AMATH 483/583
High Performance Scientific Computing

Lecture 9:
Strassen’s Algorithm
Sparse Matrix Computation

Xu Tony Liu, PhD

Paul G. Allen School of Computer Science & Engineering

University of Washington

Seattle, WA

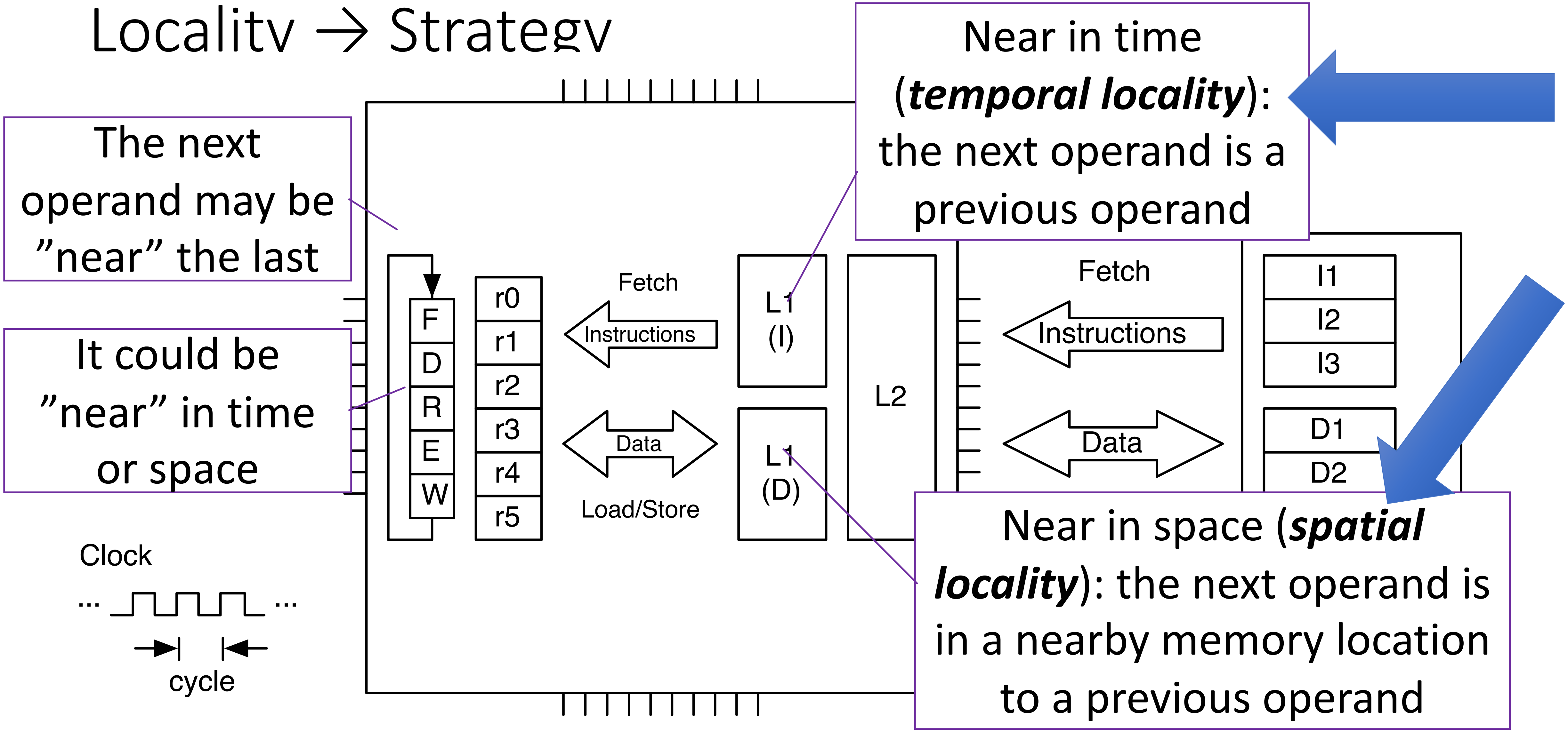
Announcements

- Mid Term out at noon 04/28/2022 due 11:59AM 05/05/2022

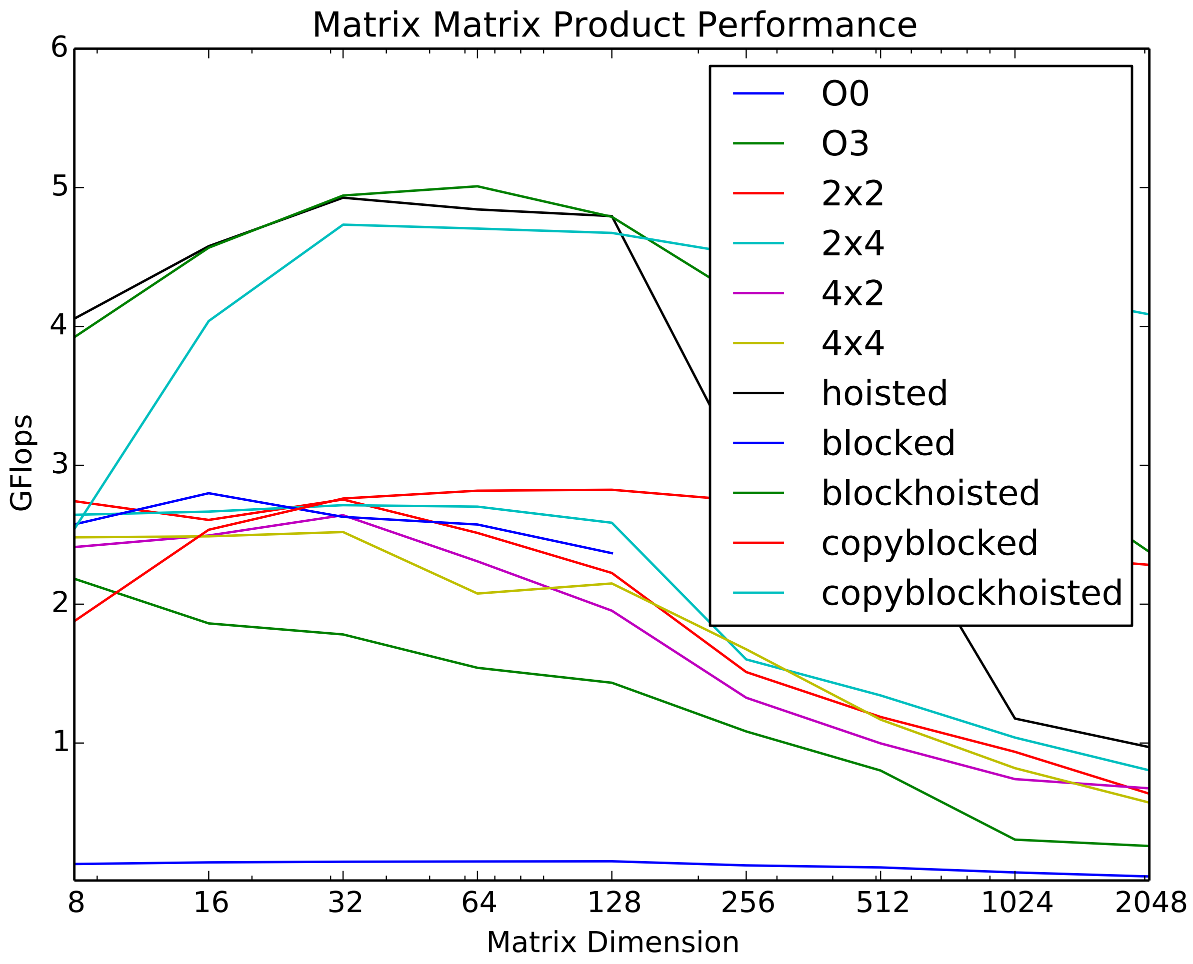
Overview

- Review
  - Locality and optimization strategies
  - SIMD, vectorization
  - Roofline model
  - Strassen’s Algorithm
- Sparsity
- Coordinate format (COO)
- Compressed sparse row (CSR)
- Multicore for HPC - an example

Locality → Strategy

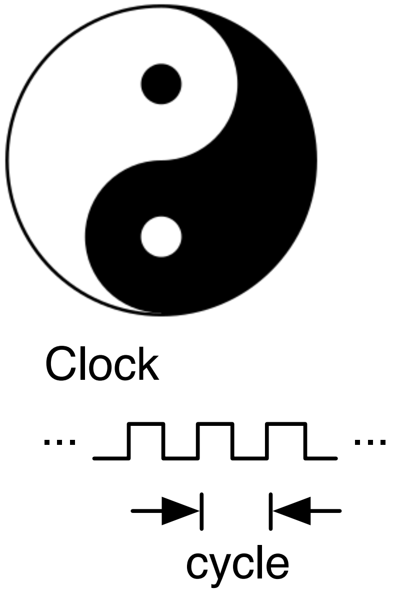


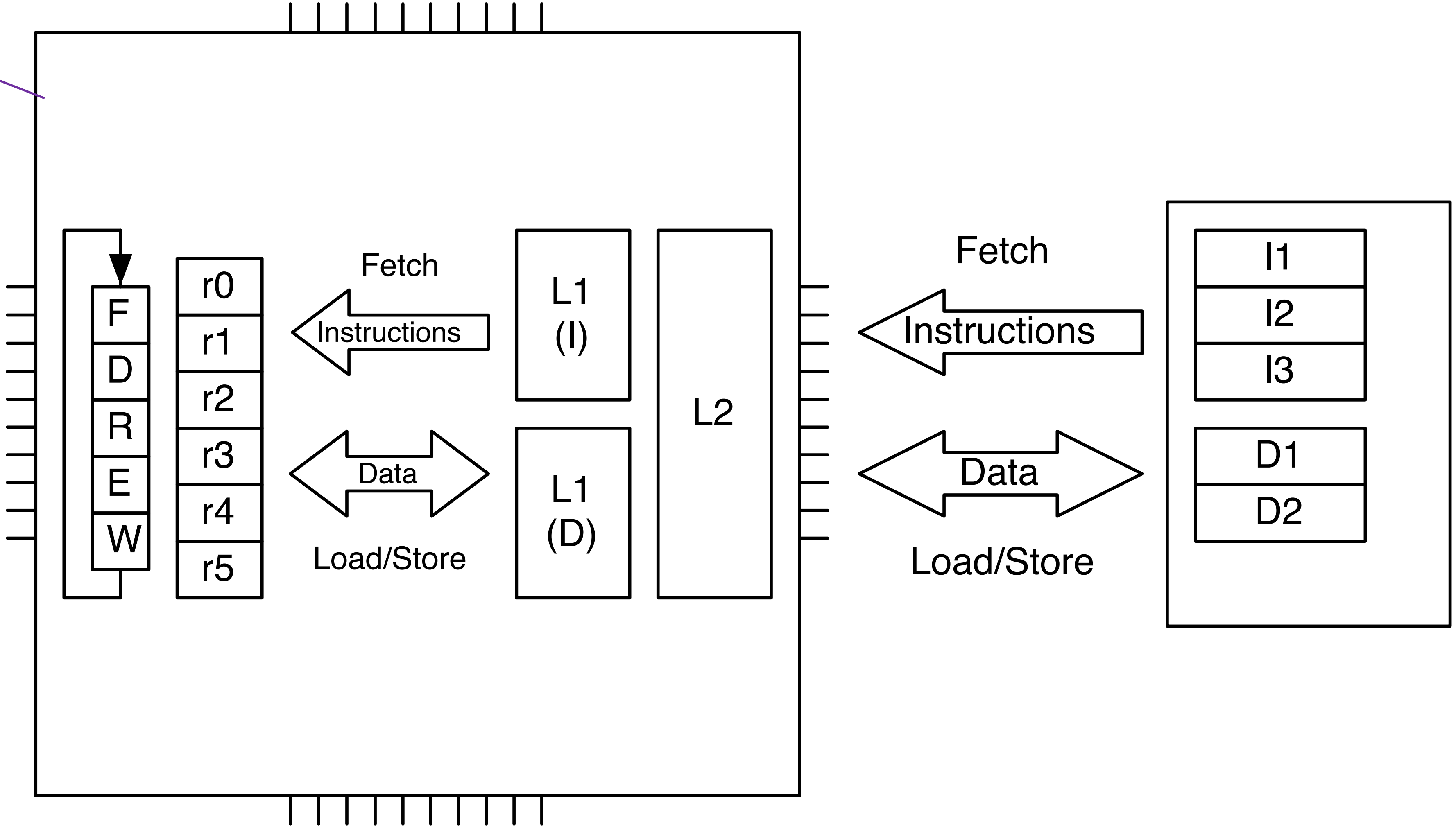
Blocking and Unrolling and Hoisting and Copying



What Else Can We Do for Performance

Exploit features that make hardware fast



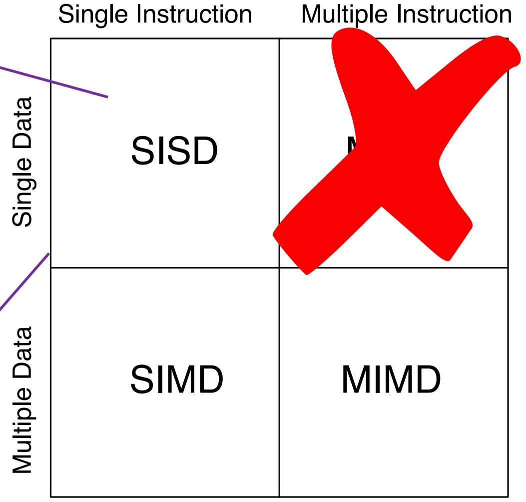


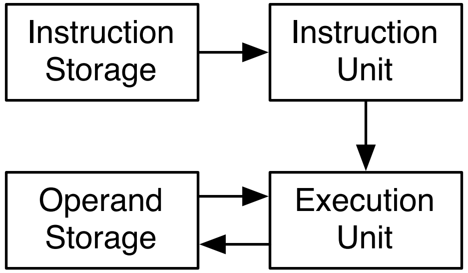
Flynn’s Taxonomy (Aside)

