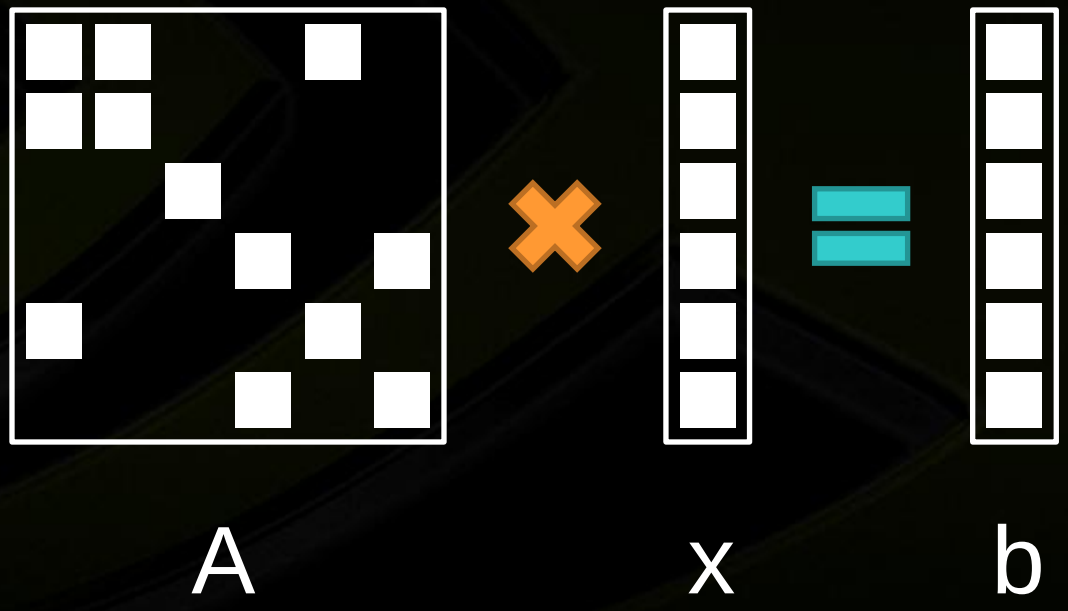
ALGEBRAIC MULTIGRID

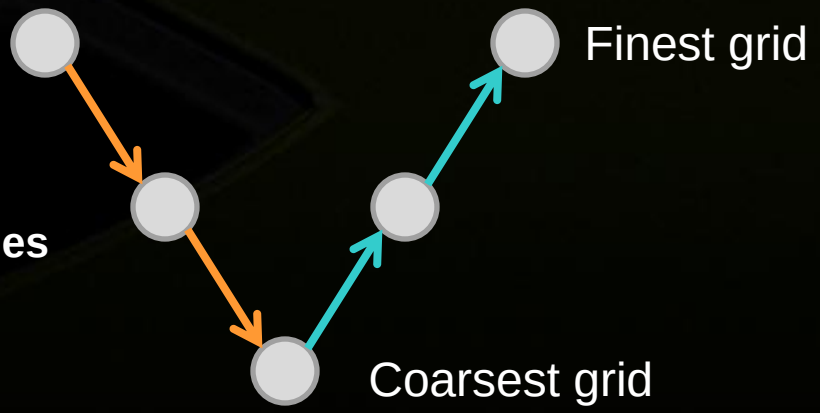
Objective

- **Solve (certain) sparse linear systems *very quickly***
  - **Optimal complexity O(N)**



