# GPU Computing with CUDA
# Lecture 7 - CUDA Libraries - Cusp

*Christopher Cooper*
*Boston University*

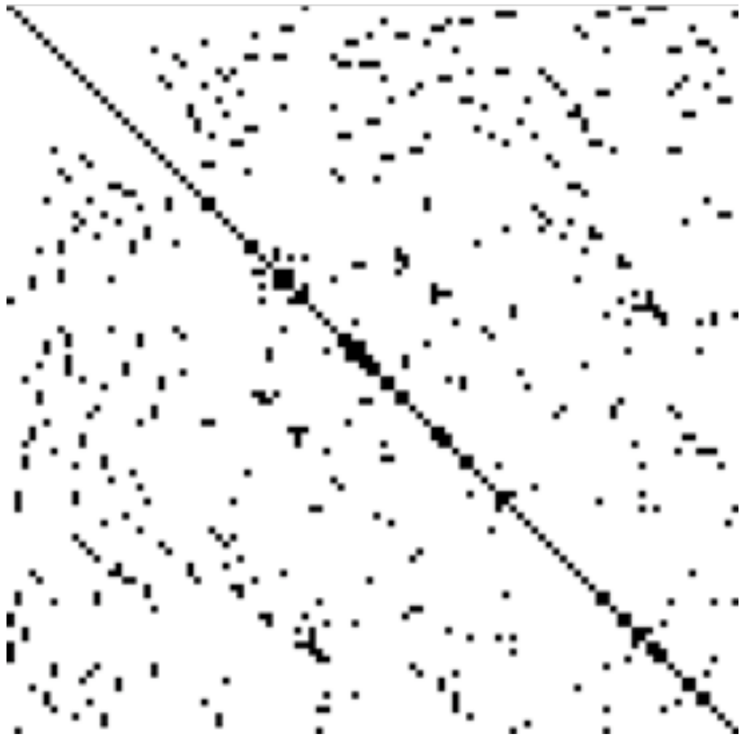
*August, 2011*
*UTFSM, Valparaíso, Chile*

# Outline of lecture

- Overview:
  - Sparse matrices
  - Preconditioners
  - Solvers
- Cusp - A sparse matrix library (slides by Nathan Bell - NVIDIA)
- Example of sparse matrix: matrix representation of Poisson problem

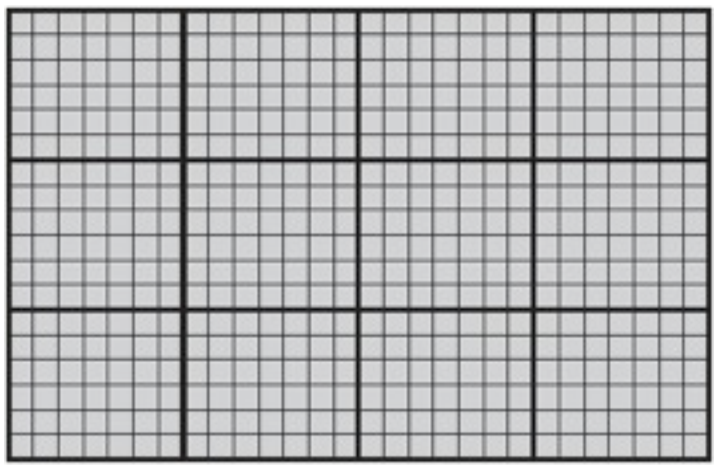
# Sparse matrices

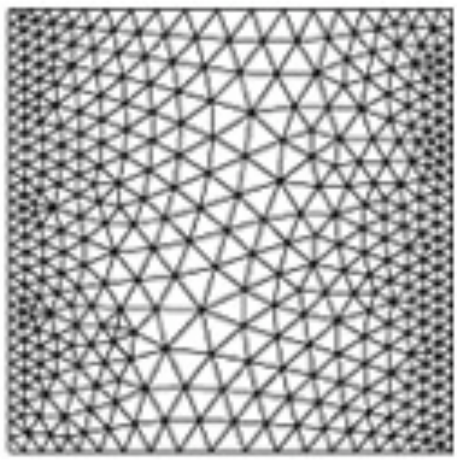
- Matrix mostly filled with zeroes
- Efficient storage and computation
  - Avoid dense matrix storage
  - Specialized data structures and algorithms



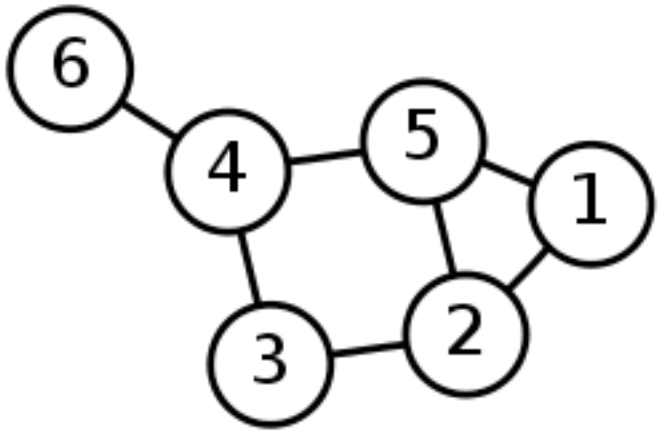
# Sparse matrices

- Mesh discretized domains for PDE solving





- Graph problems



# Sparse matrices

- Efficient storage formats of sparse matrices
  - Coordinate
  - Diagonal
  - Compressed Sparse Row (CSR)
  - ELLPACK
  - Hybrid

# Sparse matrices - COO

- Coordinate (COO)

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

# Sparse matrices - CSR

- Compressed Sparse Row (CSR)

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

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

# Sparse matrices - ELL

- ELLPACK (ELL)

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

values

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

# rows

# entries per row

column indices

| | | |
|---|---|---|
| 0 | 1 | * |
| 1 | 2 | * |
| 0 | 2 | 3 |
| 1 | 3 | * |

# Sparse matrices - DIA

- Diagonal (DIA)

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

values

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

# rows

# diagonals

# Sparse matrices - HYB

- Hybrid (HYB) ELL+COO
  - Cusp only!