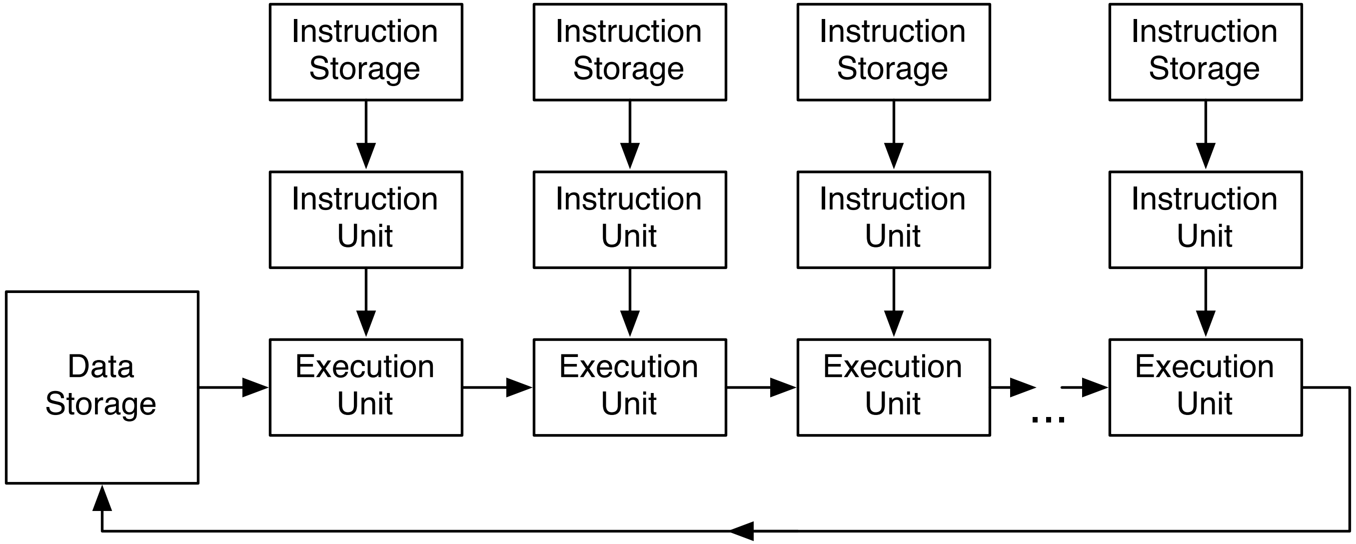
Anyone in HPC must know Flynn’s taxonomy

- **Classic** classification of parallel architectures (Michael Flynn, 1966)

Plain old sequential

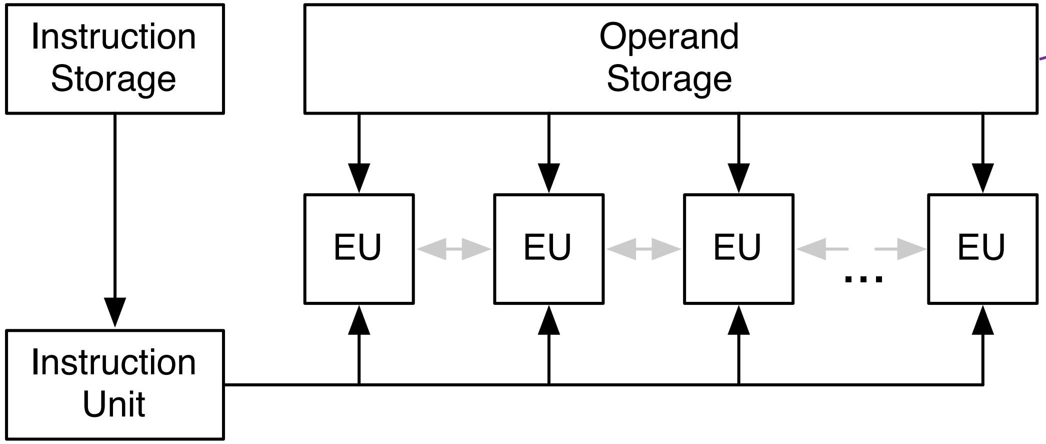






Based on multiplicity of instruction streams, data storage

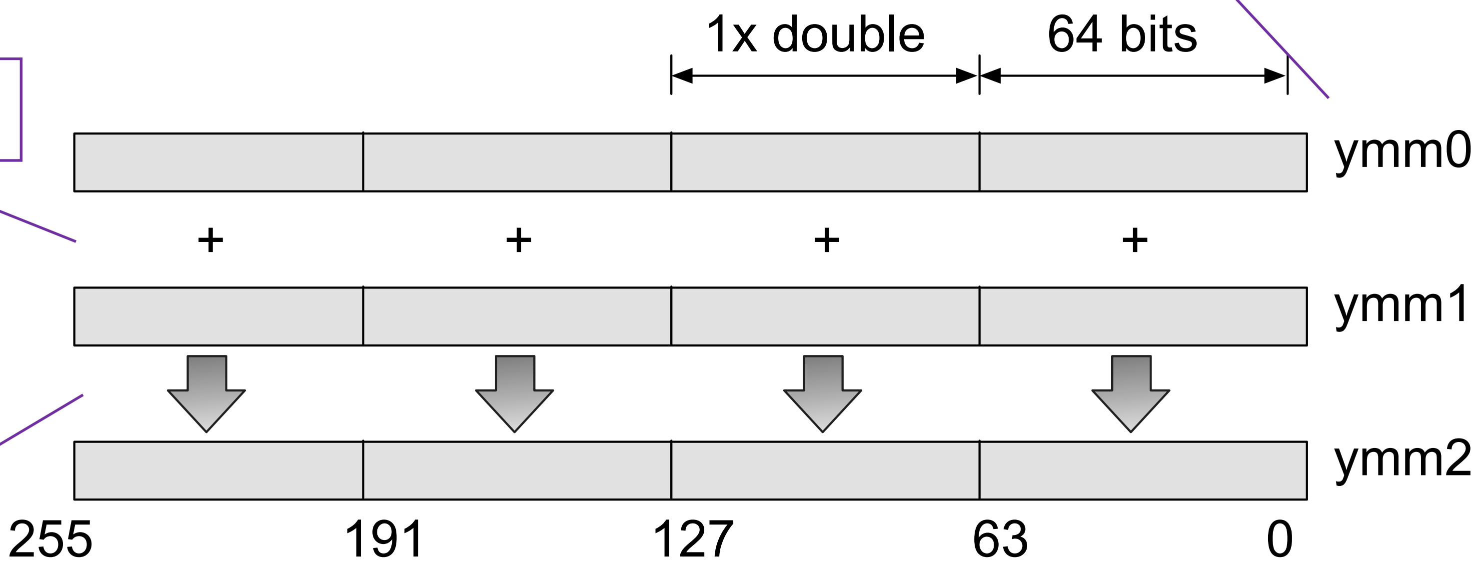
SIMD in SSE/AVX



Flynn's original conceptual model

ymm are 256 bit registers

```
vfadd231pd  %ymm0, %ymm1, %ymm2
```

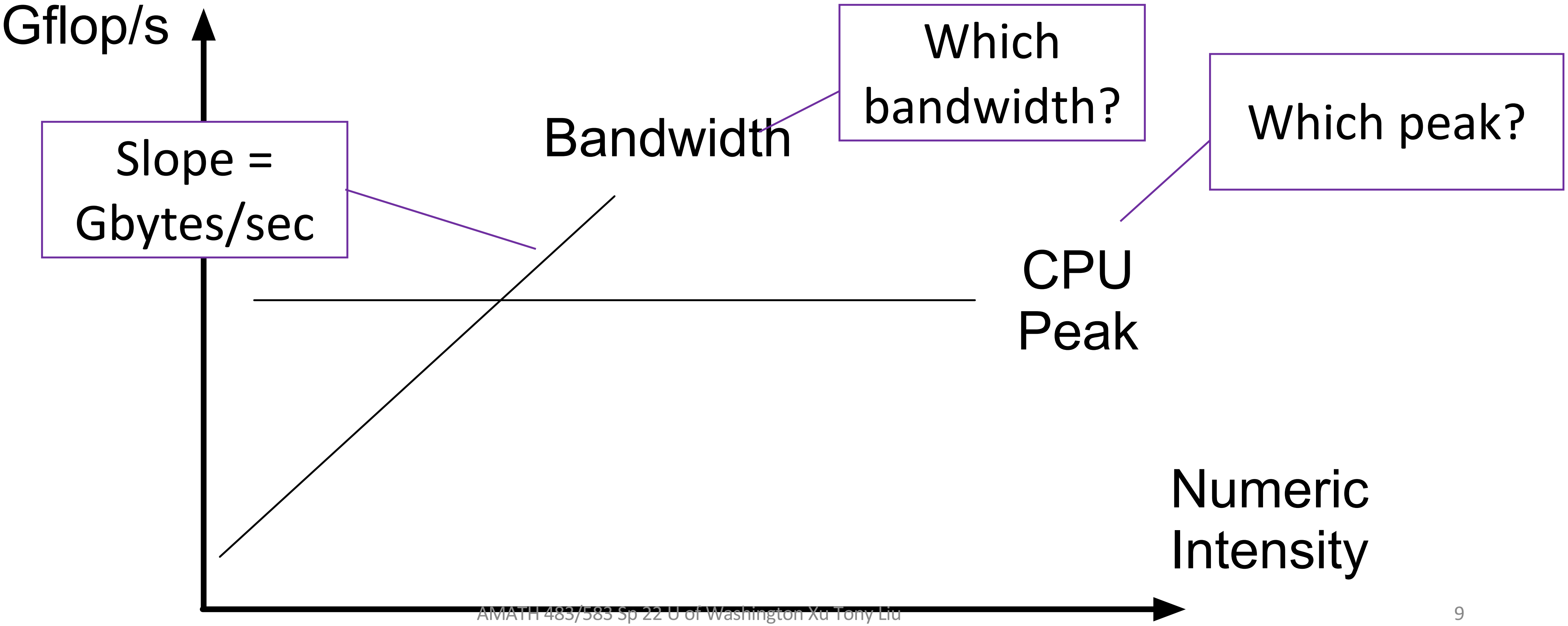


One machine instruction

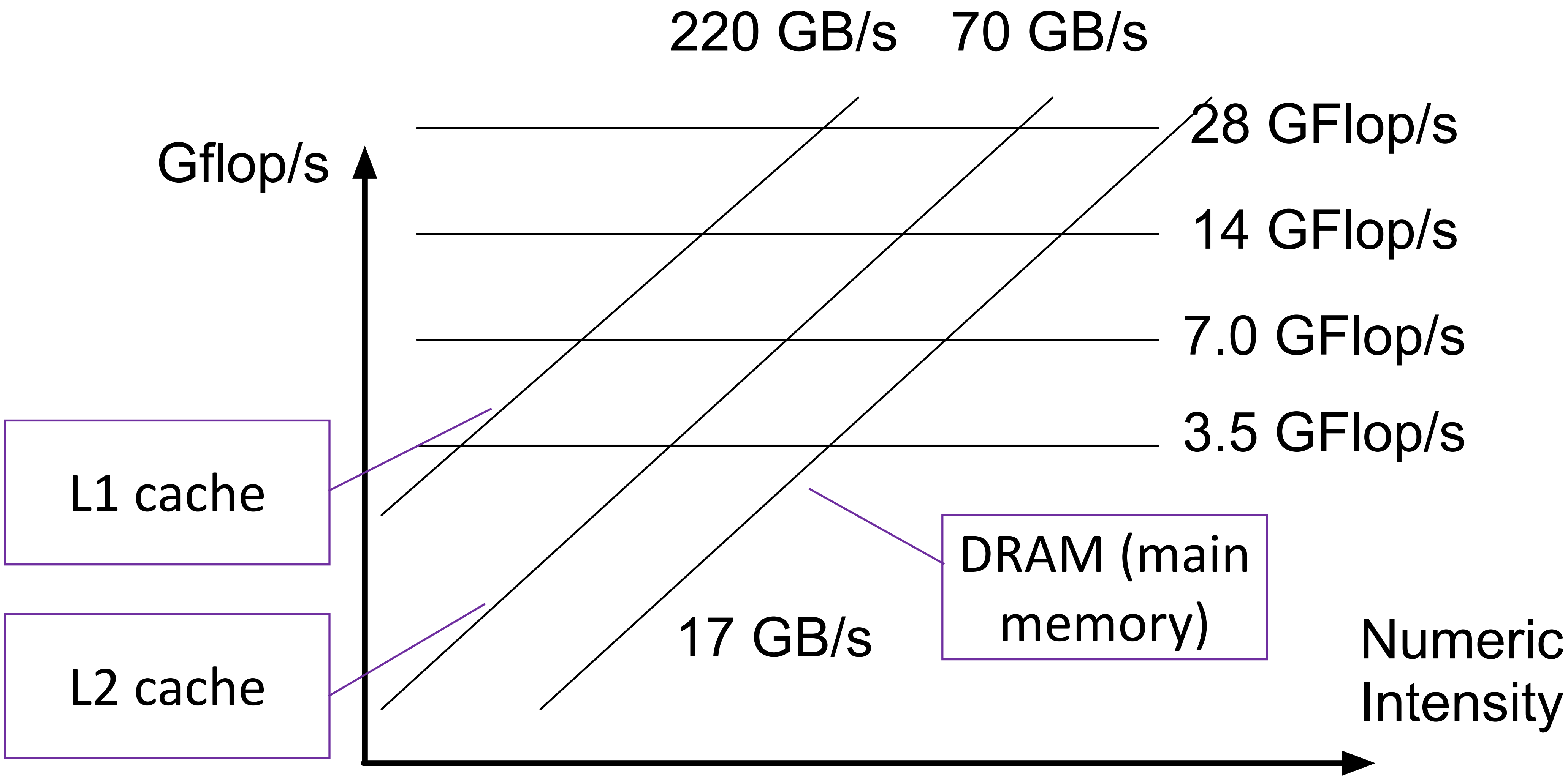
Adds all four doubles ***simultaneously***

Roofline Model

$$\text{Performance} = \frac{\text{GFlops}}{\text{second}} = \min \begin{cases} \text{CPU Peak } \frac{\text{GFlops}}{\text{second}} \\ \text{Bandwidth } \frac{\text{Gbytes}}{\text{second}} \times \text{Numerical Intensity } \frac{\text{GFlops}}{\text{Gbyte}} \end{cases}$$



Roofline Model



General Performance Principles

- Work harder
  - Faster core
- Work smarter
  - Branch predictions, etc
  - Better compilation
  - Better algorithm
  - Better implementation
- Get help

Dennard scaling (ended 2005)

What about this?

We did this

Another Way to Work Smarter

Work less

Strassen's Algorithm

Volker Strassen.
Gaussian Elimination is not Optimal.
Numer Math, Vol 13, No.4, Aug 1969.

