Sparse Matrix Representations & Iterative Solvers

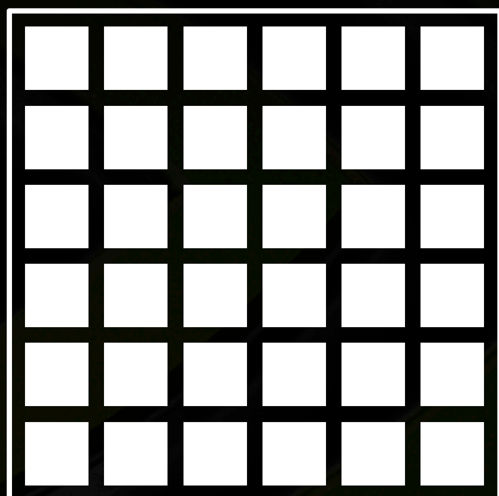
Lesson 1

Nathan Bell

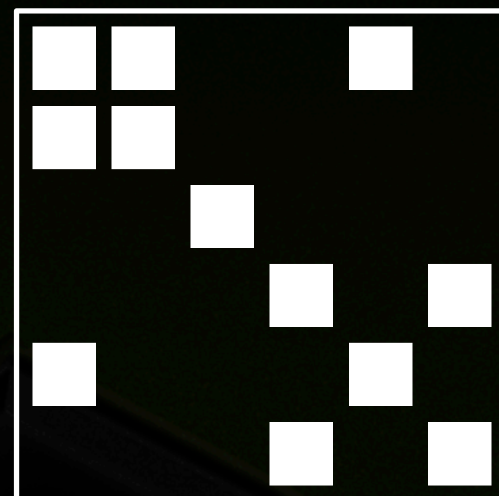
NVIDIA

Sparse Matrices

- **Have small number of non-zero entries**



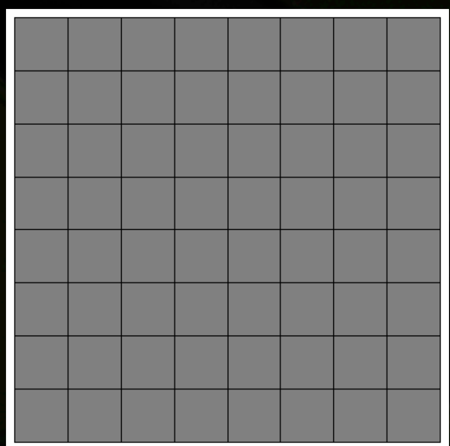
Dense Matrix

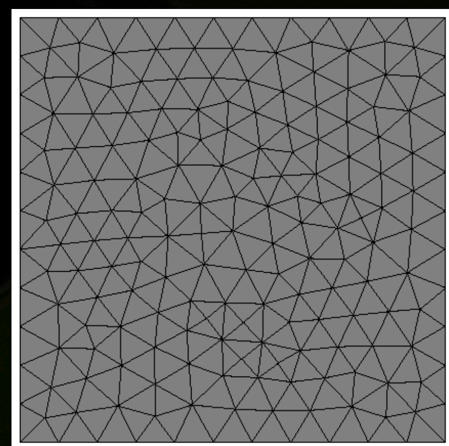


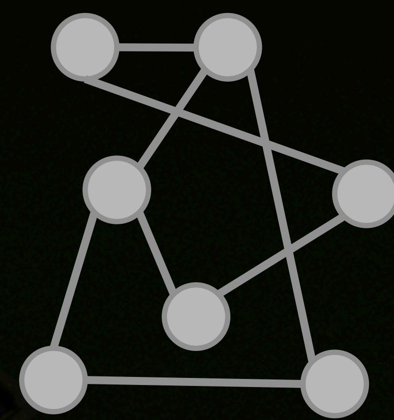
Sparse Matrix

Sparse Matrices

- Non-zeros encode connectivity
  - Finite-Element Meshes, Hyperlinks, Social Networks, …