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

**ELL**

values (# rows)

| | |
|---|---|
| 1 | 7 |
| 2 | 8 |
| 5 | 3 |
| 6 | 4 |

column indices

| | |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 0 | 2 |
| 1 | 3 |

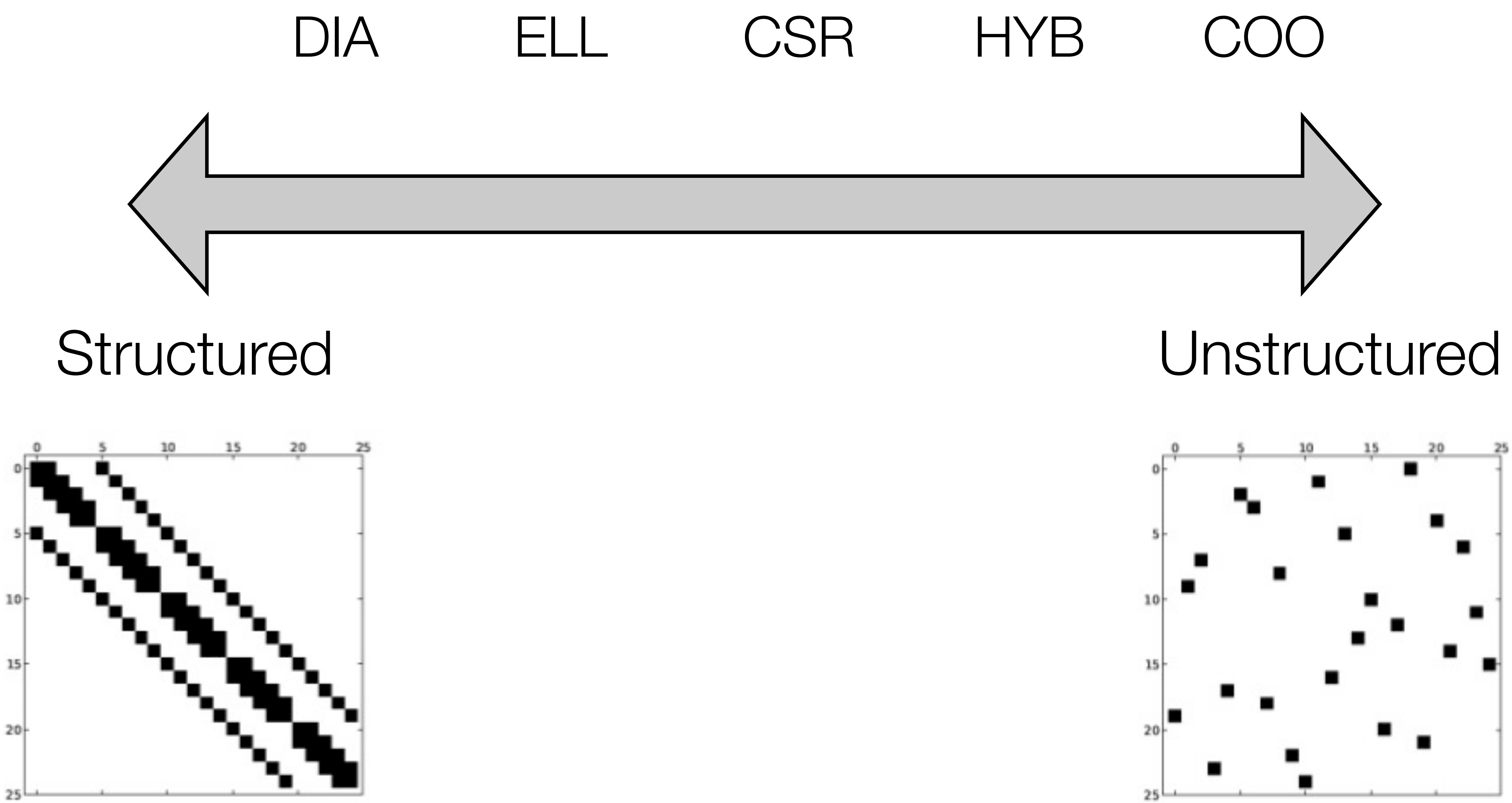
**COO**

2

3

9

# Sparse matrices



# Solvers

$$A\mathbf{x} = \mathbf{b}$$

- Direct Methods
  - Produces the right solution (within machine precision)
  - Mainly LU factorization procedures
- Iterative methods
  - Looks for an approximate solution

# Solvers

- Iterative methods
  - Converge to the right solution
- Use residual as criterion to stop iterations
  - residual = b - A*x
- Many methods
  - Jacobi
  - Gauss Seidel
  - Conjugate Gradient (CG)
  - Generalized Minimum Residual (GMRES)