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$\begin{aligned} C_{00} &= A_{00}B_{00} + A_{01}B_{10} \\ C_{01} &= A_{00}B_{01} + A_{01}B_{11} \\ C_{10} &= A_{10}B_{00} + A_{11}B_{10} \\ C_{11} &= A_{10}B_{01} + A_{11}B_{11} \end{aligned}$$

Eight multiplies

If these are matrix blocks: Eight matrix multiplies

Strassen's Algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B & \end{bmatrix}$$

Seven matrix multiplies

Seven multiplies

Recurse

$$\begin{aligned}
T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\
T_1 &= (A_{10} + A_{11})(B_{00}) \\
T_2 &= (A_{00})(B_{01} - B_{11}) \\
T_3 &= (A_{11})(B_{10} - B_{00}) \\
T_4 &= (A_{00} + A_{01})(B_{11}) \\
T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\
T_6 &= (A_{01} - A_{11})(B_{10} + B_{11})
\end{aligned}$$

$$\begin{aligned}
C_{00} &= T_0 + T_3 - T_4 + T_6 \\
C_{01} &= T_2 + T_4 \\
C_{10} &= T_1 + T_4 \\
C_{11} &= T_0 - T_1 + T_2 + T_5
\end{aligned}$$

Many adds and subtracts

Strassen's Algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

Seven matrix multiplies

Recurse

$$\begin{aligned}
T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\
T_1 &= (A_{10} + A_{11})(B_{00}) \\
T_2 &= (A_{00})(B_{01} - B_{11}) \\
T_3 &= (A_{11})(B_{10} - B_{00}) \\
T_4 &= (A_{00} + A_{01})(B_{11}) \\
T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\
T_6 &= (A_{01} - A_{11})(B_{10} + B_{11})
\end{aligned}$$

$$\begin{aligned}
C_{00} &= T_0 + T_3 - T_4 + T_6 \\
C_{01} &= T_2 + T_4 \\
C_{10} &= T_1 + T_4 \\
C_{11} &= T_0 - T_1 + T_2 + T_5
\end{aligned}$$

$$O(N^3) \text{ work vs } O(N^2) \text{ data}$$

Multiply

Add

Strassen's Algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

Divide and Conquer

$$\begin{aligned} T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\ T_1 &= (A_{10} + A_{11})(B_{00}) \\ T_2 &= (A_{00})(B_{01} - B_{11}) \\ T_3 &= (A_{11})(B_{10} - B_{00}) \\ T_4 &= (A_{00} + A_{01})(B_{11}) \\ T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\ T_6 &= (A_{01} - A_{11})(B_{10} + B_{11}) \end{aligned}$$

$$\begin{aligned} C_{00} &= T_0 + T_3 - T_4 + T_6 \\ C_{01} &= T_2 + T_4 \\ C_{10} &= T_1 + T_4 \\ C_{11} &= T_0 - T_1 + T_2 + T_5 \end{aligned}$$

Recurse

Seven matrix multiplication

$$O(N^3) \text{ work vs } O(N^2) \text{ data}$$

Each block is size $\frac{N}{2}$ → $\left(\frac{N}{2}\right)^3 = \frac{N^3}{8}$ → $\frac{7}{8}N^3$

Strassen's Algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$\frac{7}{8}\frac{7}{8}\cdots\frac{7}{8}$$

How many of these

Divide and conquer

$$\begin{aligned} T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\ T_1 &= (A_{10} + A_{11})(B_{00}) \\ T_2 &= (A_{00})(B_{01} - B_{11}) \\ T_3 &= (A_{11})(B_{10} - B_{00}) \\ T_4 &= (A_{00} + A_{01})(B_{11}) \\ T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\ T_6 &= (A_{01} - A_{11})(B_{10} + B_{11}) \end{aligned}$$

$$\begin{aligned} C_{00} &= T_0 + T_3 - T_4 + T_6 \\ C_{01} &= T_2 + T_4 \\ C_{10} &= T_1 + T_4 \\ C_{11} &= T_0 - T_1 + T_2 + T_5 \end{aligned}$$

$$\log_2(N)$$

$$O(N^{\log_2 7})$$

$$O(N^{\log_2 7}) \ll O(N^{\log_2 8}) = O(N^3)$$

Strassen's Algorithm

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$\begin{aligned} T_0 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\ T_1 &= (A_{10} + A_{11})(B_{00}) \\ T_2 &= (A_{00})(B_{01} - B_{11}) \\ T_3 &= (A_{11})(B_{10} - B_{00}) \\ T_4 &= (A_{00} + A_{01})(B_{11}) \\ T_5 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\ T_6 &= (A_{01} - A_{11})(B_{10} + B_{11}) \end{aligned}$$

$$\begin{aligned} C_{00} &= T_0 + T_3 - T_4 + T_6 \\ C_{01} &= T_2 + T_4 \\ C_{10} &= T_1 + T_4 \\ C_{11} &= T_0 - T_1 + T_2 + T_5 \end{aligned}$$

$O(N^{2.38})$

Better algorithms

Limit?

Require large N

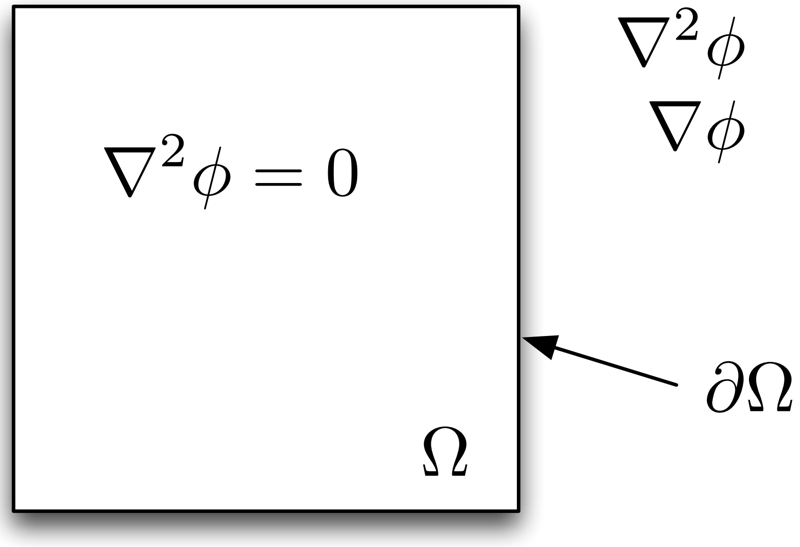
Limit Unknown, Biggest open question in numerical linear algebra

Another Way to Work Smarter

(Work less)

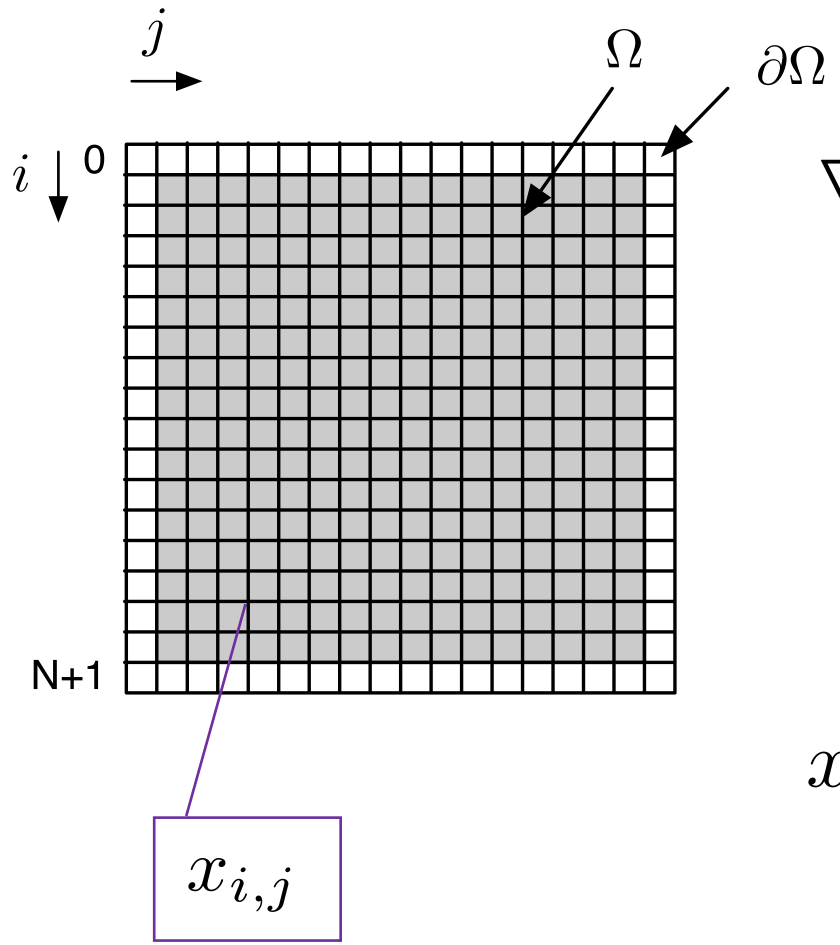
In Practice

- Many scientific applications are based on solving systems of partial differential equations that model physical phenomena
- Laplace's equation on unit square is prototypical PDE



$$\begin{aligned} \nabla^2 \phi &= 0 \quad \text{on } \Omega \\ \nabla \phi &= f \quad \text{on } \partial\Omega \end{aligned}$$

Laplace's Equation on a Regular Grid



$$
\begin{aligned}
\nabla^2\phi &= 0 \quad \text{on } \Omega \\
\phi &= f \quad \text{on } \partial\Omega
\end{aligned}
$$

Discretization

$$x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4x_{i,j} = 0$$

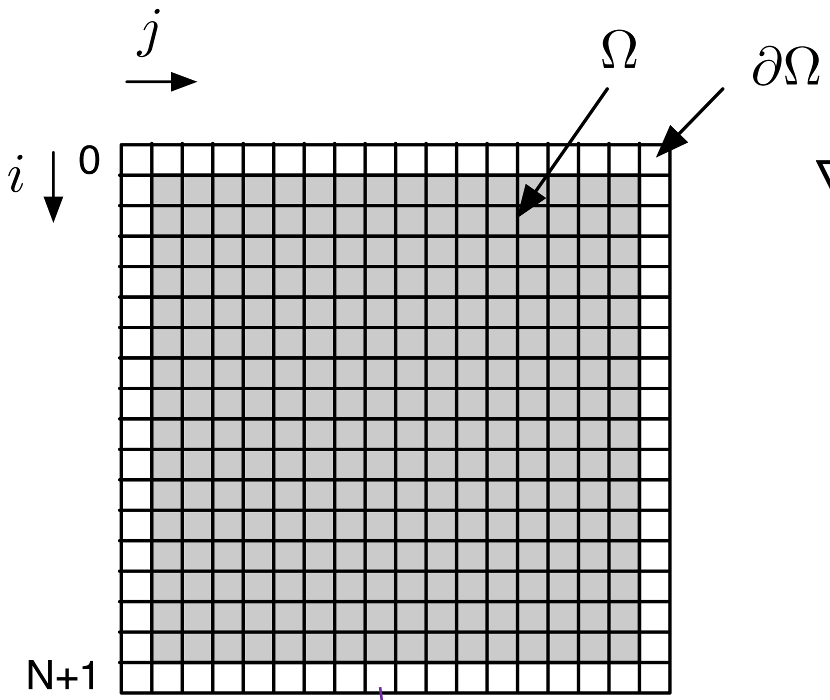
$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

$$x_{i,j} = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4$$

The value of each point on the grid

The average of its neighbors in 4 directions

Laplace's Equation on a Regular Grid



The boundary is non-zero

$$\nabla^2 \phi = 0 \quad \text{on } \Omega$$
$$\phi = f \quad \text{on } \partial\Omega$$

Why isn't 0 the solution?

