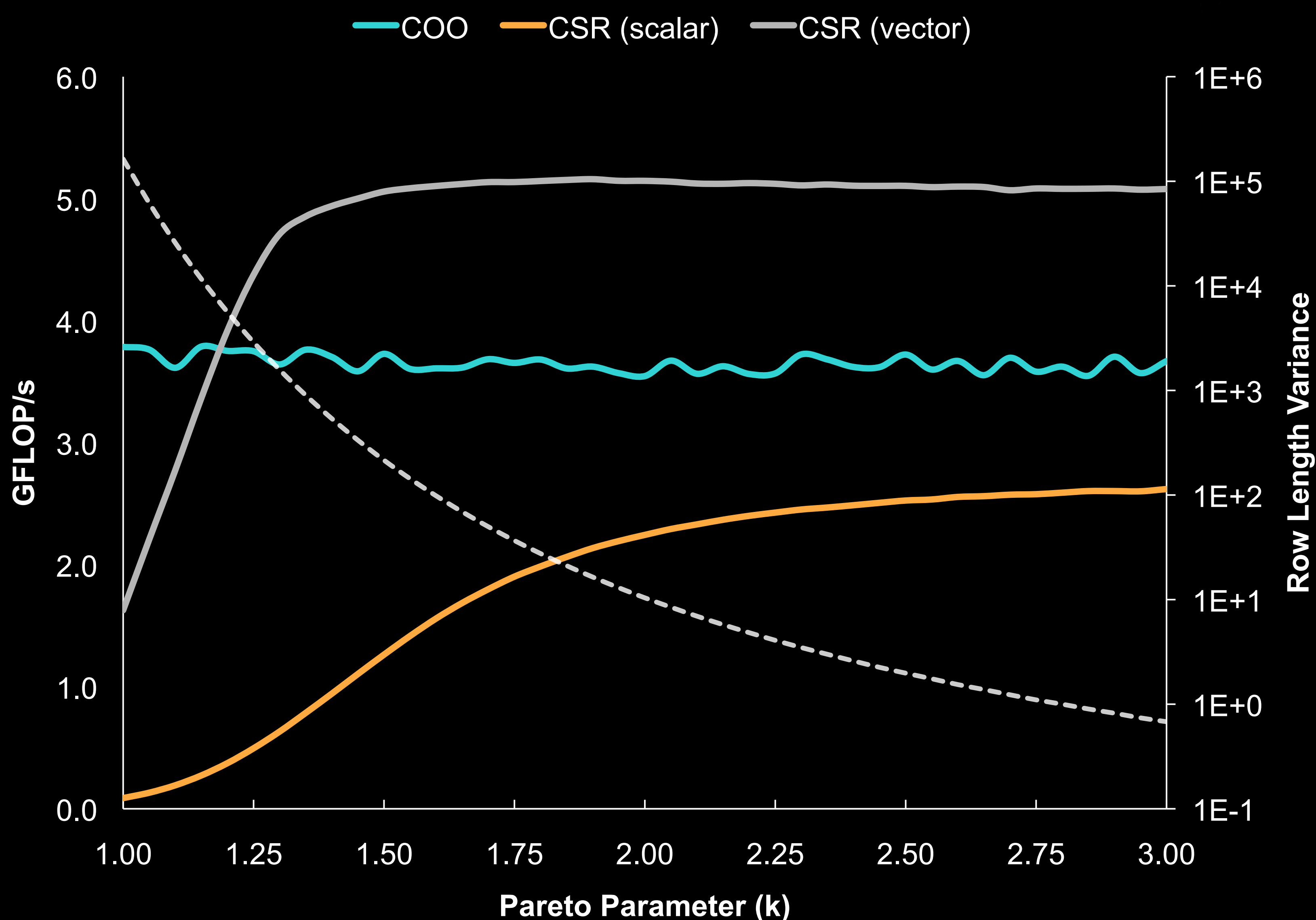
Exposing Parallelism



Execution Divergence

- Variable row lengths can be problematic
  - Idle threads in CSR (scalar)
  - Idle processors in CSR (vector)
- Robust strategies exist
  - COO is insensitive to row length