# Solvers - Iterative solvers

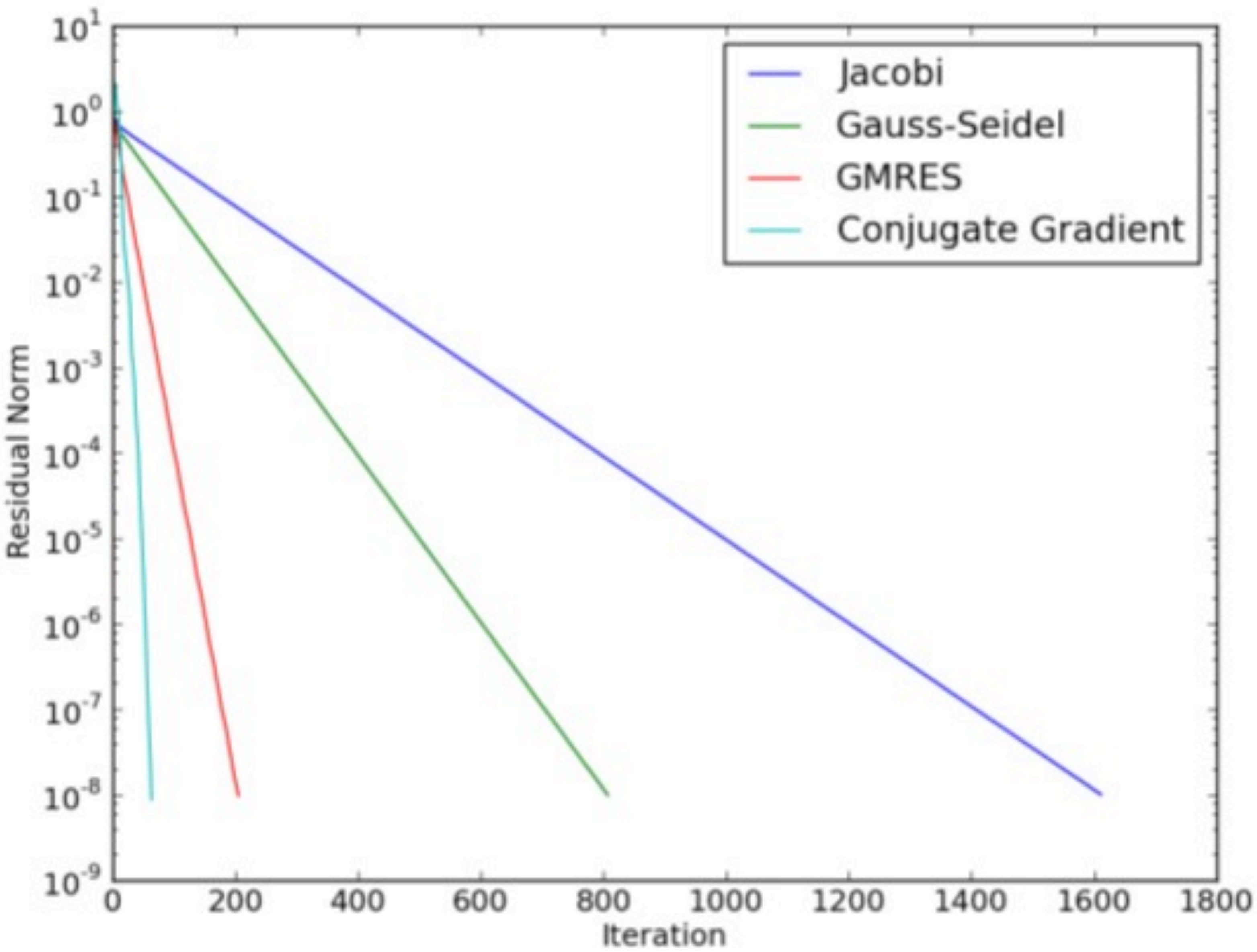
- Stationary Methods
  - Matrix splitting
  - Example: GS, Jac

$$A = M - N$$

$$\begin{aligned} M\mathbf{x}_{k+1} &= N\mathbf{x}_k + \mathbf{b} \\ \mathbf{x}_{k+1} &= M^{-1}(N\mathbf{x}_k - \mathbf{b}) \end{aligned}$$

- Krylov methods
  - Solution is taken from the Krylov subspace
  - Minimize residual
  - Example: CG, GMRES, BiCG-stab

# Solvers - Iterative solvers



# Preconditioners

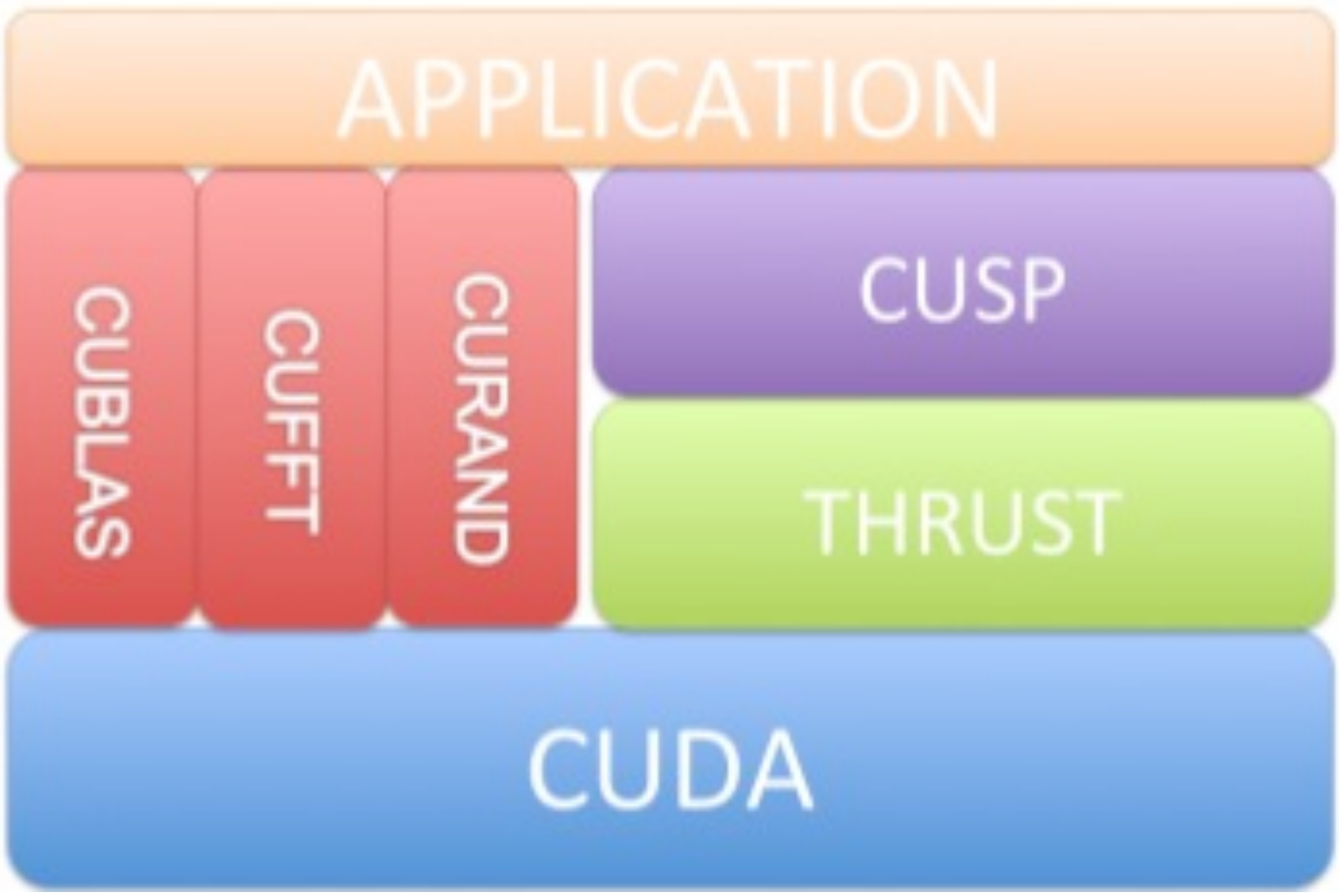
- Helps the solver to get faster to the solution
- It's an approximate of the inverse