The boundary is non-zero

Discret

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

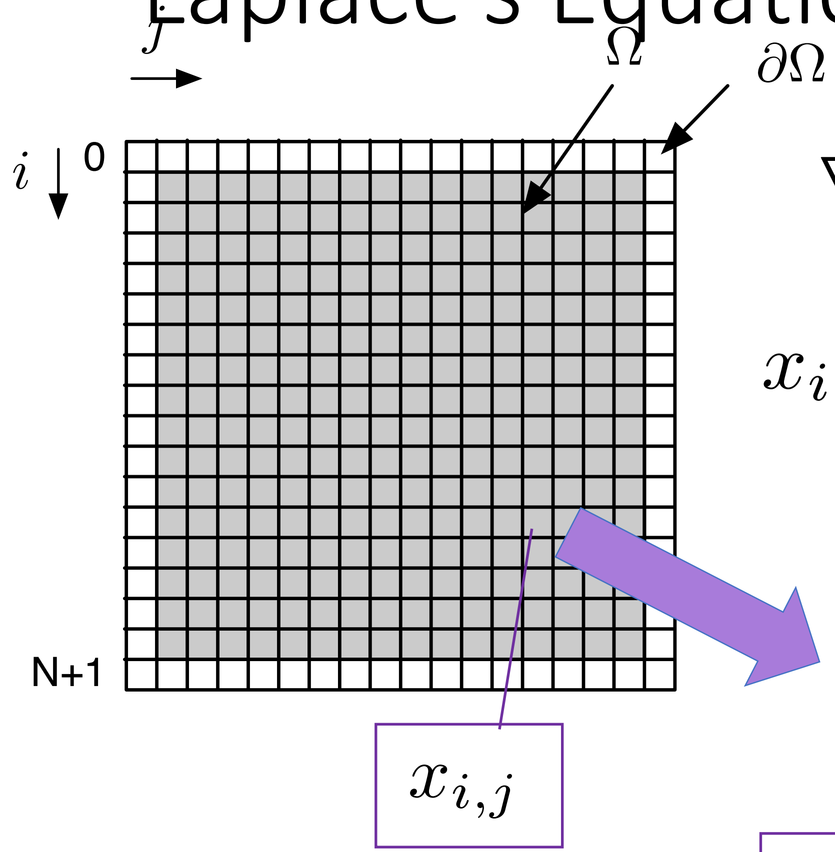
Non-zeros in here due to boundary

$$x_{i,j} = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4$$

The value of each point on the grid

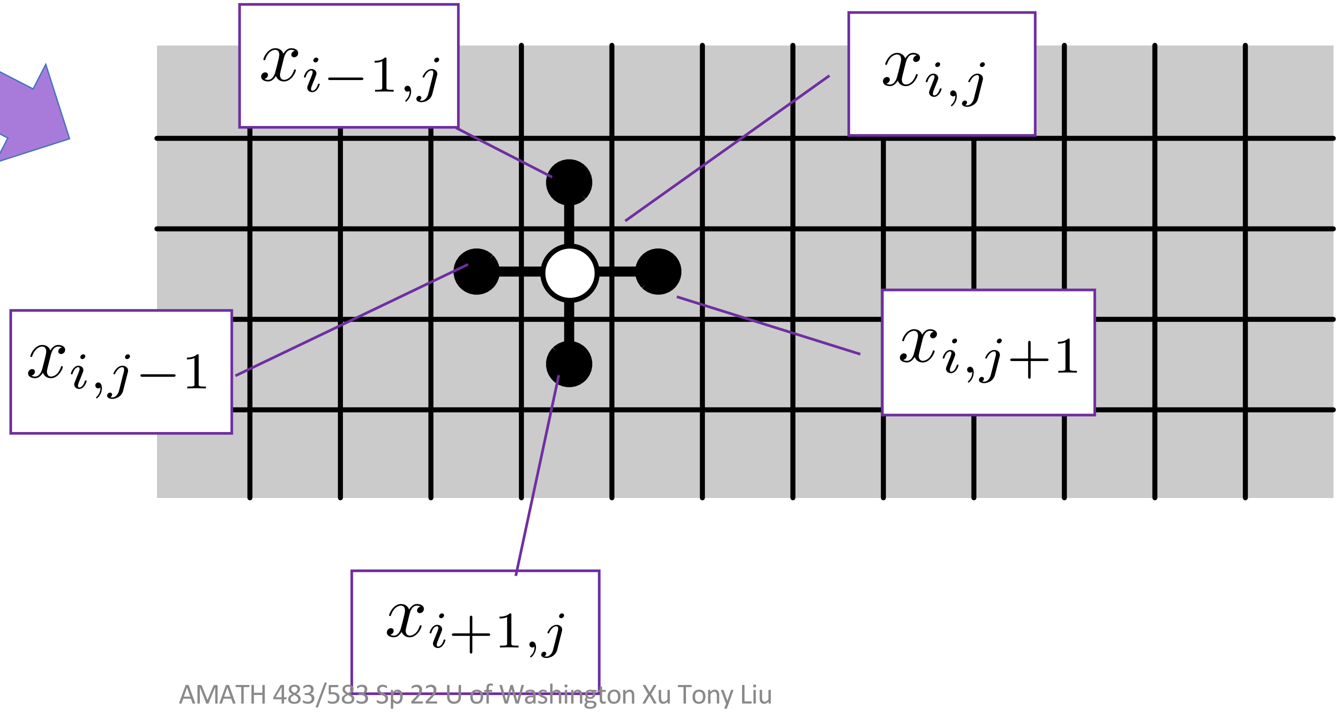
The average of its neighbors

Laplace's Equation on a Regular Grid

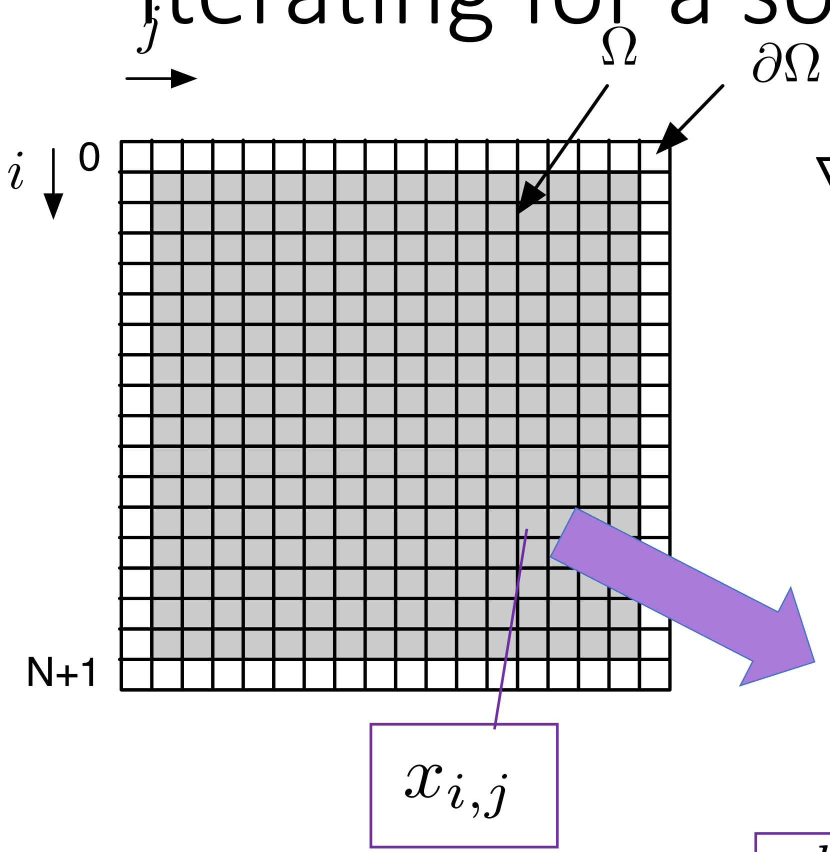


$$
\begin{aligned}
\nabla^2\phi &= 0 \quad \text{on } \Omega \\
\phi &= f \quad \text{on } \partial\Omega
\end{aligned}
$$

$$x_{i,j} = (x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1})/4$$



Iterating for a solution

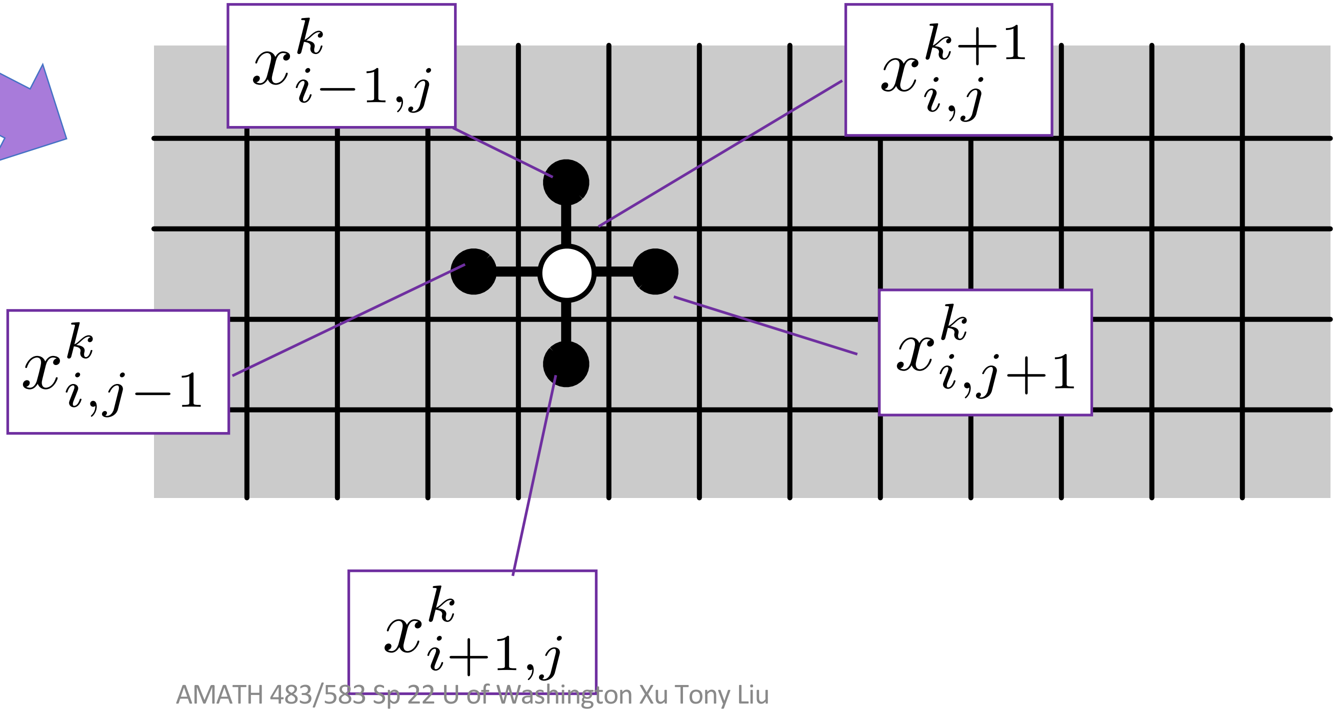


$$\nabla^2\phi = 0 \quad \text{on } \Omega$$
$$\phi = f \quad \text{on } \partial\Omega$$

Approximation at iteration k+1

Average of approximation at iteration k

$$x_{i,j}^{k+1} = (x_{i-1,j}^{k} + x_{i+1,j}^{k} + x_{i,j-1}^{k} + x_{i,j+1}^{k})/4$$



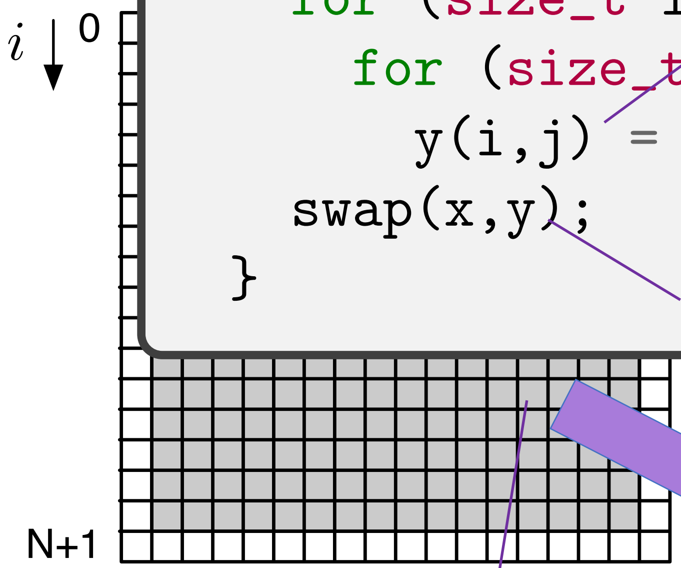
```
while (! converged())
  for (size_t i = 1; i < N+1; ++i)
    for (size_t j = 1; j < N+1; ++j)
      y(i,j) = (x(i-1,j) + x(i+1,j) + x(i,j-
  swap(x,y);
}
```

Approximation at iteration k+1

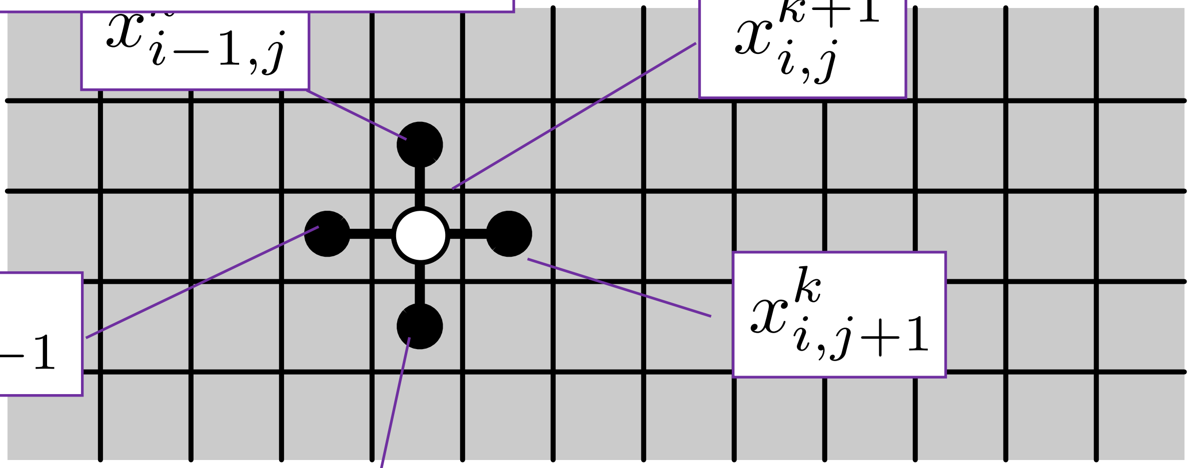
Average of approximation at iteration k

At end of each outer iteration: new becomes old (and v.v.)

Only need to use two arrays to do iteration: old and new



$x_{i,j}$



$x^{k}_{i-1,j}$

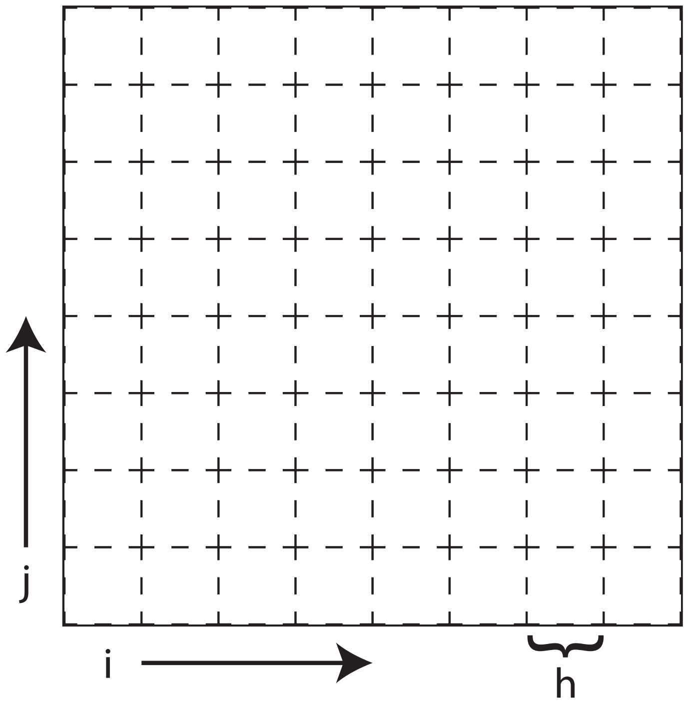
$x^{k+1}_{i,j}$

$x^{k}_{i,j-1}$

$x^{k}_{i,j+1}$

$x^{k}_{i+1,j}$

Discretized



- Del operator

$$\begin{aligned}\nabla\phi &= \frac{\partial\phi}{\partial x}+\frac{\partial\phi}{\partial y}\\ \nabla^2\phi &= \frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}\end{aligned}$$

- Finite difference approximation to derivative

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t}(t_0) &\approx \frac{x(t_0+h)-x(t_0)}{h}\\ \frac{\mathrm{d}^2x}{\mathrm{d}t^2}(t_0) &\approx \frac{\frac{\mathrm{d}x}{\mathrm{d}t}(t_0+h)-\frac{\mathrm{d}x}{\mathrm{d}t}(t_0)}{h}\\ &= \frac{x(t_0+h+h)-x(t_0+h)-x(t_0+h)+x(t_0)}{h^2}\\ &= \frac{x(t_0+2h)-2x(t_0+h)+x(t_0)}{h^2}\\ &= \frac{x(t_0+h)-2x(t_0)+x(t_0-h)}{h^2}\end{aligned}$$

- Finite difference approximation to del

$$\frac{\phi_{i+1,j}+\phi_{i-1,j}+\phi_{i,j-1}+\phi_{i,j+1}-4\phi_{i,k}}{h^2}=0$$

Matrix Formulation

- Lexicographically order unknowns (note some will be boundary values)

$$\frac{x_{i+1} + x_{i-1} + x_{i+N} + x_{i-N} - 4x_i}{h^2} = 0$$

- Formulate as a matrix problem:
- Laplacian matrix

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \end{bmatrix}$$

Linear System Solution

```
void multiply(const Matrix& A, const Matrix& B, Mat
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < B.num_cols(); ++j) {
      for (size_t k = 0; k < A.num_cols(); ++k) {
        C(i, j) += A(i, k) * B(k, j);
      }
    }
  }
}
```

Matrix-matrix product is kernel operation

What happens with the Laplacian matrix?