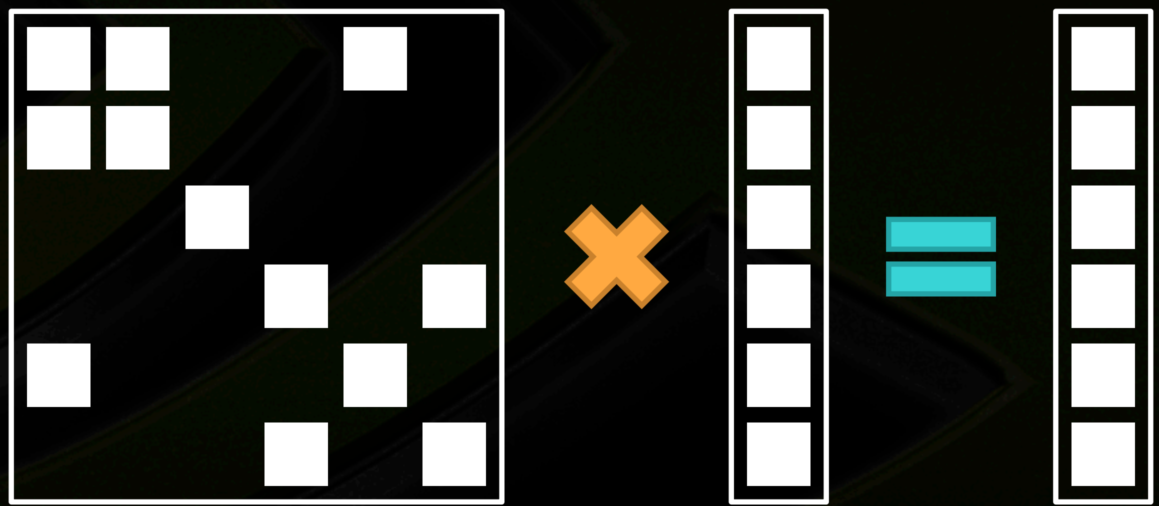

Structured Mesh


Unstructured Mesh


General Graph

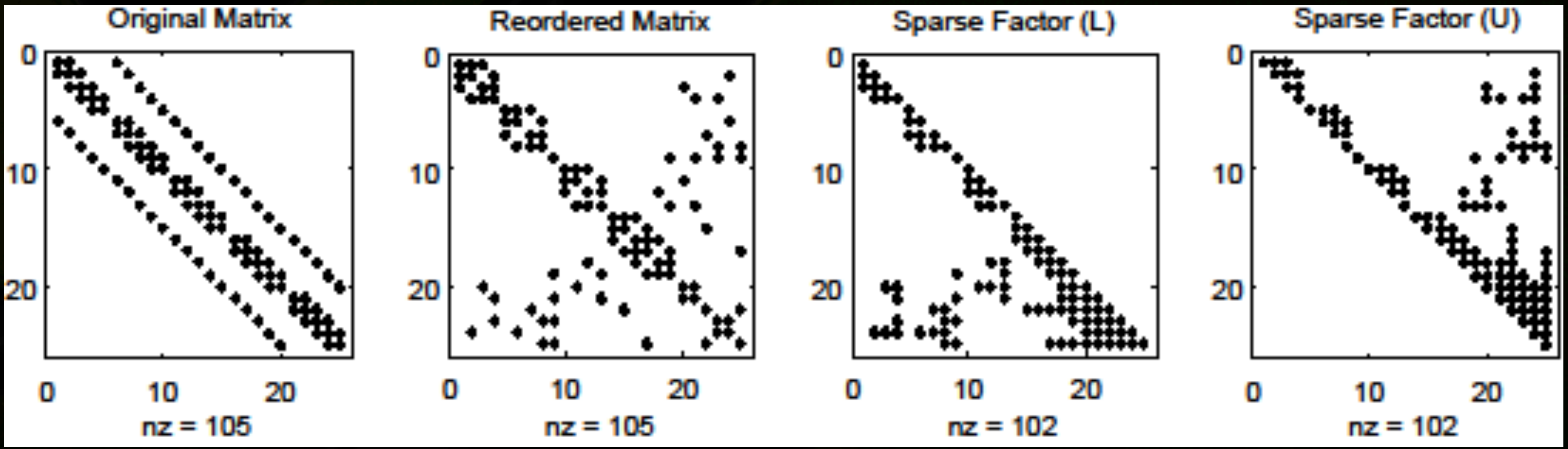
Sparse Solvers

- Solve sparse linear system (A*x = b)
- Variety of *direct* and *iterative* methods