$$M \approx A^{-1}$$
$$MA\mathbf{x} = M\mathbf{b}$$

- Example: Diagonal preconditioner

# CUDA Libraries

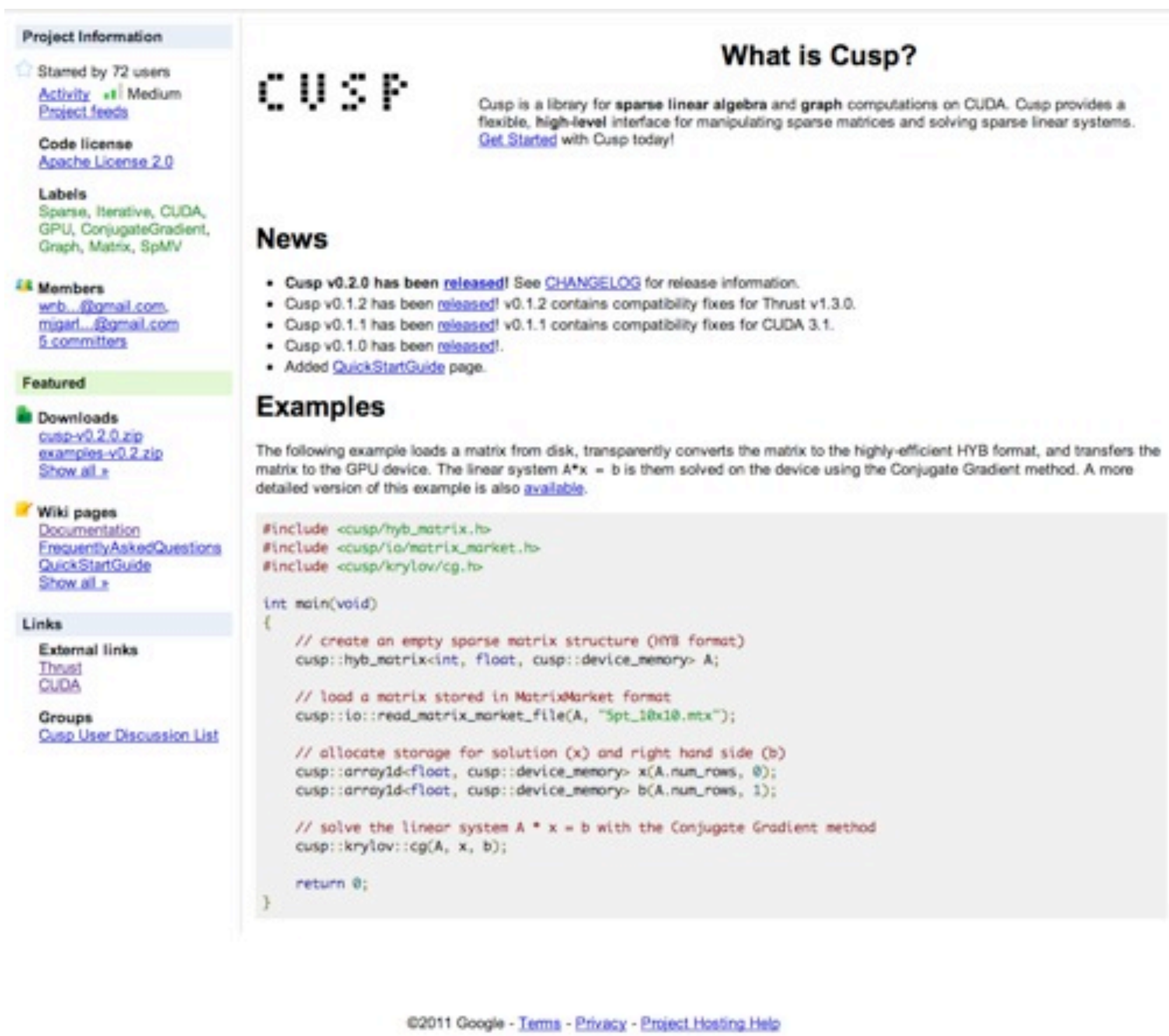
- NVIDIA has developed several libraries to abstract the user from CUDA




Bell, Dalton, Olson. Towards AMG on GPU

# Cusp

- Library for sparse linear algebra
- http://code.google.com/p/cusp-library/



# Cusp - Containers

- Matrices may be contained in host or device

```
cusp::coo_matrix<int, float, cusp::device_memory> A;
```