Multiplying and adding zero to zero

Work Smarter! Don't multiply and add zero to zero

Solution?

```
void multiply(const Matrix& A, const Matrix& B, Matrix& C) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < B.num_cols(); ++j) {
      for (size_t k = 0; k < A.num_cols(); ++k) {
        if(A(i,k) != 0.0 && B(k,j) != 0) {
          C(i, j) += A(i, k) * B(k, j);
        }
      }
    }
  }
}
```

Avoid zeros

But we still touch every element

And that's what expensive

Solution?

```
void multiply(const Matrix& A, const Matrix& B, Matrix& C) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < B.num_cols(); ++j) {
      for (size_t k = 0; k < A.num_cols(); ++k) {
        if(A(i,k) != 0.0 && B(k,j) != 0) {
          C(i, j) += A(i, k) * B(k, j);
        }
      }
    }
  }
}
```

We need to avoid zeros

Without looking to see if there is a zero

Solution: Sparse Matrices

In order to avoid zeros

Don't store zeros

A zero is a null op

Use data structures and algorithms accordingly

Sparse matrix techniques

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Solving Sparse Systems

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \end{bmatrix}$$

- Work only with non-zeros
- Direct methods
  - Perform LU factorization on sparse matrix
  - Create non-zeros during elimination process
  - Pre-order (using heuristics) to minimize the amount of fill
  - Very sequential
  - Fill can be quite significant
- Iterative methods
  - Successively create better approximations to x
  - Relaxation methods (e.g., Jacobi) – very very very slow to converge
  - Krylov subspace methods (e.g., conjugate gradient)
    - Good preconditioning often required

A zero here

Turns into a non-zero

Need to create new space (fill)

Conjugate Gradient Algorithm

Initial $r^{(0)}=b-Ax^{(0)}$

For i=1, 2, …

solve $Mz^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)T} z^{(i-1)}$

If i=1

$p^{(1)} = z^{(0)}$

Else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

Endif

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)T} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

Check convergence

end

Key operation

```
mult(A, scaled(x, -1.0), b, r);
while (! iter.finished(r)) {
  solve(M, r, z);
  rho = dot_conj(r, z);

  if ( iter.first() )
    copy(z, p);
  else {
    beta = rho / rho_1;
    add(z, scaled(p, beta), p);
  }
  mult(A, p, q);
  alpha = rho / dot_conj(p, q);
  add(x, scaled(p, alpha), x);
  add(r, scaled(q, -alpha), r);
  rho_1 = rho;
  ++iter;
}
```

Sparse Storage

- A matrix is map from two indices to a value

$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

- So if we want to store just elements that are not zero (the “non-zeros”)
- We need to store the two indices and the value

Dense Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

Dense storage: all matrix elements are kept

At location corresponding to indices

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

Matrix-Vector Product

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

```
void matvec(const Matrix& A, const Vector& x, const Vector& y) }
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
}
```

And thus all values of A

Zeros and non-zeros

We go through all possible valid indices

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

Matrix-Vector Product

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

```
void matvec(const Matrix& A, const Vector& x, const Vector& y) }
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
```

And row index

Need value of matrix entry

And column index

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

OK. We've stored all values

With dense storage, we loop through all possible indices and look up corresponding value

Sparse Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

```
void matvec(const Matrix& A, const Vector& x, const Vector& y) }
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
}
```

Goal: Loop over all indices for non-zero entries

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

So we need to store indices also

Store only the non-zeros

But what is non-zero is a property of matrix

Algorithm can't know it

Sparse Storage

$$(0,0) \qquad (0,3)$$

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

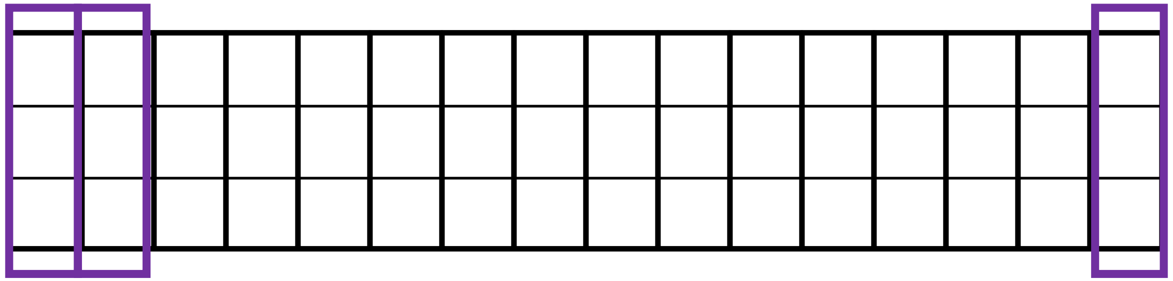
```
void matvec(const Matrix& A, const Vector& x, const Vector& y) }
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
}
```

Goal: Loop over all indices for non-zero entries

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

Does order of elements matter?

Coordinate Storage (Array of Structs)

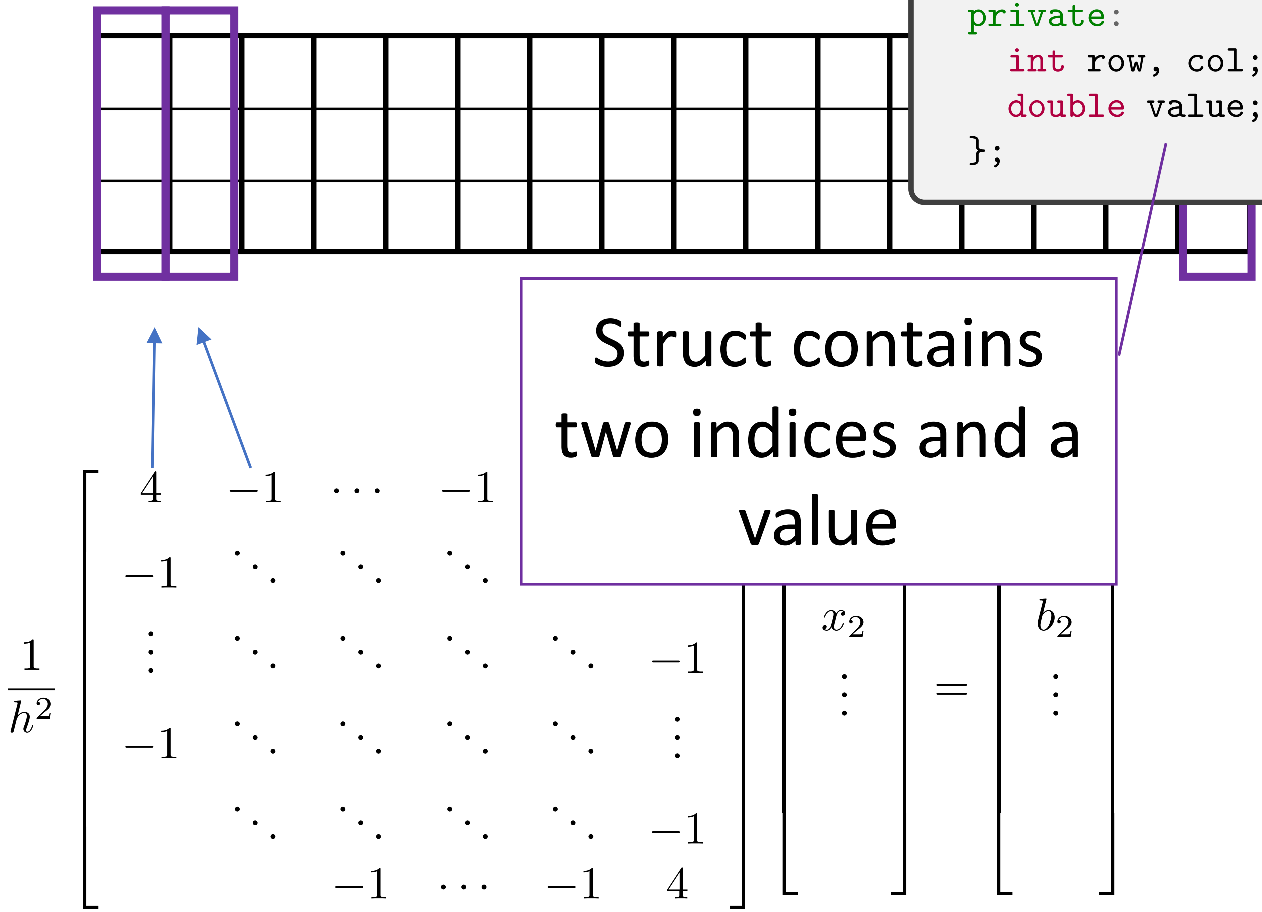


$$\frac{1}{h^2}\begin{bmatrix} 4 & -1 & \cdots & -1 & & \\ -1 & \ddots & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & -1 \\ -1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & \ddots & -1 \\ & & -1 & \cdots & -1 & 4 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \\ \\ \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Coordinate Storage (Array of Structs)

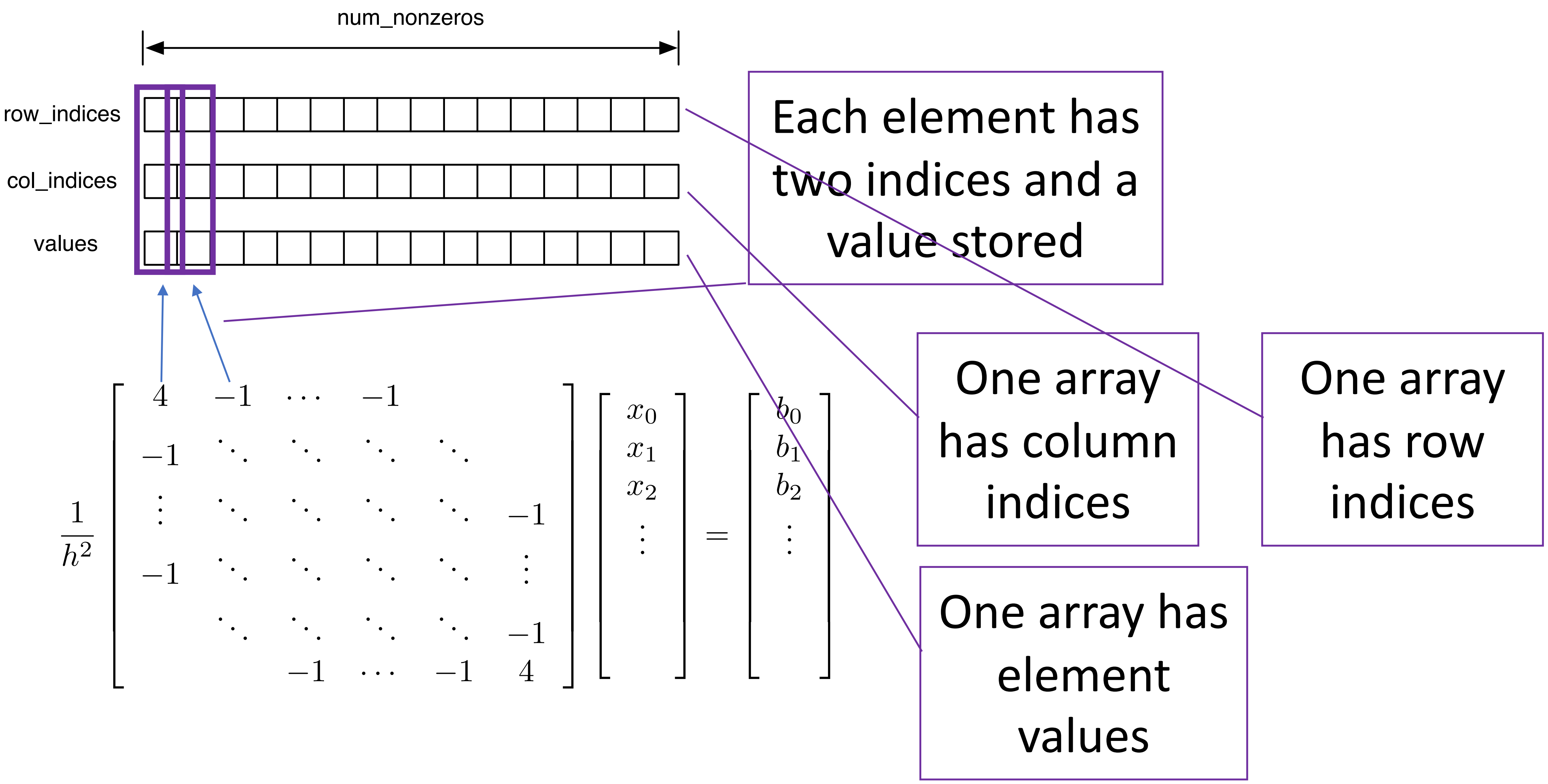
```
struct Element {
private:
  int row, col;
  double value;
};
```

```
struct COOMatrix {
private:
  std::vector<Element> arrayData;
};
```