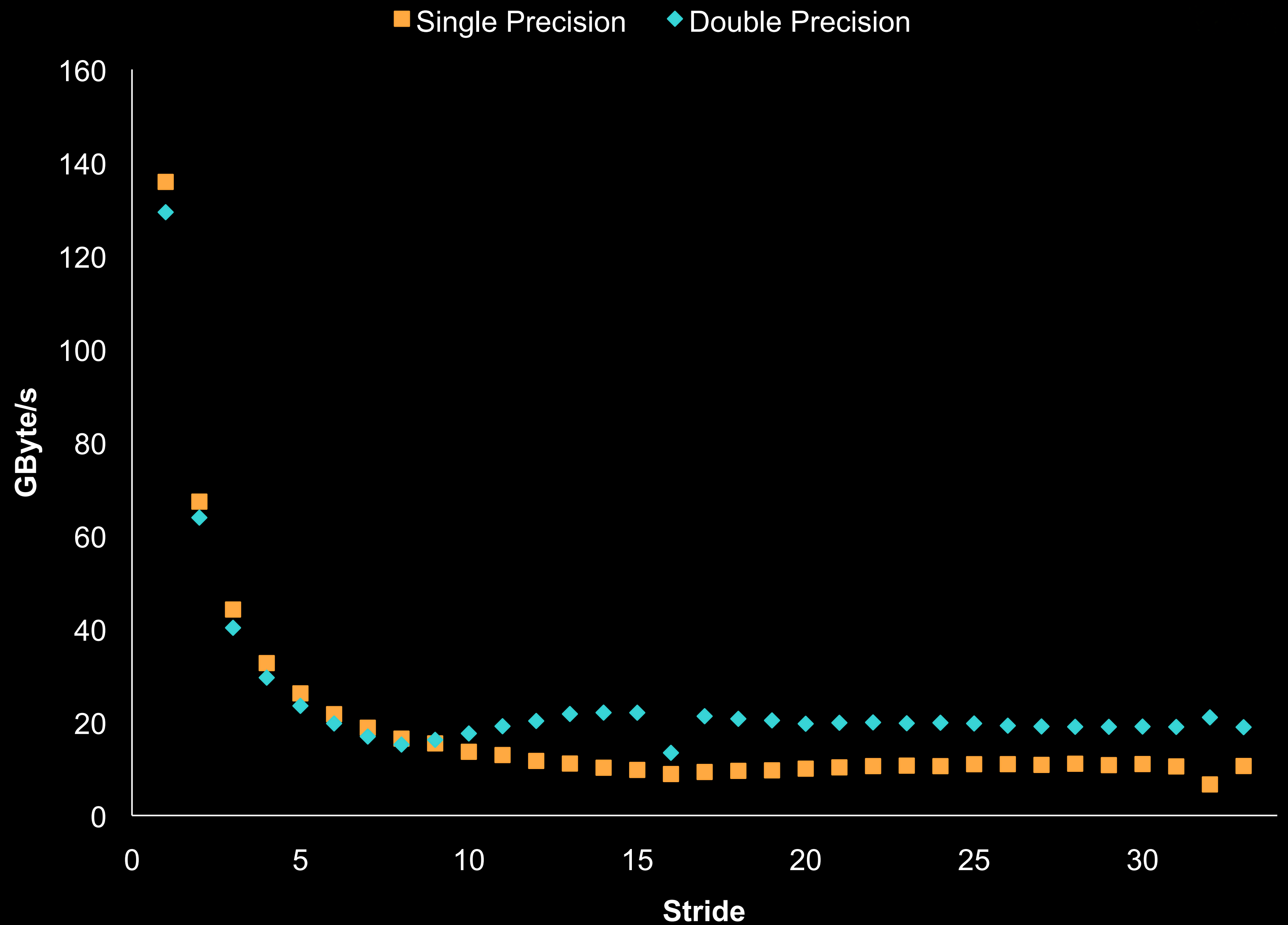
Execution Divergence



Memory Access Divergence

- Uncoalesced memory access is very costly
  - Sometimes mitigated by cache
- Misaligned access is suboptimal
  - Align matrix format to coalescing boundary