Direct Solvers

- Sparse matrix factorization
  - Try to reduce fill-in of factors
  - Use sophisticated reordering schemes
- Popular codes: PARDISO, UMFPACK, SuperLU, …



Iterative Solvers

- Sequence of approximate solutions
  - Converge to exact solution
- Measure error through residual
  - residual = b – A * x
  - residual = A * (x’ - x) = A * error
  - Stop when || b – A* x || < tolerance
- Wide variety of methods
  - Jacobi
  - Gauss-Seidel
  - Conjugate-Gradient (CG)
  - Generalized Minimum-Residual (GMRES)

Comparison

Direct Solvers

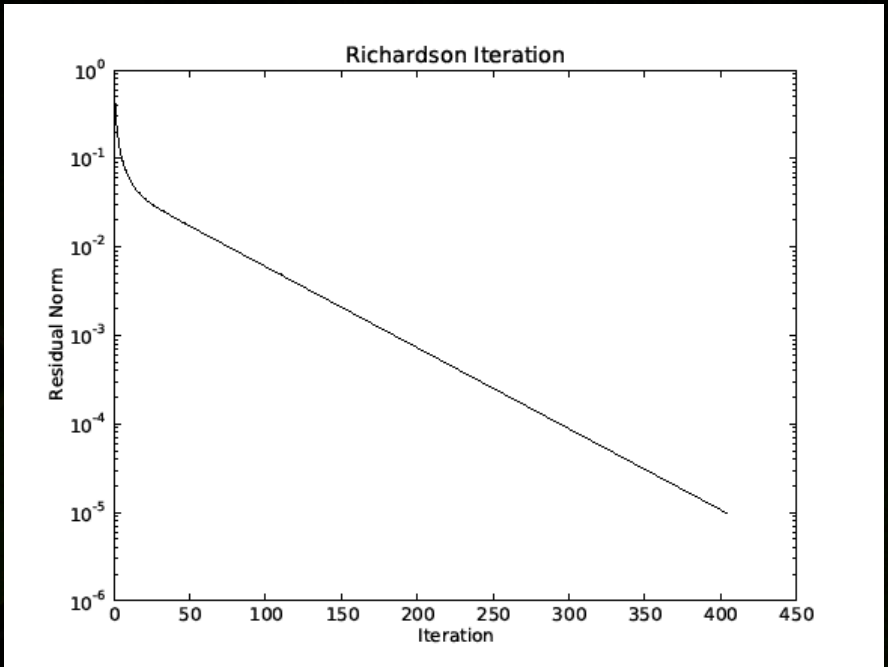
- Robust
- Black-box operation
- Difficult to parallelize
- Memory consumption
- Limited scalability

Iterative Solvers

- Breakdown issues
- Lots of parameters
- Amenable to parallelism
- Low memory footprint
- Scalable

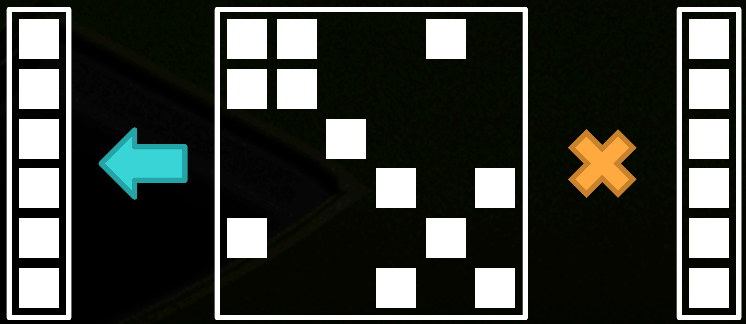
Example: Richardson Iteration

```
r = b - A * x
while norm(r) > tol
  x = x + omega * r
  r = b - A * x
```



Iterative Solver Components

- Sparse Matrix-Vector Multiplication (SpMV)
  - `y = A * x`
- Vector Scale and Add (SAXPY)
  - `x = x + omega * r`
- Vector Norm (SNRM2)
  - `norm(r)`