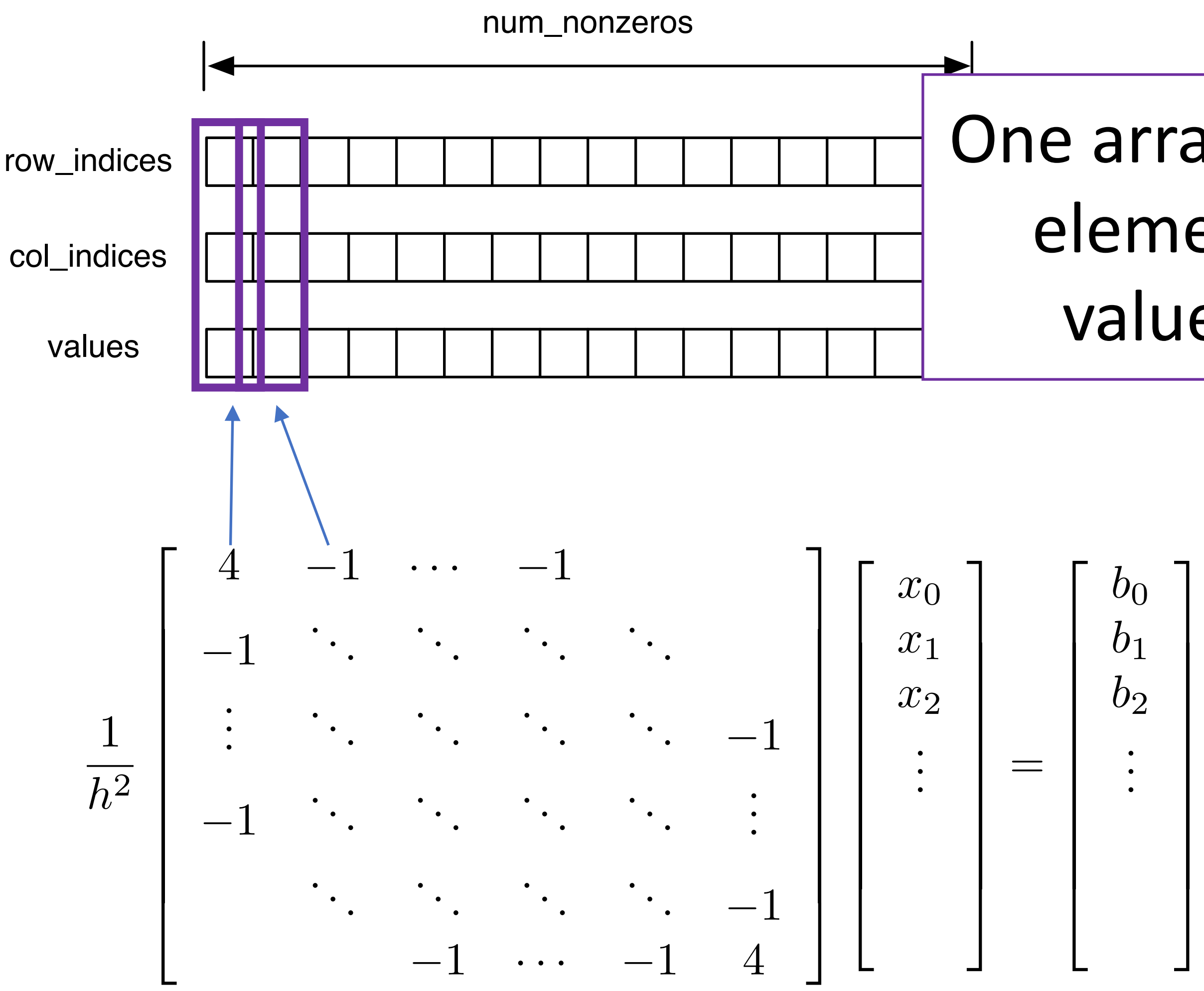


Single array with 3-element structs

Coordinate Storage (Struct of Arrays)



Coordinate Storage (Struct of Arrays)



One array has element values

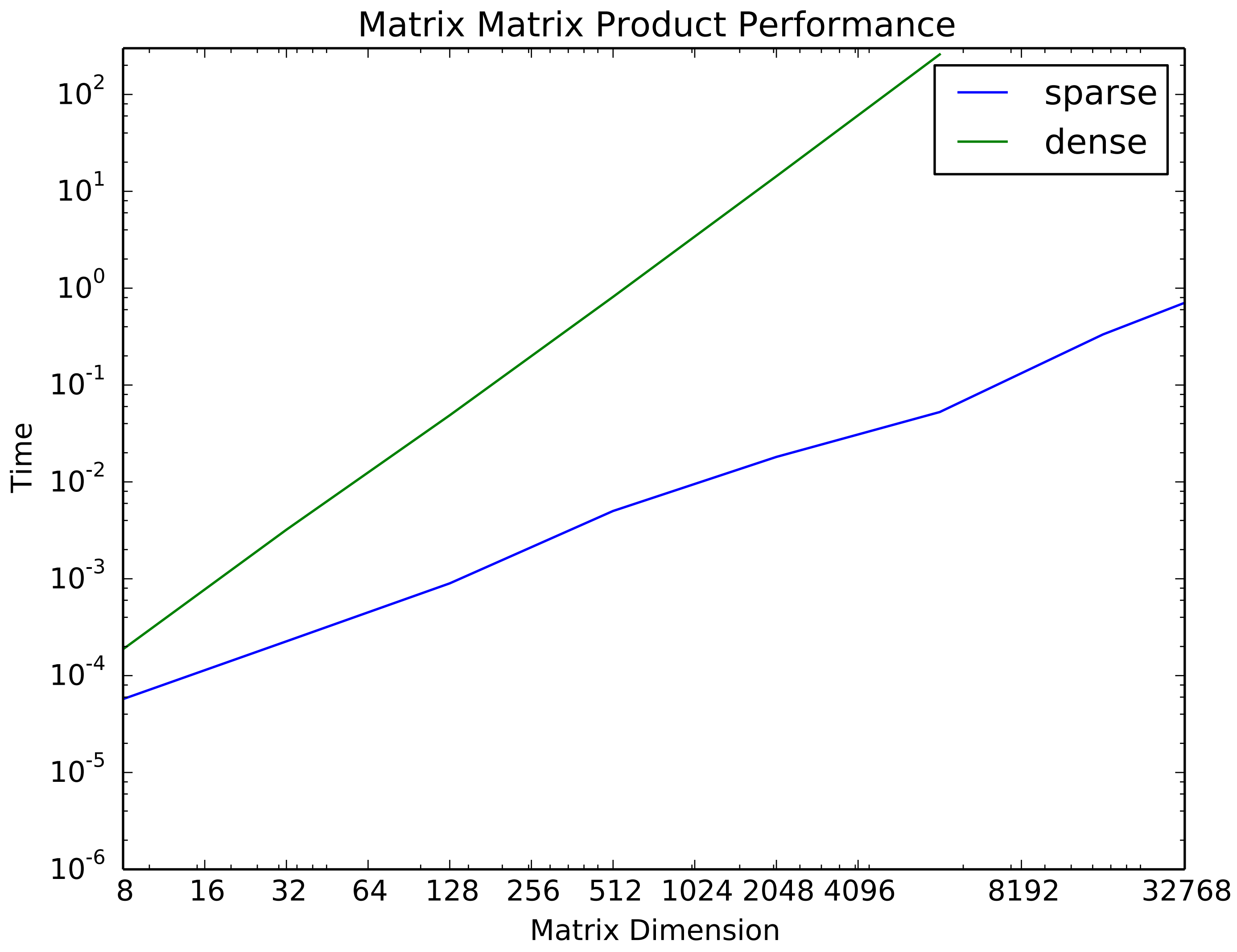
```cpp
struct COOMatrix {
private:
  std::vector<size_t> row_indices_;
  std::vector<size_t> col_indices_;
  std::vector<double> values_;
};
```

One array has column indices

One array has row indices

Conventional Wisdom: Struct of Arrays is faster

Performance Comparison



What's the Catch?

```
class Matrix {
public:
  Matrix(size_t M, size_t N) : num_rows_(M), num_cols_(N), storage_(num_rows_ * num_cols_) {}

        double& operator()(size_t i, size_t j)       { return storage_[i * num_cols_ + j]; }
  const double& operator()(size_t i, size_t j) const { return storage_[i * num_cols_ + j]; }

  size_t num_rows() const { return num_rows_; }
  size_t num_cols() const { return num_cols_; }

private:
  size_t              num_rows_, num_cols_;
  std::vector<double> storage_;
};
```

In fact, it's a reference, so we can modify it

Provide indices, get back value

In constant time

Uh…

```cpp
class COOMatrix {
public:
  COOMatrix(size_t M, size_t N) : num_rows_(M),

  size_t num_rows()     const { return num_rows_; }
  size_t num_cols()     const { return num_cols_; }

private:
  size_t num_rows_, num_cols_;
  std::vector<size_t> row_indices_, col_indices_;
  std::vector<double> storage_;
};
```

How do we get to a value (in constant time)?

We can’t

Next Problem

```
void matvec(const Matrix& A, const Vector& x, Vector& y) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
}
```

Nice external function using operator()()

```
void matvec(const COOMatrix& A, const Vector& x, Vector& y) {
 // ??
}
```

No operator()() no external function

Coordinate Matvec

```
void matvec(const Matrix& A, const Vector& x, Vector& y) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) += A(i, j) * x(j);
    }
  }
}
```

This is the row index

This is the value

This is the column index

Coordinate Matvec

```
void matvec(const Matrix& A, const Vector& x, Vector& y) {
  for (size_t i = 0; i < A.num_rows(); ++i) {
    for (size_t j = 0; j < A.num_cols(); ++j) {
      y(i) = A(i, j) * x(j);
    }
  }
}
```

Index into y with row index

Multiply by the corresponding value

Index into x with column index

We have these three things in coordinate format

Coordinate Matrix Mat Vec

```
class COOMatrix {
public:
  COOMatrix(size_t M, size_t N) : num_rows_(M), num_cols_(N)

  void matvec(const Vector& x, Vector& y) const {
    for (size_t k = 0; k < storage_.size(); ++k) {
      y(row_indices_[k]) += storage_[k] * x(col_indices[k]);
    }
  }

private:
  int num
  std::ve            _indic
  std::ve            rage_;
};
```