- Access to matrix representation
  - DIA, ELL and COO are fully coalesced
  - CSR (vector) is partially coalesced
  - CSR (scalar) is seldom coalesced

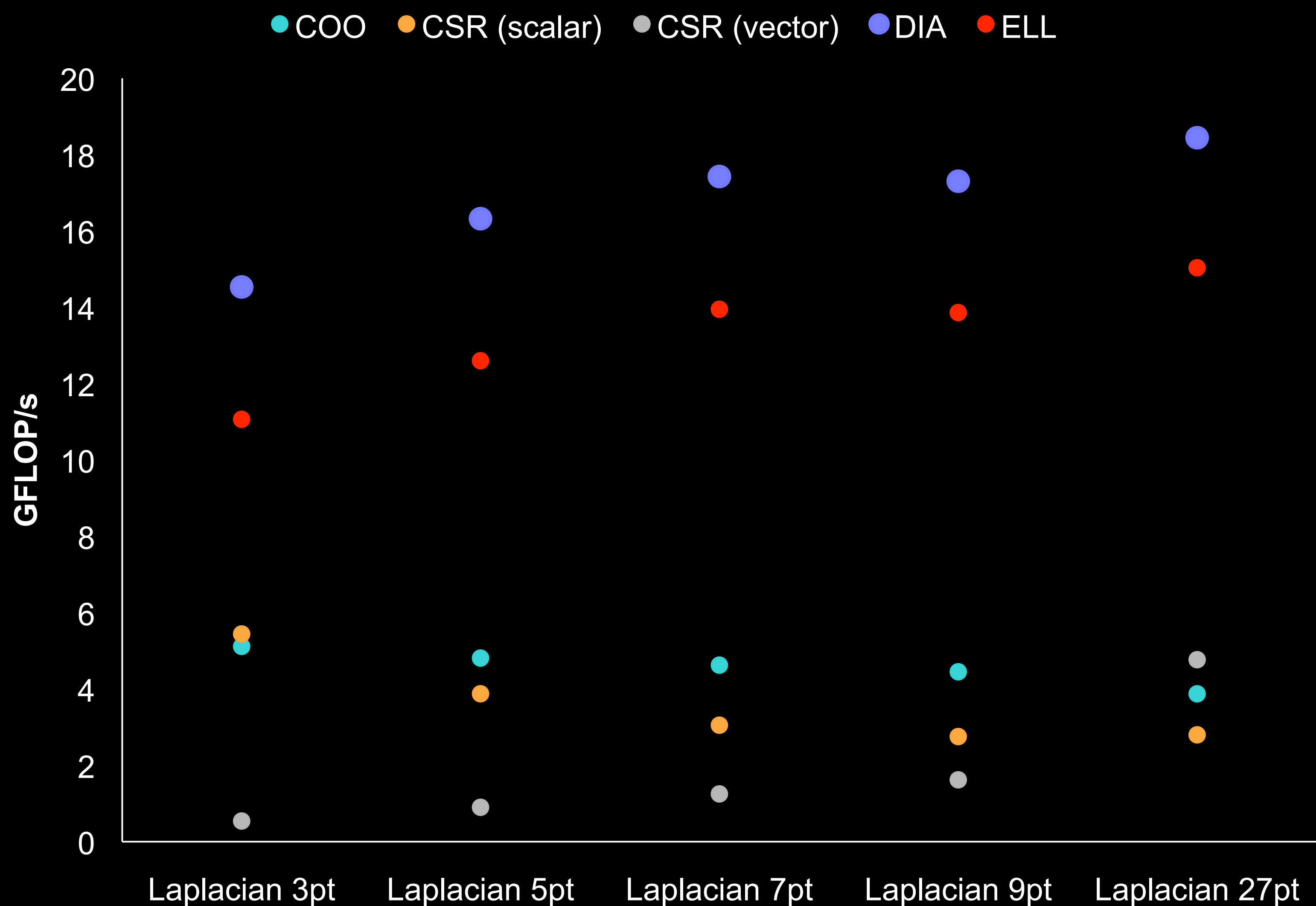
Memory Bandwidth (AXPY)