- Sparse matrix containers
  - COO
  - CSR
  - DIA
  - ELL
  - HYB

# Cusp - Containers

```
#include <cusp/coo_matrix.h>

int main(void)
{
    // allocate storage for (4,3) matrix with 6 nonzeros
    cusp::coo_matrix<int, float, cusp::host_memory> A(4,3,6);

    // initialize matrix entries on host
    A.row_indices[0] = 0; A.column_indices[0] = 0; A.values[0] = 10.0f;
    A.row_indices[1] = 0; A.column_indices[1] = 2; A.values[1] = 20.0f;
    A.row_indices[2] = 2; A.column_indices[2] = 2; A.values[2] = 30.0f;
    A.row_indices[3] = 3; A.column_indices[3] = 0; A.values[3] = 40.0f;
    A.row_indices[4] = 3; A.column_indices[4] = 1; A.values[4] = 50.0f;
    A.row_indices[5] = 3; A.column_indices[5] = 2; A.values[5] = 60.0f;

    // convert COO->CSR on the host and transfer to the device
    cusp::csr_matrix<int, float, cusp::device_memory> B = A;

    // convert CSR->ELL on the device
    cusp::ell_matrix<int, float, cusp::device_memory> C;
    cusp::convert(B, C);

    return 0;
}
```

# Cusp - Containers

- Dense containers: `array1d, array2d`

```
#include <cusp/array1d.h>
#include <cusp/array2d.h>
int main(void)
{
    // allocate storage for (4,3) matrix filled with zeros
    cusp::array2d<float, cusp::host_memory, cusp::column_major> B(4, 3, 0.0f);
    // set array2d entries on host
    B(0,0) = 10;
    B(0,2) = 20;
    B(2,2) = 30;
    B(3,0) = 40;
    B(3,1) = 50;
    B(3,2) = 60;
    // B now represents the following matrix, stored in column-major order
    //    [10  0 20]
    //    [ 0  0  0]
    //    [ 0  0 30]
    //    [40 50 60]
    return 0;
}
```

# Cusp - Algorithms

- Cusp comes with BLAS (Basic Linear Algebra Library)

```
#include <cusp/array1d.h>
#include <cusp/blas.h>

int main(void)
{
    size_t N = 15;

    // allocate vectors
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> y(N);

    // initialize vectors
    ...

    // compute vector 2-norm ||x||
    float x_norm = cusp::blas::nrm2(x);

    // compute y = y + 3 * x
    cusp::blas::axpy(x, y, 3.0f);
    return 0;
}
```

# Cusp - Algorithms

```
#include <cusp/coo_matrix.h>
#include <cusp/array1d.h>
#include <cusp/multiply.h>

int main(void)
{
    size_t M   = 10;
    size_t N   = 15;
    size_t NNZ = 43;

    // allocate 10x15 COO matrix and vectors
    cusp::coo_matrix<int, float, cusp::device_memory> A(M, N, NNZ);
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> y(M);

    // initialize A and x
    ...

    // compute matrix-vector product y = A * x
    cusp::multiply(A, x, y);

    return 0;
}
```

# Cusp - Algorithms

```
#include <cusp/transpose.h>
#include <cusp/array2d.h>
#include <cusp/print.h>

int main(void)
{
    // initialize a 2x3 matrix
    cusp::array2d<float, cusp::host_memory> A(2,3);
    A(0,0) = 10;  A(0,1) = 20;  A(0,2) = 30;
    A(1,0) = 40;  A(1,1) = 50;  A(1,2) = 60;

    // print A
    cusp::print(A);

    // compute the transpose
    cusp::array2d<float, cusp::host_memory> At;
    cusp::transpose(A, At);

    // print A^T
    cusp::print(At);

    return 0;
}
```

# Cusp - Solvers

- Cusp supports: CG, GMRES, BiCG-stab, Jacobi relaxation

```
#include <cusp/coo_matrix.h>
#include <cusp/array1d.h>
#include <cusp/krylov/cg.h>

int main(void)
{
    size_t N   = 15;
    size_t NNZ = 43;

    // allocate 10x15 COO matrix and vectors
    cusp::coo_matrix<int, float, cusp::device_memory> A(N, N, NNZ);
    cusp::array1d<float, cusp::device_memory> x(N);
    cusp::array1d<float, cusp::device_memory> b(N);

    // initialize A and b
    ...

    // solve A * x = b to default tolerance with CG
    cusp::krylov::cg(A, x, b);
    return 0;
}
```

# Cusp - Monitors

- Monitors give you information about your solve

```
// set stopping criteria of default_monitor:
//  iteration_limit    = 100
//  relative_tolerance = 1e-6
cusp::default_monitor<float> monitor(b, 100, 1e-6);
// solve A * x = b to specified tolerance
cusp::krylov::cg(A, x, b, monitor);
```

```
// set stopping criteria of verbose_monitor:
//  iteration_limit    = 100
//  relative_tolerance = 1e-6
cusp::verbose_monitor<float> monitor(b, 100, 1e-6);
// solve A * x = b to specified tolerance
cusp::krylov::cg(A, x, b, monitor);
```