Meditate on this

Index into y with row index

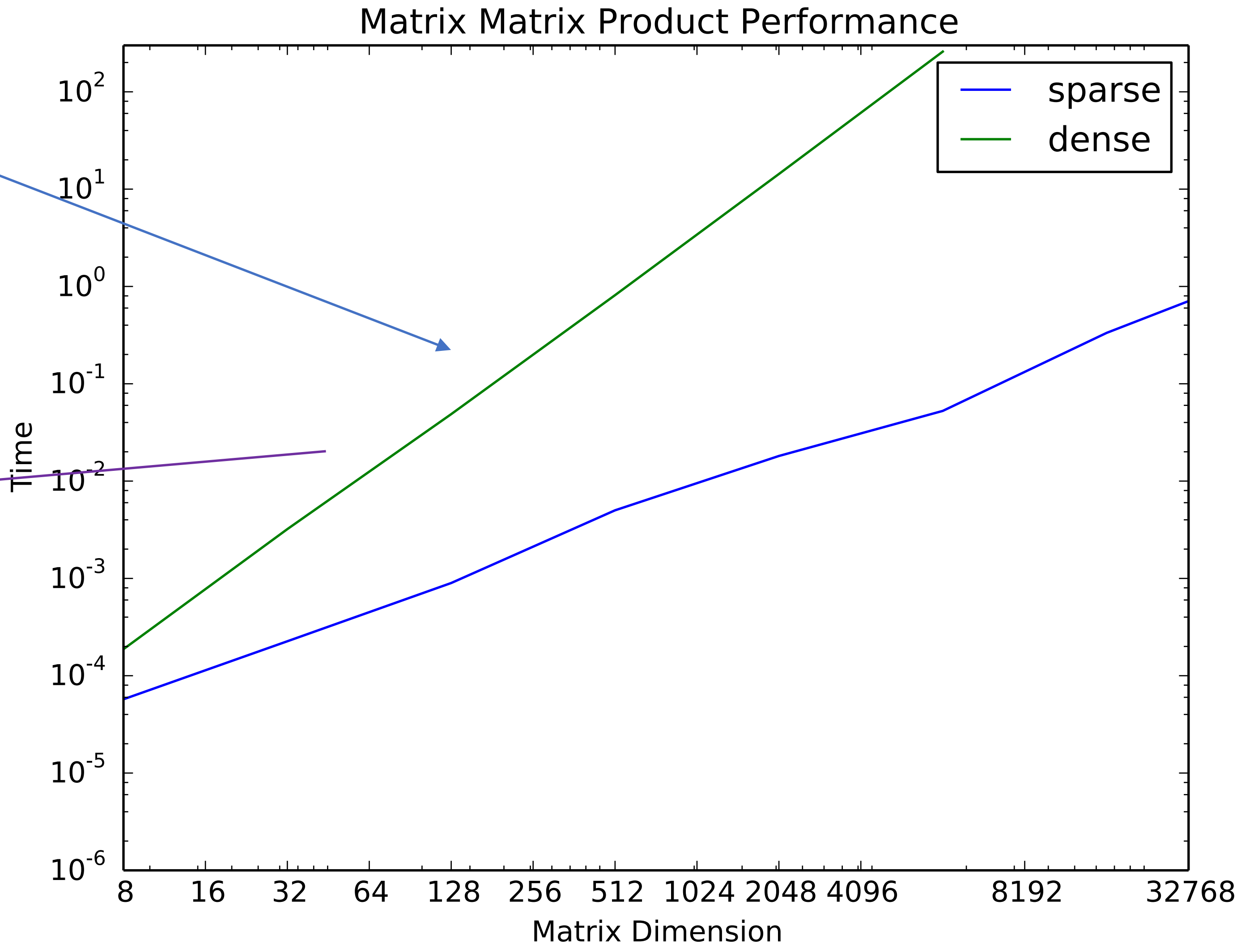
Multiply by corresponding value

Index into x with column index

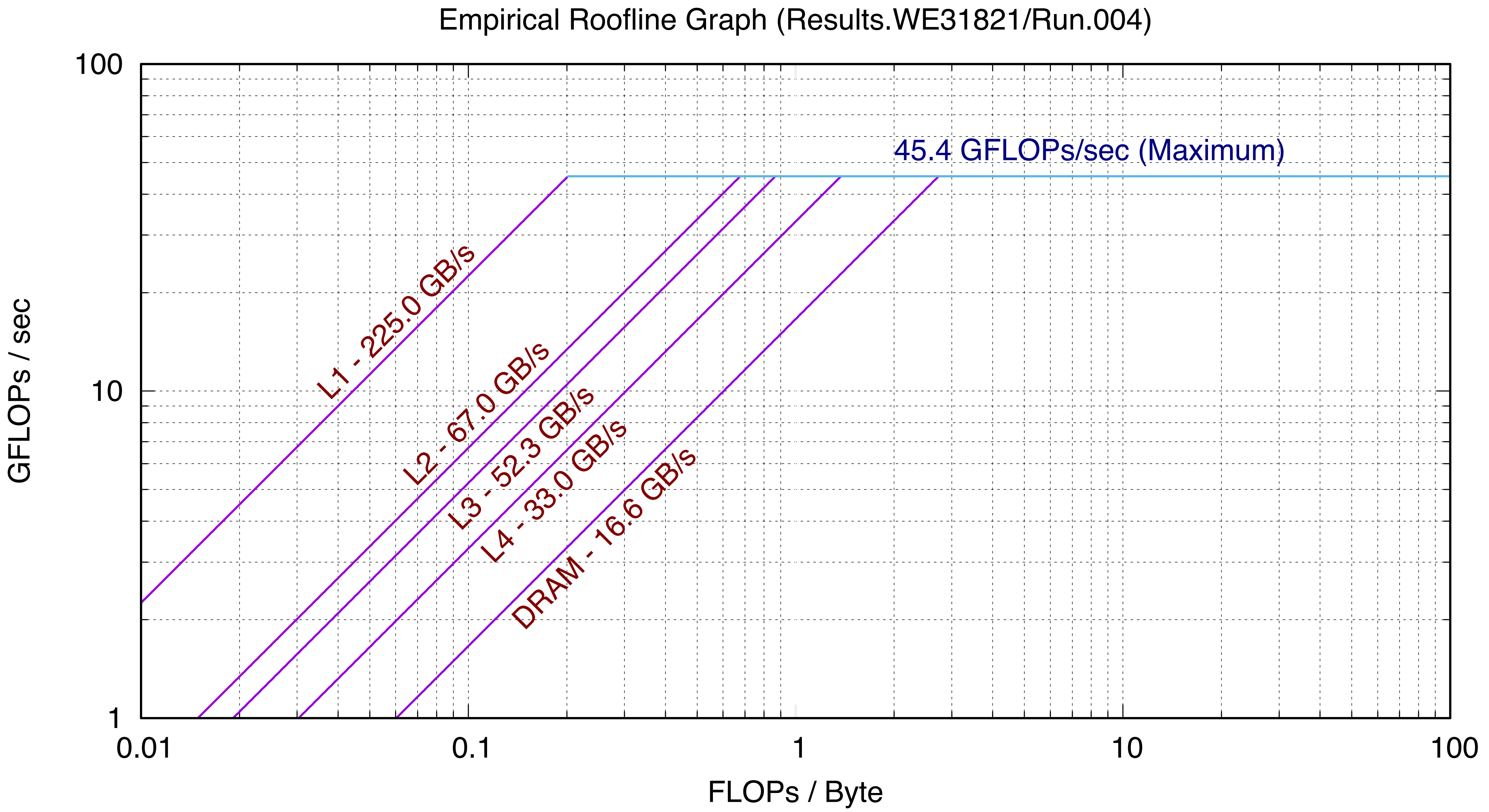
Performance Comparison

$O(N^3)$

10X problem size = 1000X run time



Roofline



Empirical Roofline Graph (Results.WE31821/Run.004)
45.4 GFLOPs/sec (Maximum)
L1 - 225.0 GB/s
L2 - 67.0 GB/s
L3 - 52.3 GB/s
L4 - 33.0 GB/s
DRAM - 16.6 GB/s
GFLOPs / sec
100
10
1
0.01
0.1
1
10
100
FLOPs / Byte

Numerical Intensity

```
void matvec(const Vector& x, Vector& y) const {
  for (size_type k = 0; k < arrayData.size(); ++k) {
    y(rowIndices[k]) += arrayData[k] * x(rowIndices[k]);
  }
}
```

Three doubles + 2 ints
= 32 bytes? (36 bytes?)

Two flops

2 NNZ Flops

10N
Flops

5N

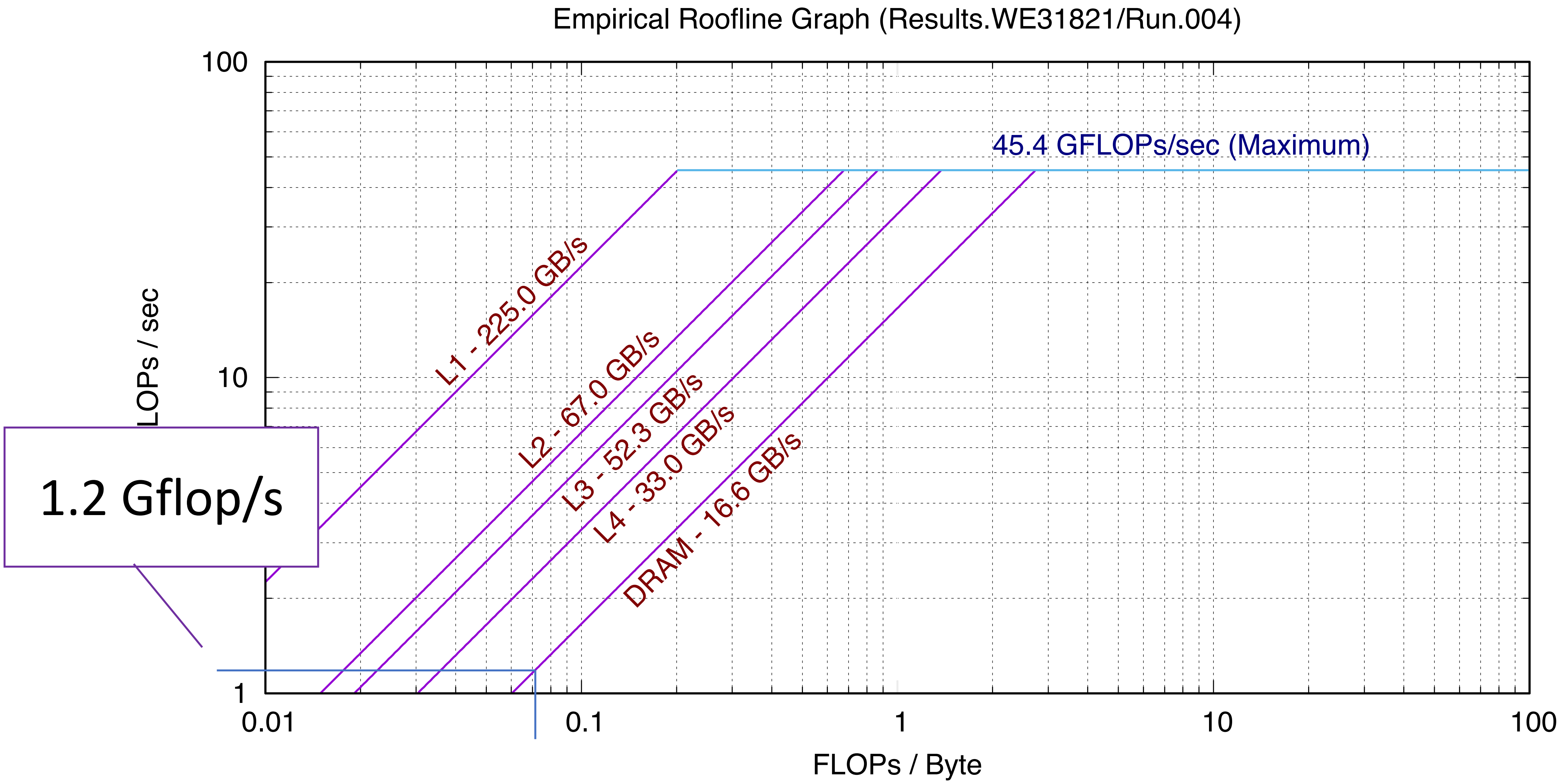
NNZ doubles
$+2$ NNZ indexes
$+2N$ doubles

7N doubles =
56 bytes

$\frac{1}{14}$ $\frac{\text{Flop}}{\text{byte}}$

10N indexes =
40, 80 bytes

Measured


Empirical Roofline Graph (Results.WE31821/Run.004)
45.4 GFLOPs/sec (Maximum)
L1 - 225.0 GB/s
L2 - 67.0 GB/s
L3 - 52.3 GB/s
L4 - 33.0 GB/s
DRAM - 16.6 GB/s
LOPs / sec
100
10
1
0.01
0.1
1
10
100
FLOPs / Byte
1.2 Gflop/s

Coordinate Storage

```
class COOMatrix {
public:
  COOMatrix(size_t M, size_t N) : num_rows_(M), num_cols_(N) {}

  void matvec(const Vector& x, Vector& y) const {
    for (size_t k = 0; k < storage_.size(); ++k) {
      y(row_indices_[k]) += storage_[k] * x(col_indices[k]);
    }
  }

private:
  int num_rows, num_cols;
  std::vector<size_t> row_indices_, col_indices_;
  std::vector<double> storage_;
};
```

How do we initialize storage_?

In fact, how do we create a sparse matrix?

Filling a Sparse Matrix

```
class COOMatrix {
public:
  COOMatrix(size_t M, size_t N) : num_rows_(M), num_col

  void insert(sizet i, size_t j, double val) {
    row_indices_.push_back(i);
    col_indices_.push_back(j);
    storage_.push_back(val);
  }


private:
  size_t num_rows_, num_cols_;
  std::vector<size_t> row_indices_, col_indices_;
  std::vector<double> storage_;
};
```

Often treated like variable initialization

Matrix is filled with something when created

Can also append elements (no ordering required)

Compressed Sparse Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

Values repeat

But we can sort the elements by either row index or column index

Each array stores same number of elements (nnz)

row_indices	0	0	1	1	1	2	3	3	3	4	4	5
col_indices	0	3	1	2	4	5	0	2	3	1	4	5
storage	3	8	1	4	6	7	5	4	1	3	5	9

Compressed Sparse Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

row_indices	4	0	3	0	1	1	3	1	2	3	5	4
col_indices	4	0	2	3	1	2	3	4	5	0	5	1
storage	5	3	4	8	1	4	1	6	7	5	9	3

Unordered elements

row_indices	0	0	1	1	1	2	3	3	3	4	4	5
col_indices	0	3	1	2	4	5	0	2	3	1	4	5
storage	3	8	1	4	6	7	5	4	1	3	5	9

Elements ordered by row

Note all arrays get reordered

Data representing an element stay together

Run Length Encoding of Row Indices

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

row_indices	0	1	2	3	4	5						
run_length	2	3	1	3	2	1						
col_indices	0	3	1	2	4	5	0	2	3	1	4	5
storage	3	8	1	4	6	7	5	4	1	3	5	9

Do we need this?

Keeps a running total

```
size_t row_ptr = 0;
for (size_t i = 0; i < num_rows_; ++i) {
  for (size_t j = row_ptr; j < row_ptr + row_run_length[i]; ++j)
    y[row_indices_[i]] += storage_[j] * x[col_indices_[j]];
  row_ptr = row_ptr + row_ptr + row_run_length[i];
}
```

Compressed Sparse Row (CSR) Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

row_indices	0	2	5	6	9	11	12					
col_indices	0	3	1	2	4	5	0	2	3	1	4	5
storage	3	8	1	4	6	7	5	4	1	3	5	9

Store running total instead of computing it

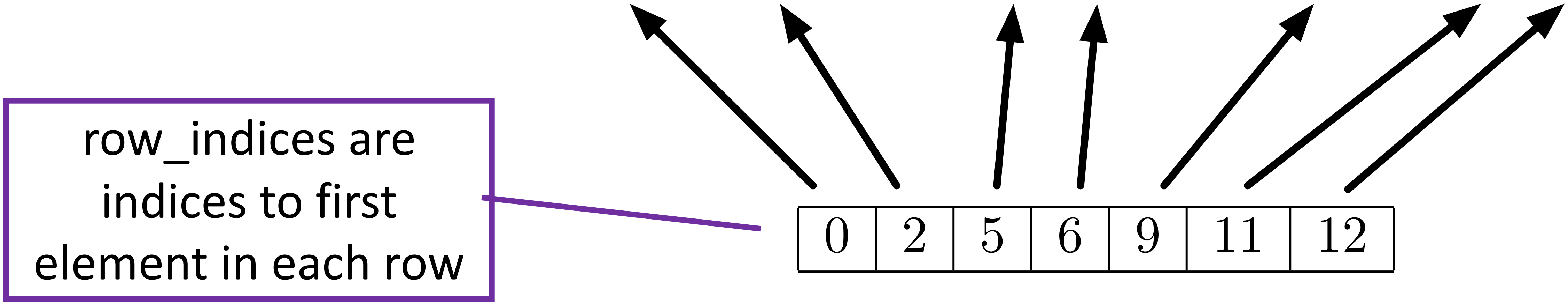
Compressed Sparse Row (CSR) Storage

$$\begin{bmatrix} 3 & 0 & 0 & 8 & 0 & 0 \\ 0 & 1 & 4 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 5 & 0 & 4 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

Size is num_rows_ + 1

One past the end

row_indices	0	2	5	6	9	11	12					
col_indices	0	3	1	2	4	5	0	2	3	1	4	5
storage	3	8	1	4	6	7	5	4	1	3	5	9



CSR Implementation

```
class CSRMatrix {

    ze_t M, size_t N) :  num_rows_(M), num_cols_(N), row_indices_(nu

    ws()     const { return num_rows_; }
    ls()     const { return num_cols_; }
  size_t num_nonzeros() const { return storage_.size(); }

private:
  size_t num_rows_, num_cols_;
  std::vector<size_t> row_indices_, col_indices_;
  std::vector<double> storage_;
};
```