Sparse Matrix Storage Formats

- **Coordinate (COO)**

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

0	0	1	1	2	2	2	3	3	row indices
0	1	1	2	0	2	3	1	3	column indices
1	7	2	8	5	3	9	6	4	values

entries

Sparse Matrix Storage Formats

- **Compressed Sparse Row (CSR)**

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

rows + 1

row offsets	0	2	4	7	9				
column indices	0	1	1	2	0	2	3	1	3
values	1	7	2	8	5	3	9	6	4

entries

Sparse Matrix Storage Formats

- **ELLPACK (ELL)**

$$
\underset{\text{Matrix}}{\begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}}
\qquad
\underset{\text{column indices}}{\overbrace{\begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 1 & 2 \\ 1 & 3 & * \end{bmatrix}}^{\text{\# entries per row}}}
\qquad
\underset{\text{values}}{\begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix}}
\left.\vphantom{\begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix}}\right\}\text{\# rows}
$$

padding

Sparse Matrix Storage Formats

- **Diagonal (DIA)**

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

diagonals

*	1	7
*	2	8
5	3	9
6	4	*

rows

values

Sparse Matrix Storage Formats

- **Hybrid (HYB) ELL + COO**

1	7	0	0
0	2	8	0
5	0	3	9
0	6	0	4

Matrix

ELL

0	1
1	2
0	1
1	3

column indices

1	7
2	8
5	3
6	4

values

COO

2	row indices