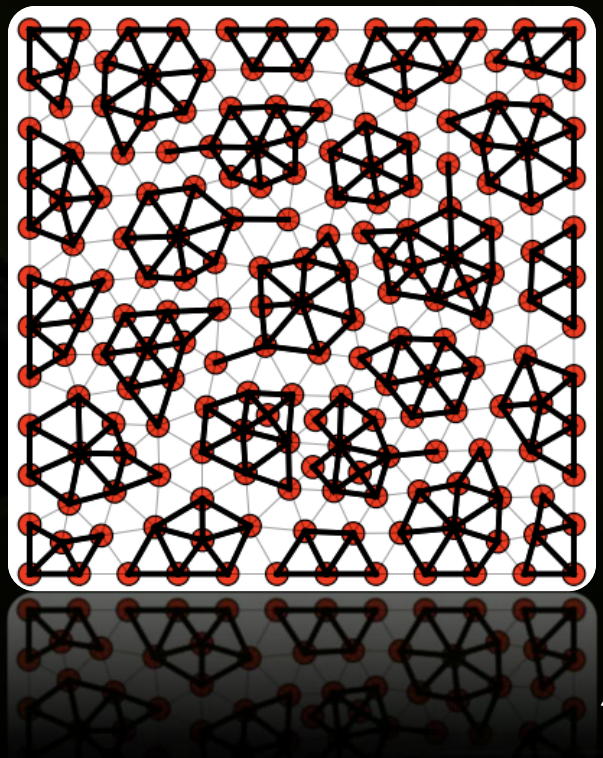
Multigrid in a Nutshell

- **Setup Phase**
  - Construct a hierarchy of grids
  - Sequence of coarser versions of the problem
- **Cycling/Solve Phase**
  - Reduce high-frequency errors with relaxation
  - Restrict remaining low-frequency errors to coarse grid
  - Interpolate coarse-grid solution back to fine grid
- **Applied recursively**
  - Eliminates all error modes
  - Results in V-cycle
  - Converges to solution in multiple cycles



Algebraic Multigrid (AMG)

- Constructs “grids” directly from sparse matrix
  - Hierarchy of sparse matrices
  - Requires no geometric knowledge
- Aggregation-based AMG
  - Coarsens clusters of nodes
  - Works for unstructured meshes

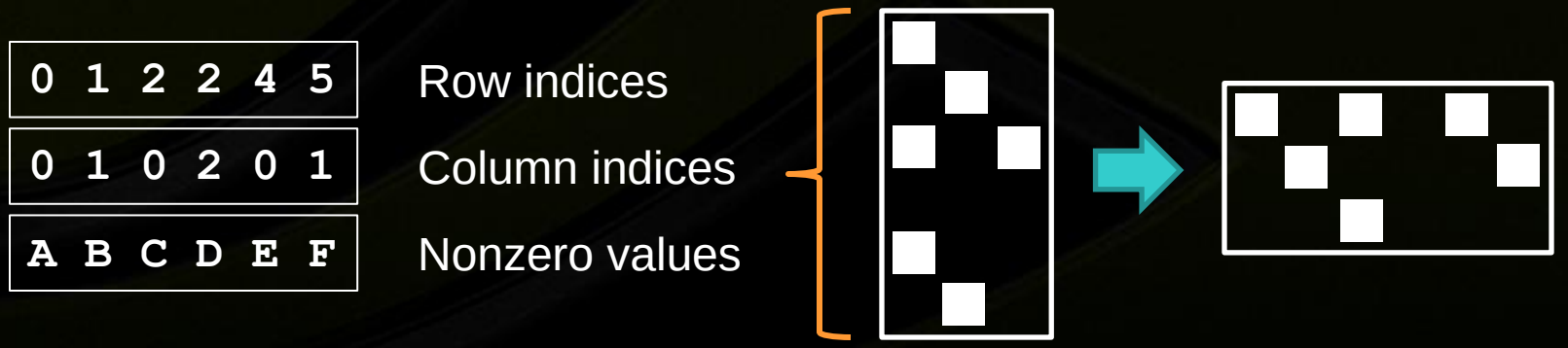


AMG on the GPU

- **Implemented both Setup and Cycling on GPU**
  - Need to identify *fine-grained* parallelism everywhere
  - Distill complex algorithms into *parallel primitives*
- **Setup Phase**
  - Sparse matrix-matrix multiplication (C = A * B)
  - Sparse matrix transpose
  - Matrix format conversions, parallel aggregation, etc.
- **Cycling Phase**
  - Sparse matrix-vector multiplication (y = A * x)
  - Level 1 BLAS operations (e.g. dot product)