Constructor

And row_indices_

Note initial value

Initialize num_rows and num_cols

Matrix size accessors

Useful info for sparse matrix

Private implementation

CSR Implementation (Matrix Vector Multiply)

```
class CSRMatrix {

public:
  CSRMatrix(size_t M, size_t N) :  num_rows_(M), num_cols_(N ... _rows_+1, 0)  {}

  void matvec(const Vector& x, Vector& y) const {
  for (size_t i = 0; i < num_rows_; ++i) {
    for (size_t j = row_indices_[i]; j < row_indices_[i+1]; ++j) {
      y(i) += storage_[j] * x(col_indices_[j]);
    }
  }
}

private:
  size_t num_rows_, num_cols_;
  std::vector<size_t> row_indices_, col_indices_;
  std::vector<double> storage_;
};
```

For each row

For each element in that row

Row index

Matrix value

Column index

Meditate on this

Building a CSR Matrix

```
class CSRMatrix {

public:
   void open_for_push_back()  { is_open = true; }

  void close_for_push_back() { is_open = false;
    for (size_t i = 0; i < num_rows_; ++i) row_indices_[i+1] += row_indices_[i];
    for (size_t i = num_rows_; i > 0; --i)   row_indices_[i] = row_indices_[i-1];
    row_indices_[0] = 0;
  }

  void push_back(size_t i, size_t j, double value) {
    ++row_indices_[i];
    col_indices_.push_back
    storage_.push_back(val
  }

private:
  bool is_open;
  size_t num_rows_, num_cols_;
  std::vector<size_t> row_indices
  std::vector<double> storage_;
};
```

When done pushing, accumulate run lengths to offsets

Should be checked

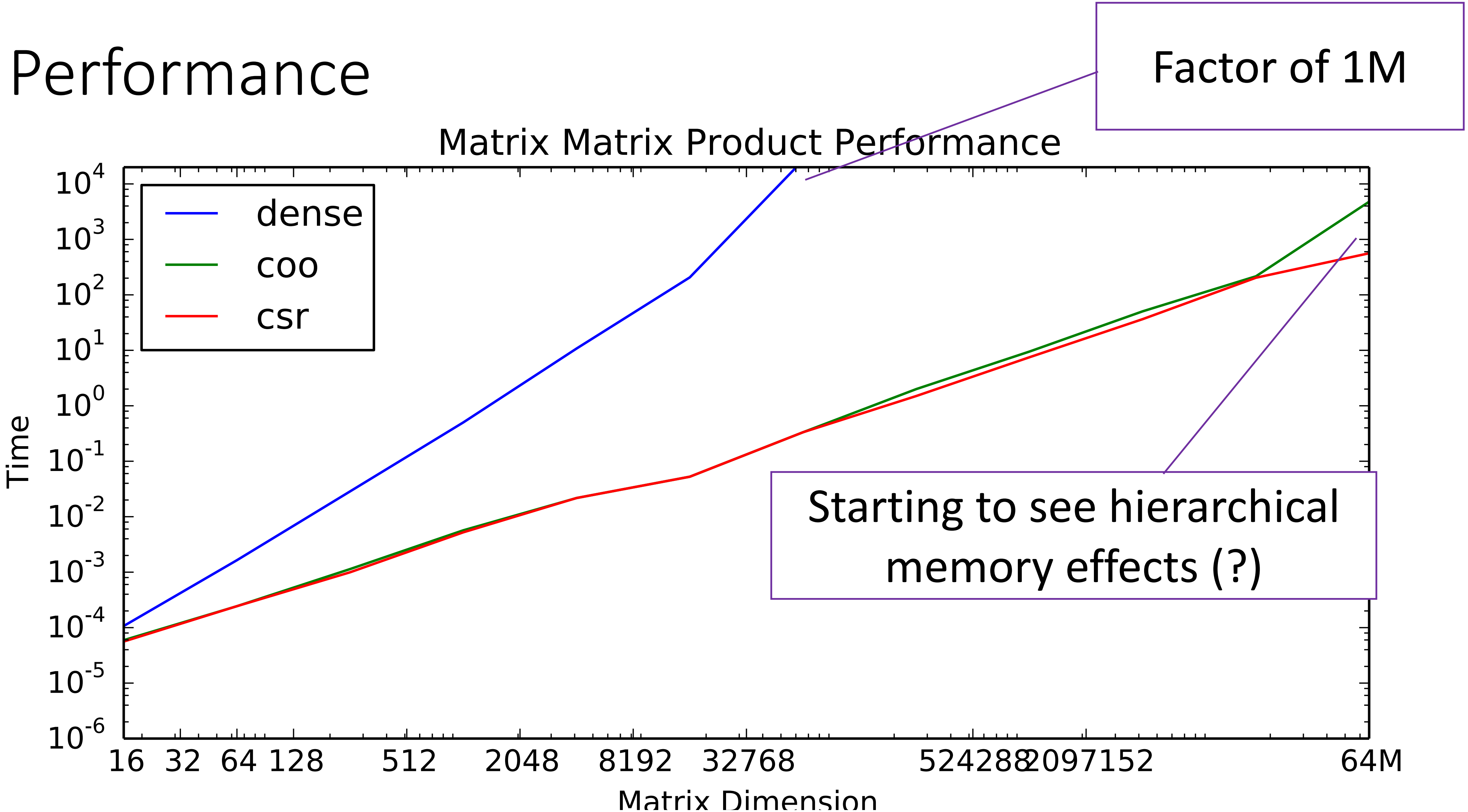
Push elements back (similar to COO)

Accumulate run row lengths

Push column index and value

Rows ***must*** be added in order and contiguously

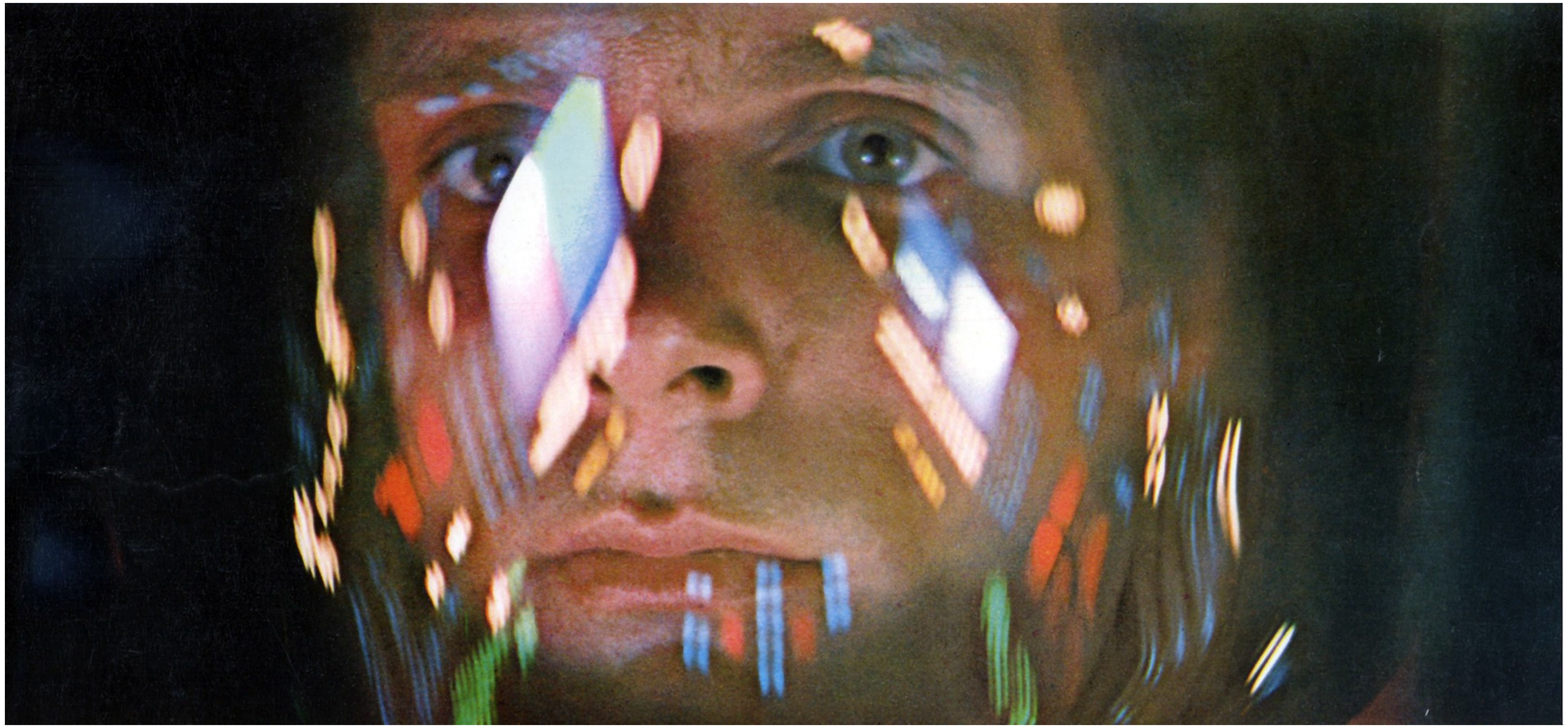
Performance



Review

- Explored variety of techniques for matching algorithm structure to hardware performance features (work smarter)
  - And we pushed this pretty far
- Strassen’s algorithm (work way smarter)
- Sparse matrix representations and algorithms (don’t do work you don’t have to do)
- Get help

Last Chance for Questions Before we Leave the Single Core World



Multicore for HPC

- How do multicore chips operate (how does the hardware work)?
- How do they get high performance?
- How does the software exploit the hardware (how do we write our software to exploit the hardware)?
- What are the abstractions that we need to use to reason about multicore systems?
- What are the programming abstractions and mechanisms?
- Terminology: Program, process, thread
- More terminology: Parallel, concurrent, asynchronous

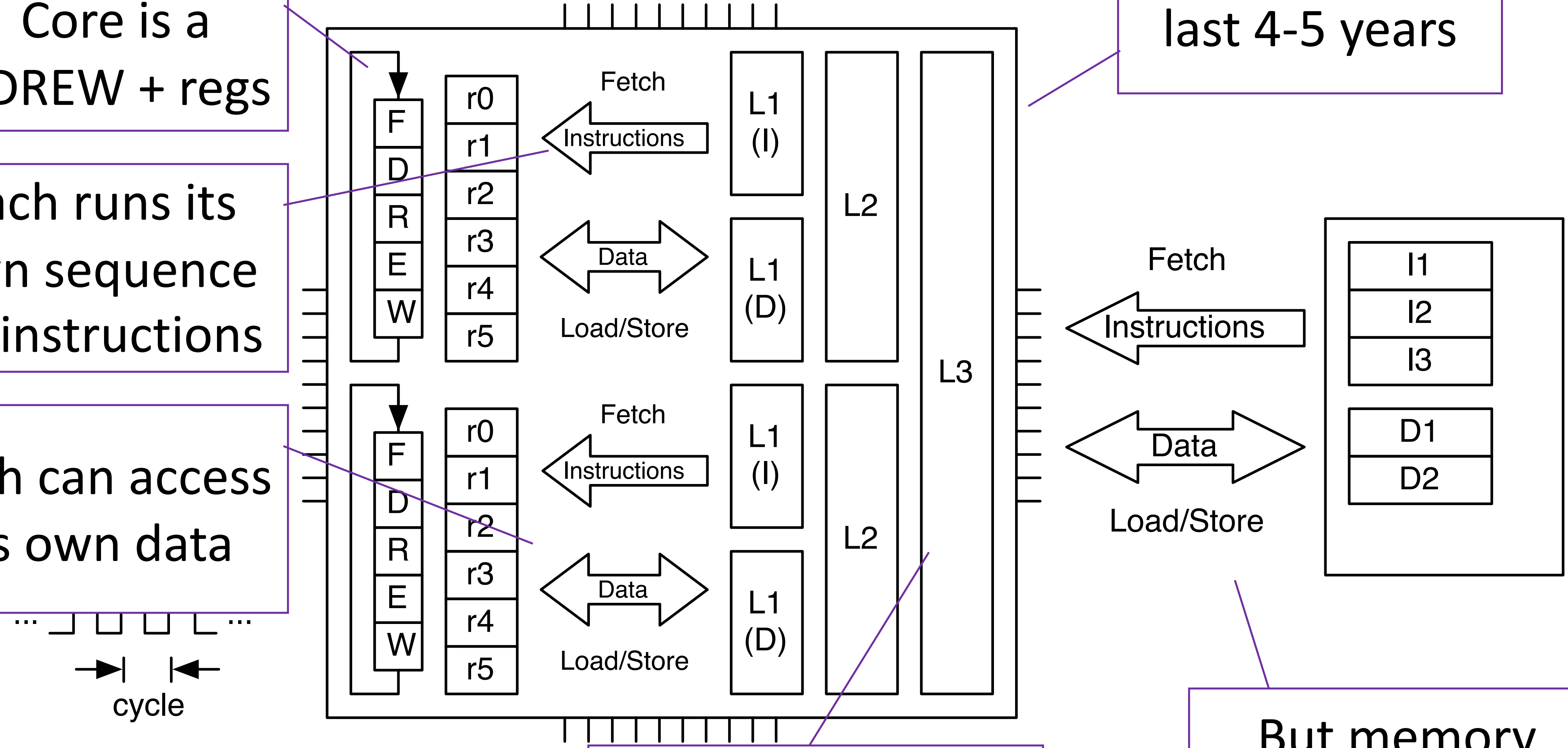
Multicore Architecture

Core is a FDREW + regs

Each runs its own sequence of instructions

Each can access its own data

Any CPU in the last 4-5 years



Each has memory hierarchy

But memory might be shared

Parallelization Example

- You are the TA for AMATH 483 and have to grade 22 exams
- The exam has 8 questions on it
- It takes 3 minutes to grade one question

- How long will it take you to grade all of the exams?

Parallelization Example

- You are the TA for AMATH 483 and have to grade 22 exams
- The exam has 8 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach
- List your assumptions

Parallelization Example

- You are the TA for AMATH 483 and have to grade 1012 exams (1012 = 46 * 22)
- The exam has 8 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach
- Describe another approach
- List your assumptions

Parallelization Example

- You are the TA for AMATH 483 and have to grade 8 exams
- The exam has 22 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- Describe your approach

Parallelization Example

- You are the TA for AMATH 483 and have to grade 368 exams (368 = 46 * 8)
- The exam has 22 questions on it
- It takes 3 minutes to grade one question
- You ask 21 friends who agree to help you

- How long will it take the 22 of you to grade all of the exams?

- What if you had 368 friends? 368*22?

Compare And Contrast

- Time for everyone grades one exam
- Time for everyone grades one question

- How (why) did you use the approaches you did?

Thank you!

Creative Commons BY-NC-SA 4.0 License