3	column indices
9	values

Storage Format Comparison

(DIA) Diagonal

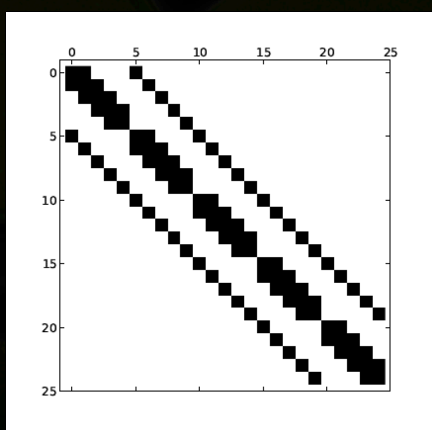
(ELL) ELLPACK

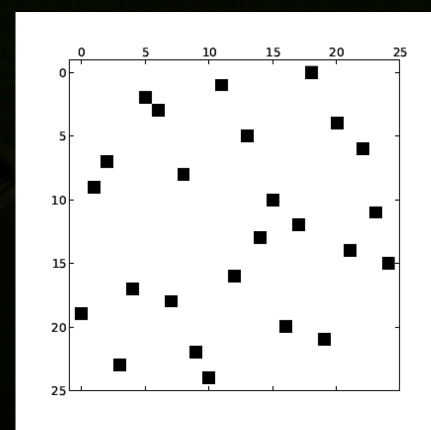
(CSR) Compressed Row

(HYB) Hybrid

(COO) Coordinate

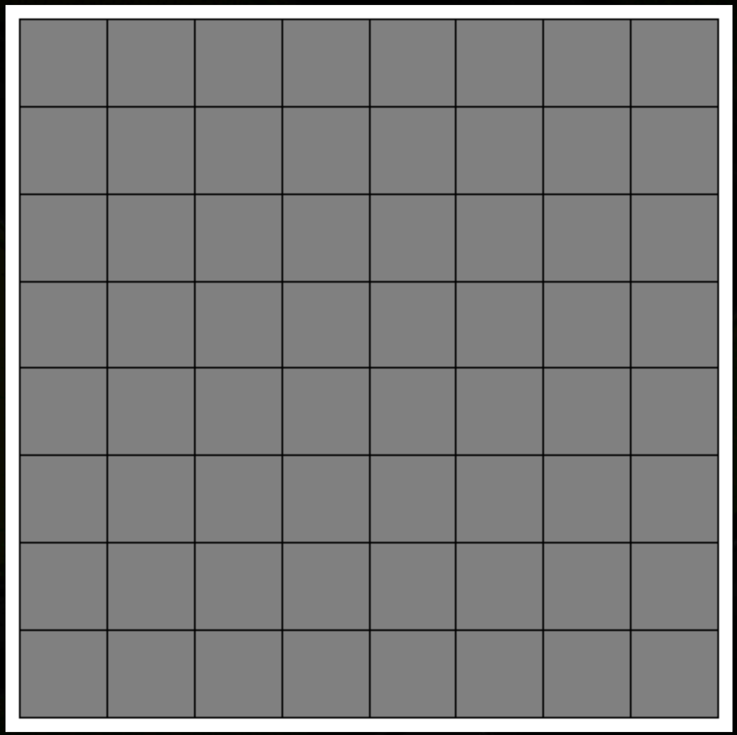
Structured ⟷ Unstructured

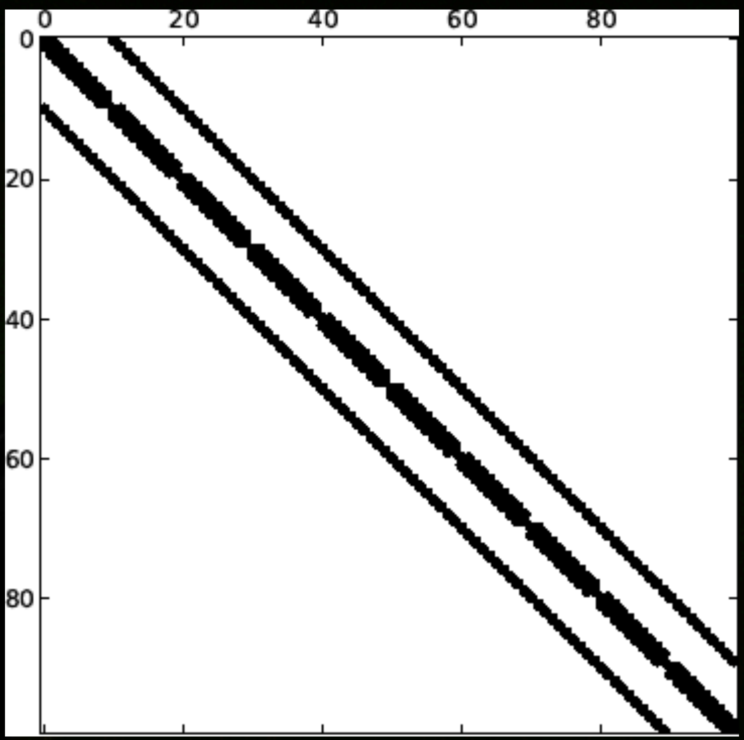




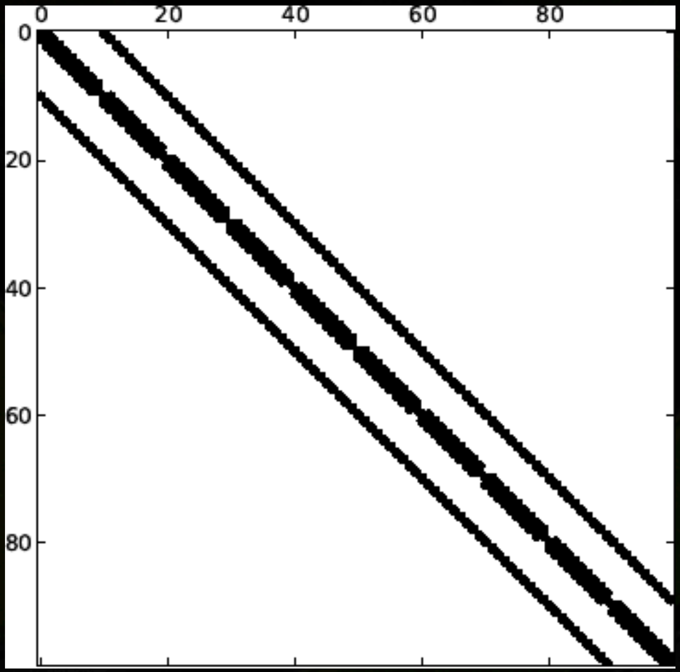
Storage Format Comparison

- **Structured Mesh**





Storage Format Comparison



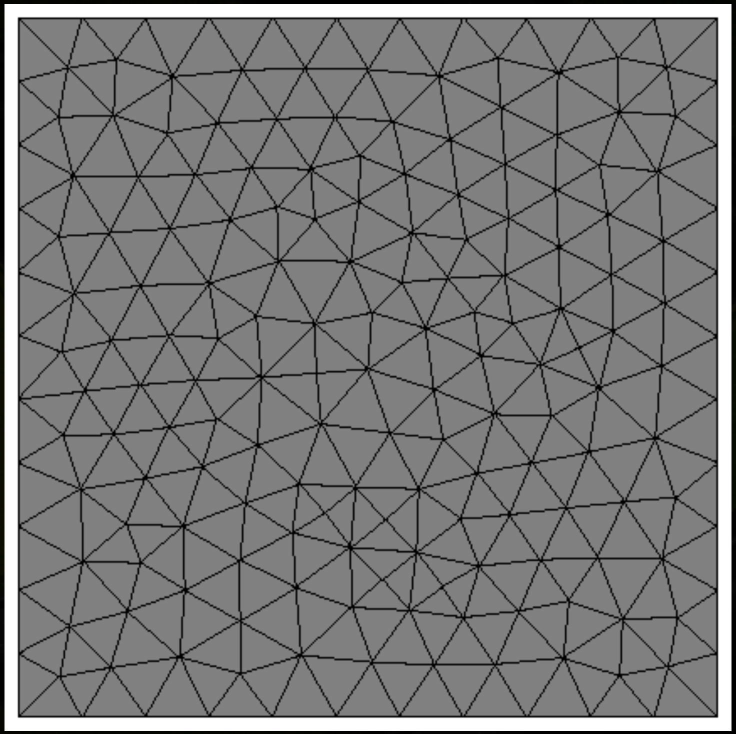
Matrix

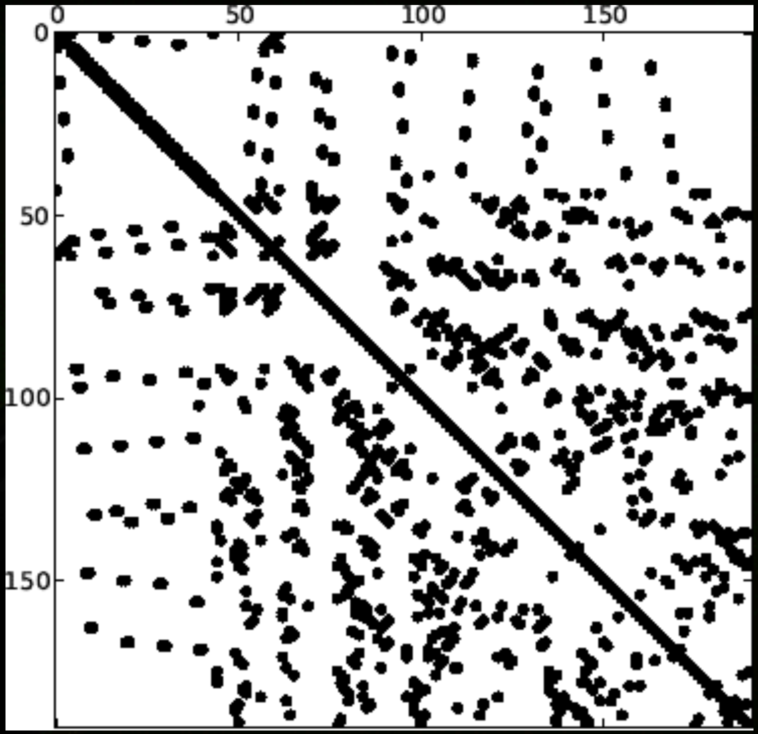
Format	float	double
COO	12.00	16.00
CSR	8.45	12.45
DIA	4.05	8.10
ELL	8.11	12.16
HYB	8.11	12.16