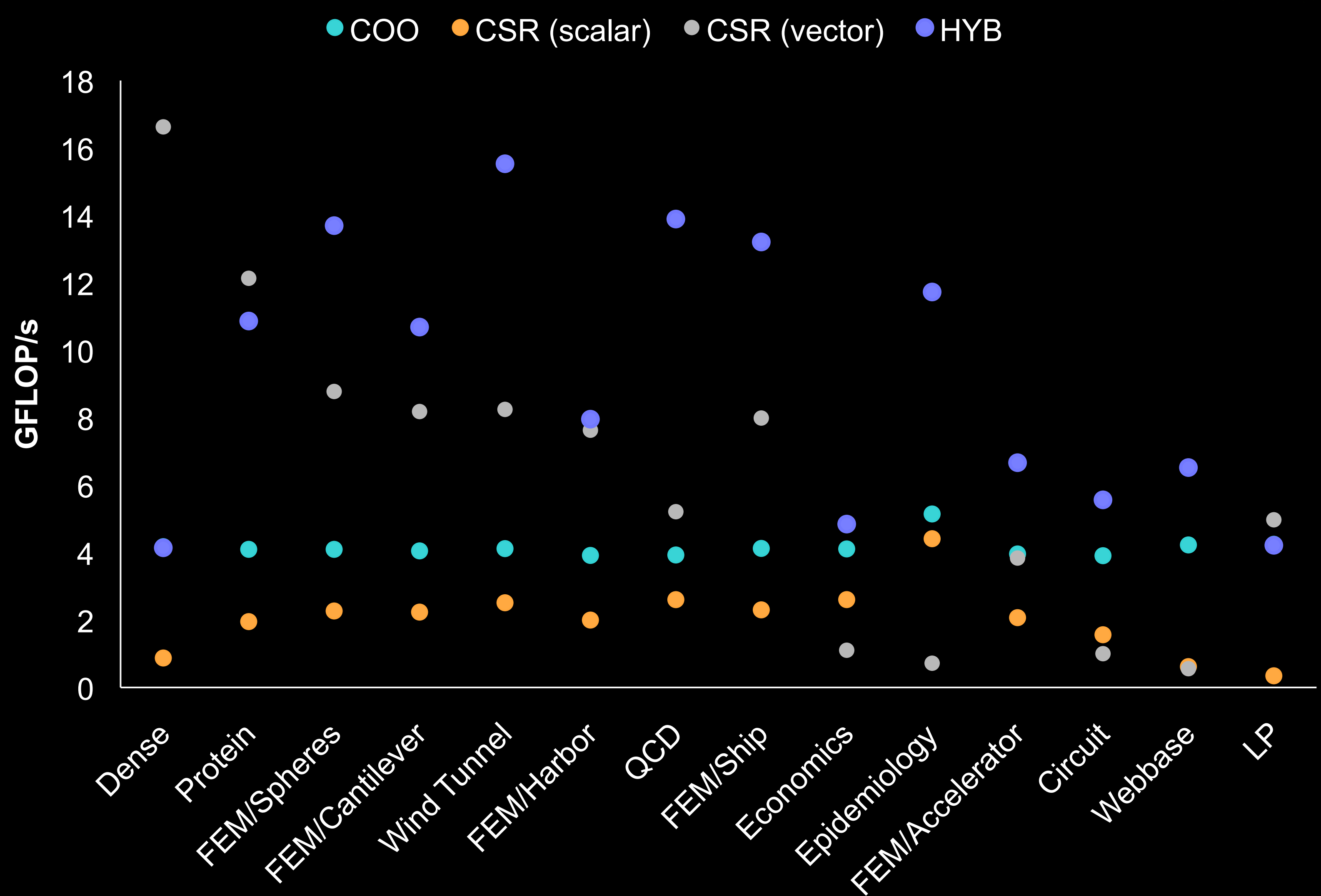
Performance Results

- GeForce GTX 285
  - Peak Memory Bandwidth: 159 GByte/s
  - All results in double precision
  - Source vector accessed through texture cache
- Structured Matrices
  - Common stencils on regular grids
- Unstructured Matrices