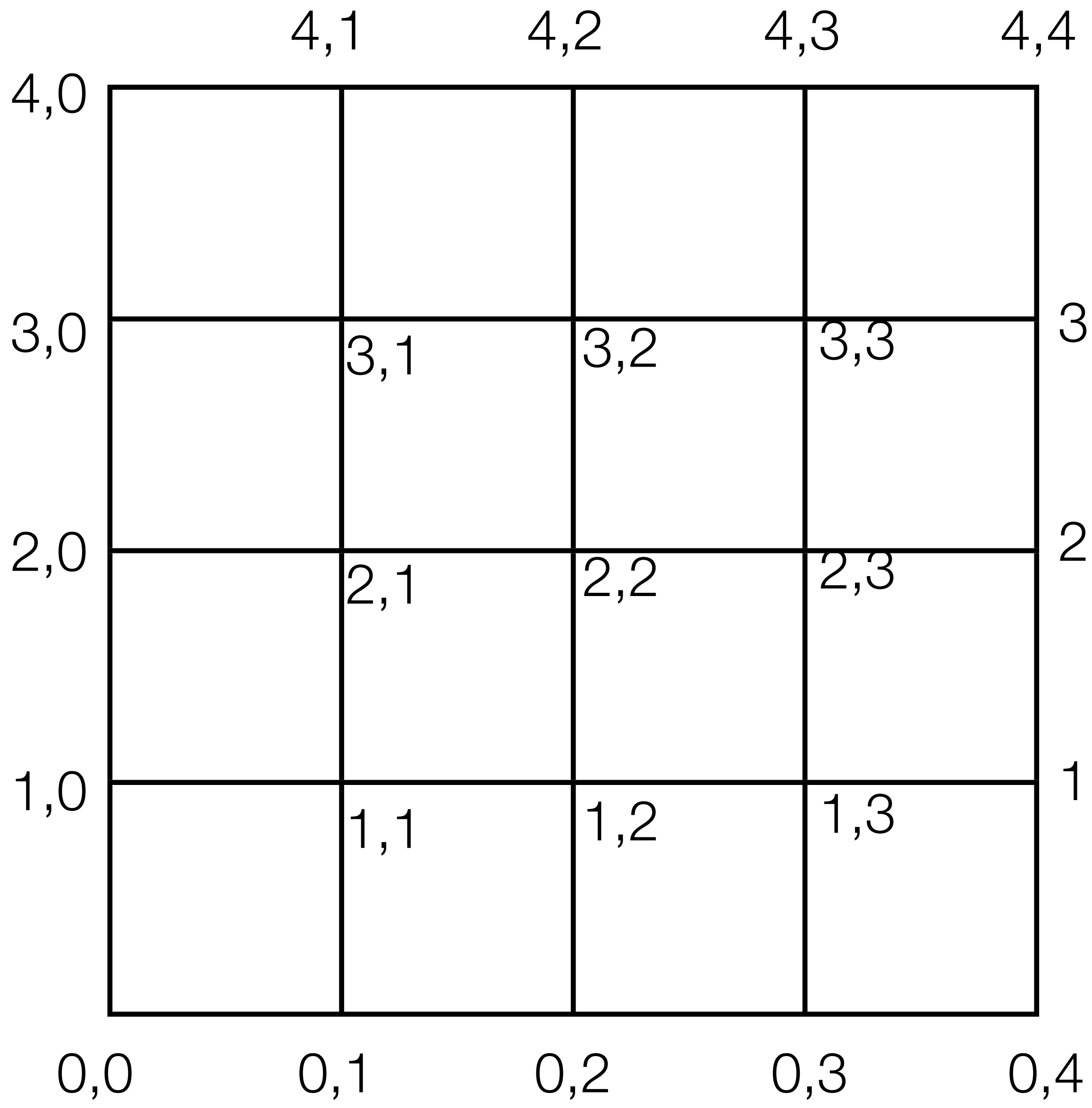
# Cusp - Preconditioners

```
#include <cusp/krylov/cg.h>
#include <cusp/precond/smoothed_aggregation.h>
...

// set stopping criteria
//  iteration_limit = 100
//  relative_tolerance = 1e-6
cusp::default_monitor<float> monitor(b, 100, 1e-6);

// setup preconditioner
cusp::precond::smoothed_aggregation<int, float,
cusp::device_memory> M(A);

// solve A * x = b to default tolerance with preconditioned CG
cusp::krylov::cg(A, x, b, monitor, M);
```

# Sparse matrix example

- Solving a Poisson problem in 2D on a 5x5 matrix, Dirichlet BCs

$$\nabla^2 u = g$$



# Sparse matrix example

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta y^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta x^2} = g_{i,j}$$

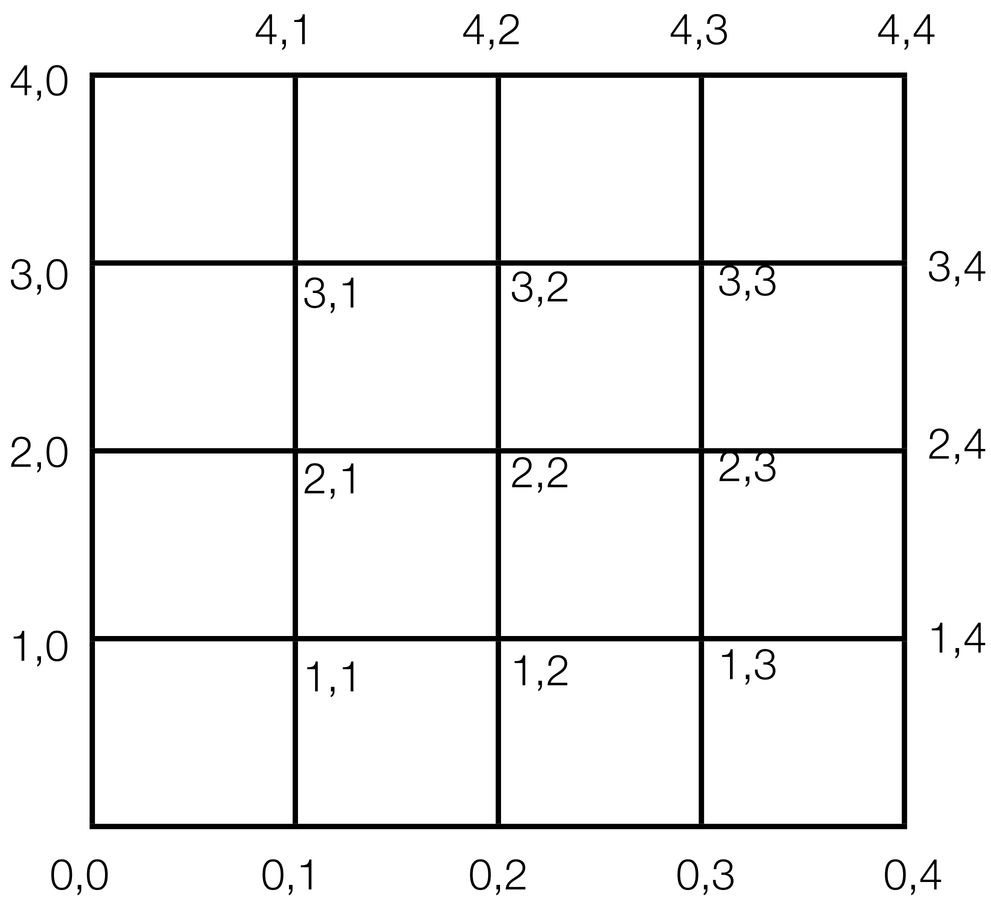
Assume Δx=Δy=h

$$\frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}}{h^2} = g_{i,j}$$

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = g_{i,j}h^2$$

# Sparse matrix example

- Just need to calculate the internal points!