Bytes per Nonzero Entry

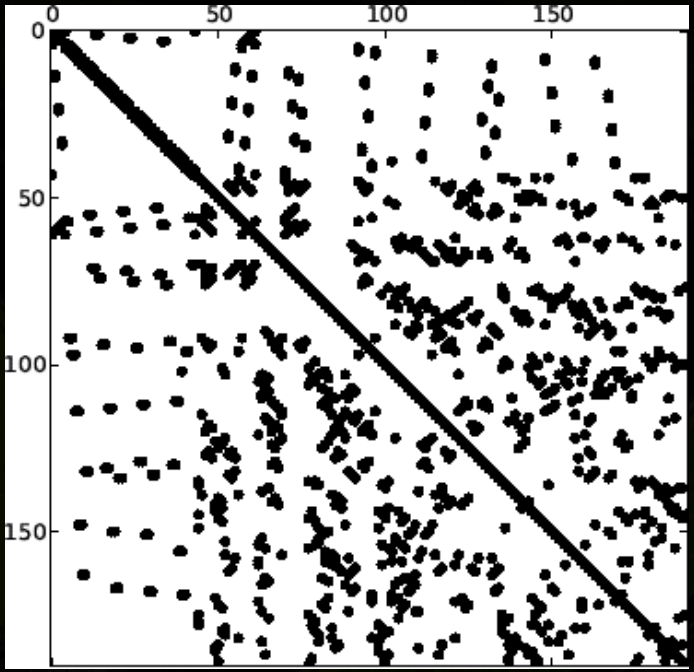
Storage Format Comparison

- **Unstructured Mesh**





Storage Format Comparison



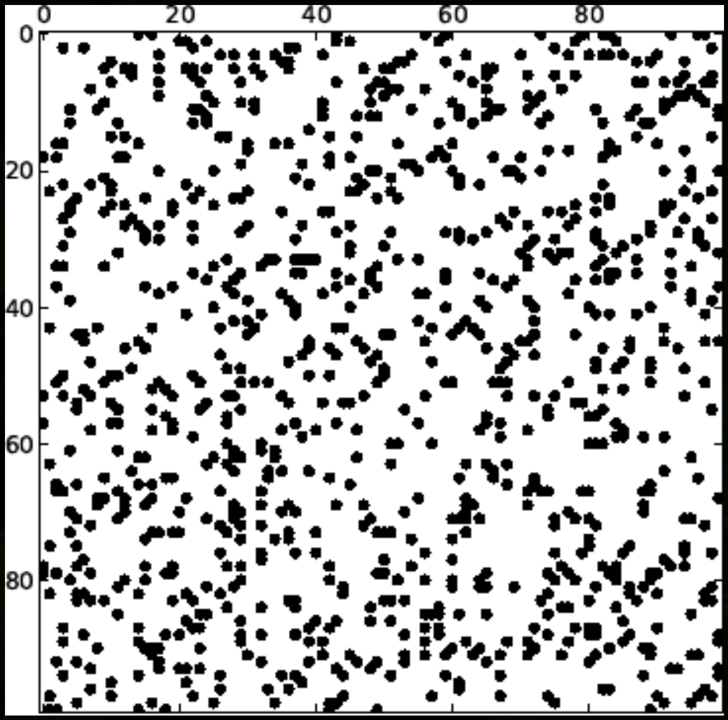
Matrix

Format	`float`	`double`
COO	12.00	16.00
CSR	8.62	12.62
DIA	164.11	328.22
ELL	11.06	16.60
HYB	9.00	13.44