  - Wide variety of applications and sparsity patterns

Structured Matrices



Unstructured Matrices



Performance Comparison

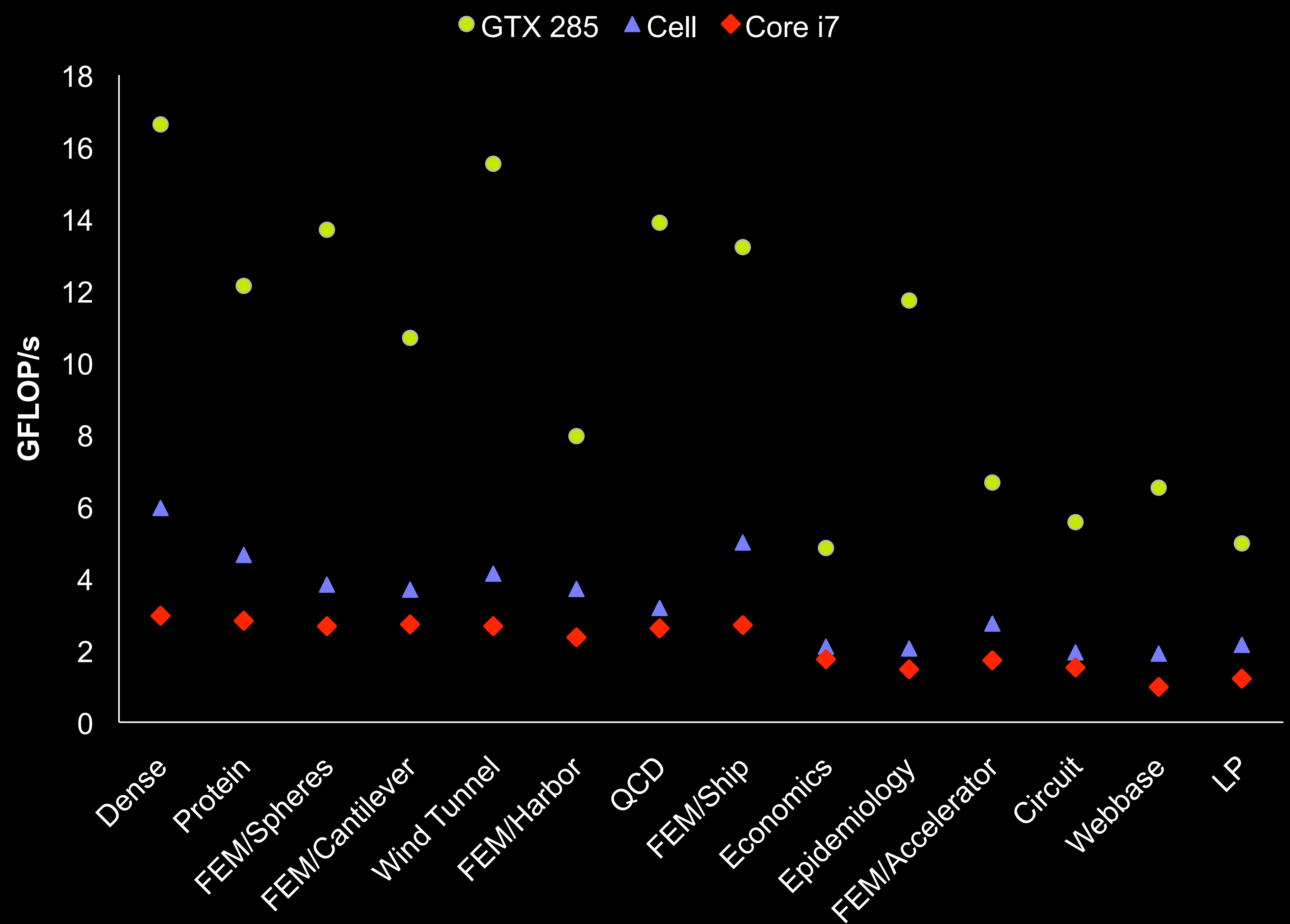
System	Cores	Clock (GHz)	Notes
GTX 285	240	1.5	NVIDIA GeForce GTX 285
Cell	8 (SPEs)	3.2	IBM QS20 Blade (half)
Core i7	4	3.0	Intel Core i7 (Nehalem)