$$u_{0,1} + u_{2,1} + u_{1,0} + u_{1,2} - 4u_{1,1} = g_{1,1}h^2$$

$$u_{0,2} + u_{2,2} + u_{1,1} + u_{1,3} - 4u_{1,2} = g_{1,2}h^2$$

$$u_{0,3} + u_{2,3} + u_{1,2} + u_{1,4} - 4u_{1,3} = g_{1,3}h^2$$

$$u_{1,1} + u_{3,1} + u_{2,0} + u_{2,2} - 4u_{2,1} = g_{2,1}h^2$$

$$u_{1,2} + u_{3,2} + u_{2,1} + u_{2,3} - 4u_{2,2} = g_{2,2}h^2$$

$$u_{1,3} + u_{3,3} + u_{2,2} + u_{2,4} - 4u_{2,3} = g_{2,3}h^2$$

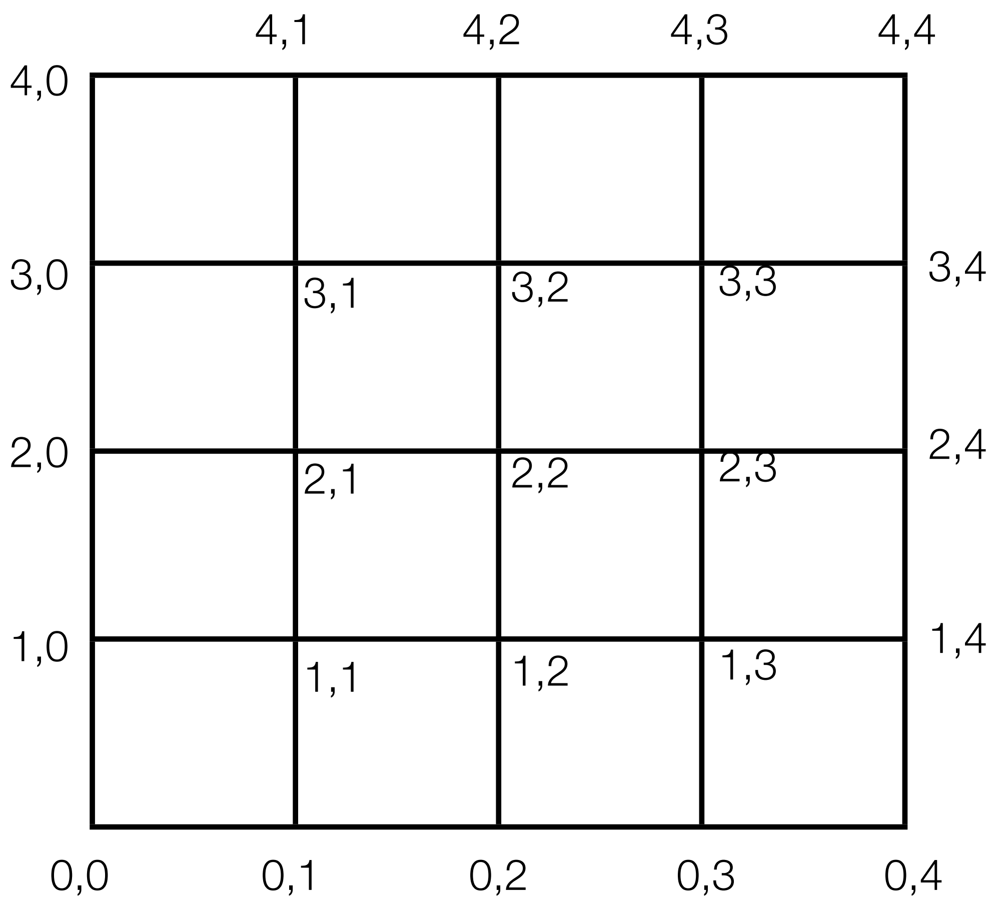
$$u_{2,1} + u_{4,1} + u_{3,0} + u_{3,2} - 4u_{3,1} = g_{3,1}h^2$$

$$u_{2,2} + u_{4,2} + u_{3,1} + u_{3,3} - 4u_{3,2} = g_{3,2}h^2$$

$$u_{2,3} + u_{4,3} + u_{3,2} + u_{3,4} - 4u_{3,3} = g_{3,3}h^2$$

# Sparse matrix example

- Just need to calculate the internal points!