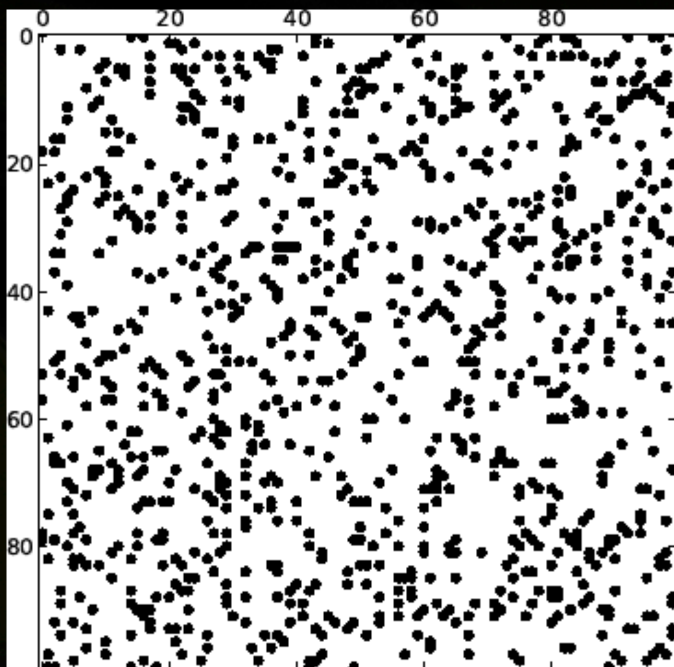
Bytes per Nonzero Entry

Storage Format Comparison

- **Random matrix**



Storage Format Comparison



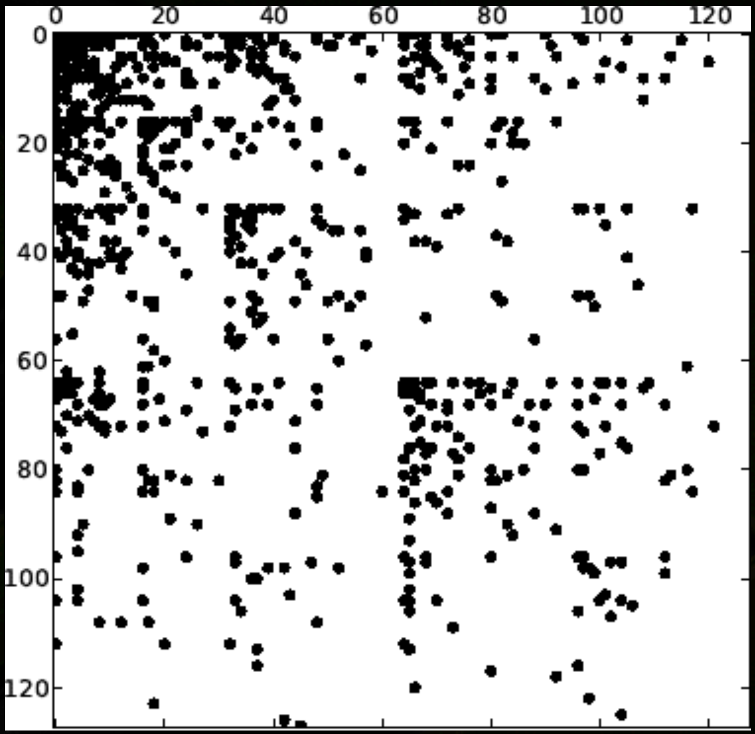
Format	Float	Double
COO	12.00	16.00
CSR	8.42	12.42
DIA	76.83	153.65
ELL	14.20	21.29
HYB	9.60	14.20

Matrix

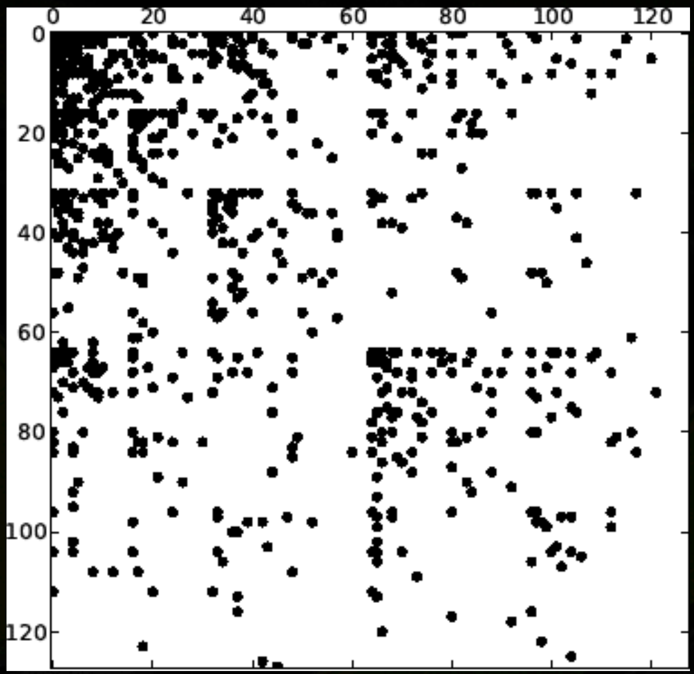
Bytes per Nonzero Entry

Storage Format Comparison

- **Power-Law Graph**



Storage Format Comparison



Matrix

Format	Float	Double
COO	12.00	16.00
CSR	8.74	12.73
DIA	118.83	237.66
ELL	49.88	74.82
HYB	13.50	19.46

Bytes per Nonzero Entry

Summary

- Iterative Solvers
  - Measure accuracy through residual (b – A*x)
  - Stop when ||b – A*x|| is small
- Sparse Formats
  - Numerous Options
  - Best format depends on matrix structure
- Next Lesson
  - Implementing Sparse Matrix-Vector Multiplication

References

Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors

Nathan Bell and Michael Garland

Proceedings of Supercomputing '09

Efficient Sparse Matrix-Vector Multiplication on CUDA

Nathan Bell and Michael Garland

NVIDIA Technical Report NVR-2008-004, December 2008

Iterative Methods for Sparse Linear Systems

Yousef Saad

http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf (online)