Sources:

Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors
N. Bell and M. Garland, Proc. Supercomputing '09, November 2009

Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms
Samuel Williams et al., Supercomputing 2007.

Performance Comparison



ELL kernel

```
__global__ void ell_spmv(const int num_rows,           const int num_cols,
                         const int num_cols_per_row, const int stride,
                         const double * Aj,          const double * Ax,
                         const double * x,                 double * y)
{
    const int thread_id = blockDim.x * blockIdx.x + threadIdx.x;
    const int grid_size = gridDim.x * blockDim.x;

    for (int row = thread_id; row < num_rows; row += grid_size) {
        double sum = y[row];

        int offset = row;

        for (int n = 0; n < num_cols_per_row; n++) {
            const int col = Aj[offset];

            if (col != -1)
                sum += Ax[offset] * x[col];

            offset += stride;
        }

        y[row] = sum;
    }
}
```

```
#include <cusp/hyb_matrix.h>
#include <cusp/io/matrix_market.h>
#include <cusp/krylov/cg.h>

int main(void)
{
   // create an empty sparse matrix structure (HYB format)
   cusp::hyb_matrix<int, double, cusp::device_memory> A;

   // load a matrix stored in MatrixMarket format
   cusp::io::read_matrix_market_file(A, "5pt_10x10.mtx");

   // allocate storage for solution (x) and right hand side (b)
   cusp::array1d<double, cusp::device_memory> x(A.num_rows, 0);
   cusp::array1d<double, cusp::device_memory> b(A.num_rows, 1);

   // solve linear system with the Conjugate Gradient method
   cusp::krylov::cg(A, x, b);

   return 0;
}
```

CUSP

http://cusp-library.googlecode.com

Extensions & Optimizations

- Block formats (register blocking)
  - Block CSR
  - Block ELL
- Block vectors
  - Solve multiple RHS
  - Block Krylov methods
- Other optimizations
  - Better CSR (vector)

Further Reading

Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors
N. Bell and M. Garland
Proc. Supercomputing '09, November 2009

Efficient Sparse Matrix-Vector Multiplication on CUDA
N. Bell and M. Garland
NVIDIA Tech Report NVR-2008-004, December 2008

Optimizing Sparse Matrix-Vector Multiplication on GPUs
M. M. Baskaran and R. Bordawekar.
IBM Research Report RC24704, IBM, April 2009

Model-driven Autotuning of Sparse Matrix-Vector Multiply on GPUs
J. W. Choi, A. Singh, and R. Vuduc
Proc. ACM SIGPLAN (PPoPP), January 2010

Questions?

nbell@nvidia.com