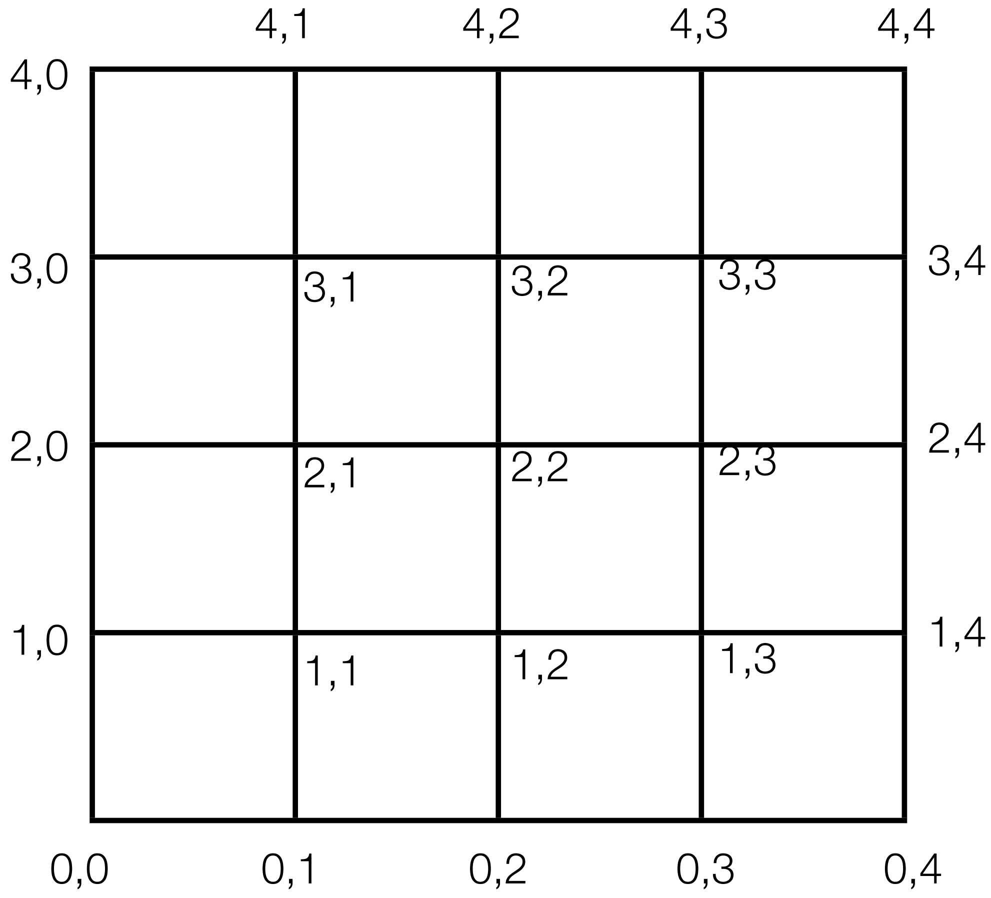



$$u_{0,1} + u_{2,1} + u_{1,0} + u_{1,2} - 4u_{1,1} = g_{1,1}h^2$$
$$u_{0,2} + u_{2,2} + u_{1,1} + u_{1,3} - 4u_{1,2} = g_{1,2}h^2$$
$$u_{0,3} + u_{2,3} + u_{1,2} + u_{1,4} - 4u_{1,3} = g_{1,3}h^2$$
$$u_{1,1} + u_{3,1} + u_{2,0} + u_{2,2} - 4u_{2,1} = g_{2,1}h^2$$
$$u_{1,2} + u_{3,2} + u_{2,1} + u_{2,3} - 4u_{2,2} = g_{2,2}h^2$$
$$u_{1,3} + u_{3,3} + u_{2,2} + u_{2,4} - 4u_{2,3} = g_{2,3}h^2$$
$$u_{2,1} + u_{4,1} + u_{3,0} + u_{3,2} - 4u_{3,1} = g_{3,1}h^2$$
$$u_{2,2} + u_{4,2} + u_{3,1} + u_{3,3} - 4u_{3,2} = g_{3,2}h^2$$
$$u_{2,3} + u_{4,3} + u_{3,2} + u_{3,4} - 4u_{3,3} = g_{3,3}h^2$$

Take them to RHS!

# Sparse matrix example

- Just need to calculate the internal points!




Diagonal

$$u_{0,1} + u_{2,1} + u_{1,0} + u_{1,2} - 4u_{1,1} = g_{1,1}h^2$$
$$u_{0,2} + u_{2,2} + u_{1,1} + u_{1,3} - 4u_{1,2} = g_{1,2}h^2$$
$$u_{0,3} + u_{2,3} + u_{1,2} + u_{1,4} - 4u_{1,3} = g_{1,3}h^2$$
$$u_{1,1} + u_{3,1} + u_{2,0} + u_{2,2} - 4u_{2,1} = g_{2,1}h^2$$
$$u_{1,2} + u_{3,2} + u_{2,1} + u_{2,3} - 4u_{2,2} = g_{2,2}h^2$$
$$u_{1,3} + u_{3,3} + u_{2,2} + u_{2,4} - 4u_{2,3} = g_{2,3}h^2$$
$$u_{2,1} + u_{4,1} + u_{3,0} + u_{3,2} - 4u_{3,1} = g_{3,1}h^2$$
$$u_{2,2} + u_{4,2} + u_{3,1} + u_{3,3} - 4u_{3,2} = g_{3,2}h^2$$
$$u_{2,3} + u_{4,3} + u_{3,2} + u_{3,4} - 4u_{3,3} = g_{3,3}h^2$$

Take them to RHS!

# Sparse matrix example

- We get a system of 9 unknowns and 9 equations
- Following a row major ordering and multiplying by -1

$$[A]\mathbf{u} = b$$

$$b = \begin{bmatrix} -h^2 g_{1,1} + u_{0,1} + u_{1,0} \\ -h^2 g_{1,2} + u_{0,2} \\ -h^2 g_{1,3} + u_{0,3} + u_{1,4} \\ -h^2 g_{2,1} + u_{2,0} \\ -h^2 g_{2,2} \\ -h^2 g_{2,3} + u_{2,4} \\ -h^2 g_{3,1} + u_{4,1} + u_{3,0} \\ -h^2 g_{3,2} + u_{4,2} \\ -h^2 g_{3,3} + u_{4,3} + u_{3,4} \end{bmatrix}.$$

# Sparse matrix example

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}$$

$$\mathbf{u} = \left[u_{1,1}, u_{1,2}, u_{1,3}, u_{2,1}, u_{2,2}, u_{2,3}, u_{3,1}, u_{3,2}, u_{3,3}\right]^T$$