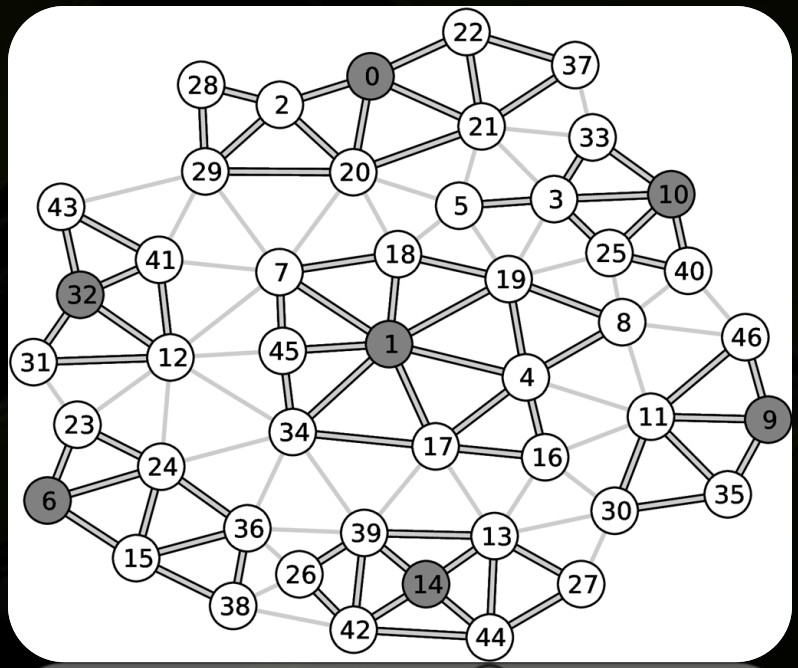
Example: Transpose

- Matrix in coordinate format (COO)
- Sort rows and values by column index
- Implemented with `thrust::sort_by_key`



Example: Aggregation

- Compute MIS(2) in parallel with extension of Luby's method
- Create aggregates around each MIS(2) node
- Close analog of standard greedy aggregation scheme



Performance Study

- Eight matrix example
  - Isotropic Poisson Problems
  - Structured and Unstructured
- GPU System
  - Tesla C2050 GPU
  - CUDA 4.0
  - Thrust v1.4
  - Cusp v0.2
- CPU System
  - Intel Core i7 950 CPU
  - MKL v10.3
- Reference Solver
  - Trilinos/ML v5.0

Matrix	Notes	Rows	Nonzeros
1a	2D FD, 5-point	~1M	~5M
1b	2D FE, 9-point	~1M	~9M
2a	3D FD, 7-point	~1M	~7M
2b	3D FE, 27-point	~1M	~27M
3a	2D FE, h=0.03	~500K	~4M
3b	2D FE, h=0.02	~1M	~8M
3c	2D FE, h=0.015	~2M	~15M
4	3D FE, h=0.15	~1M	~17M

Individual Components

Algorithm	Speedup
Dot Product	6.62x
Vector Addition	6.35x
Sparse Transpose*	2.90x
Sparse Matrix-Vector Multiply*	6.00x
Sparse Matrix-Matrix Multiply*	1.67x

*Average speedup across eight example matrices

Setup Phase

Matrix	Type	CPU	GPU	Speedup	Trilinos/ML
1a	2D FD	892 ms	518 ms	1.72x	2040 ms
1b	2D FE	1133 ms	649 ms	1.75x	2298 ms
2a	3D FD	1639 ms	944 ms	1.74x	2906 ms
2b	3D FE	2845 ms	2124 ms	1.34x	4420 ms
3a	2D FE	657 ms	335 ms	1.96x	1324 ms
3b	2D FE	1484 ms	648 ms	2.29x	2785 ms
3c	2D FE	2901 ms	1151 ms	2.52x	5236 ms
4	3D FE	3157 ms	1726 ms	1.83x	4967 ms

Cycling Phase

Matrix	Type	CPU	GPU	Speedup	Trilinos/ML
1a	2D FD	1221 ms (20)	423 ms (51)	**7.66x**	14,190 ms (33)
1b	2D FE	1097 ms (16)	461 ms (46)	**7.52x**	10,590 ms (22)
2a	3D FD	1760 ms (23)	295 ms (27)	**6.76x**	14,800 ms (31)
2b	3D FE	1683 ms (14)	482 ms (24)	**5.98x**	13,840 ms (20)
3a	2D FE	1534 ms (42)	286 ms (49)	**5.40x**	14,020 ms (53)
3b	2D FE	3704 ms (47)	633 ms (54)	**5.77x**	34,410 ms (68)
3c	2D FE	7804 ms (53)	1424 ms (65)	**5.75x**	44,530 ms (65)
4	3D FE	4369 ms (43)	1498 ms (50)	**2.96x**	28,380 ms (47)

Notes
- Time to solve Ax=b to 1e-10 relative tolerance
- AMG as a preconditioner to CG solver
- Iteration counts shown in parentheses
- Reporting per-iteration speedup

Summary

- **Fully-Parallelized AMG on GPU**
  - **1.89x Speedup in Setup Phase (average)**
  - **5.89x Speedup in Cycling Phase (average)**

- **References**

"Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods"
Nathan Bell (NVIDIA), Steven Dalton (UIUC), Luke Olson (UIUC),
NVIDIA Technical Report NVR-2011-002, June 2011
http://research.nvidia.com/publications

Cusp Library
http://cusp-library.googlecode.com

Thrust Library
http://thrust.googlecode.com