Ordinary Differential Equations and Partial Differential Equations: Solutions and Parallelism

UCSB CS240A

Tao Yang

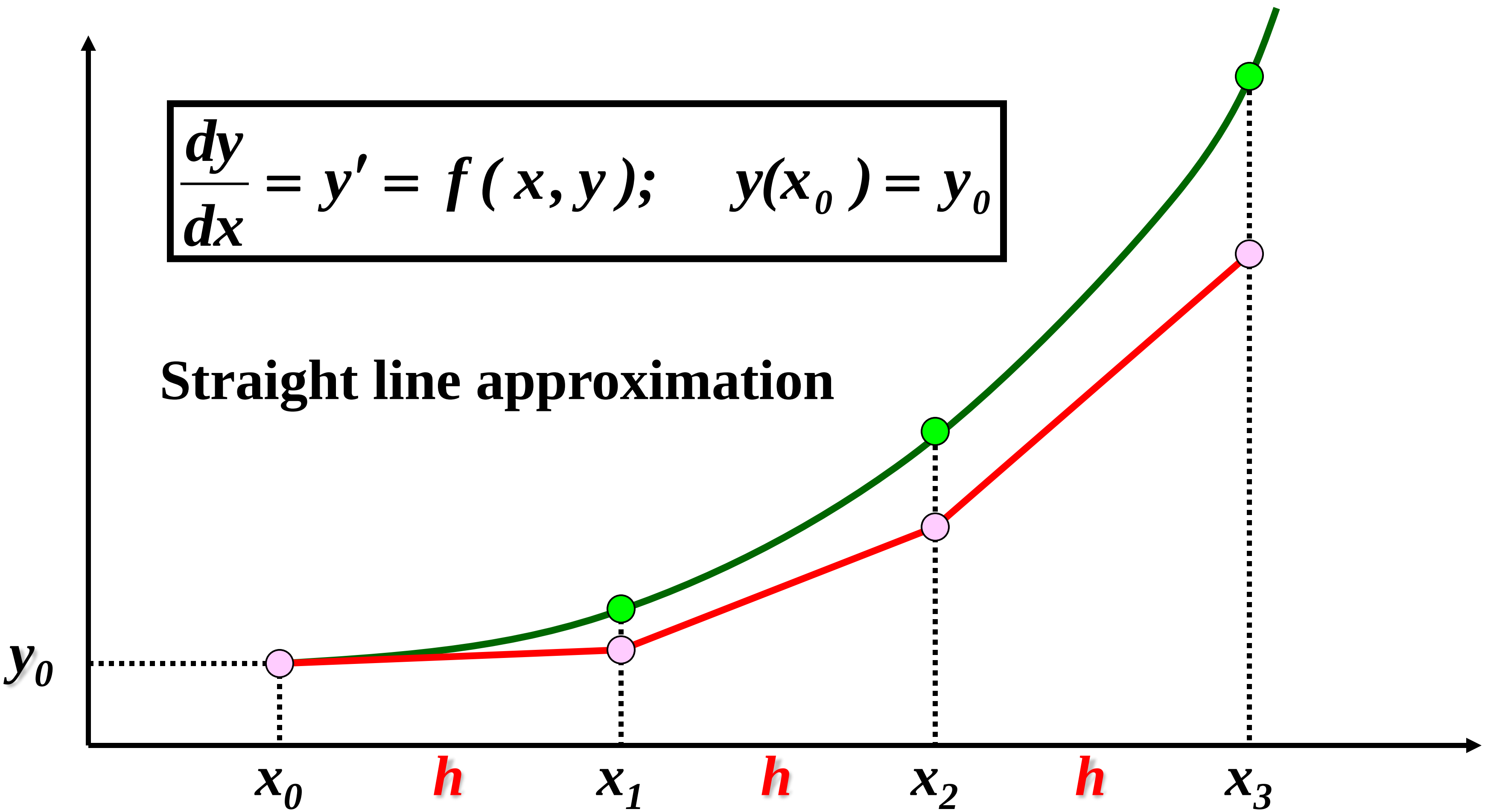
Some slides are from K. Yalick/Demmel

Finite-Difference Method for ODE/PDE

- Discretize domain of a function
- For each point in the discretized domain, name it with a variable, setup equations.
- The unknown values of those points form equations. Then solve these equations

Euler's method for ODE Initial-Value Problems

$$\frac{dy}{dx} = y' = f(x, y); \quad y(x_0) = y_0$$



Euler Method

Approximate: $y'(x_0) \approx (y(x+\Delta h) - y(x_0)) / \Delta h$

Then: $$y_{\mathrm{n+1}} = y_{\mathrm{n}} + \Delta h\, y_{\mathrm{n}}{}' + O(\Delta h^2)$$

$$y_{\mathrm{n+1}} = y_{\mathrm{n}} + \Delta h\, \mathrm{f}(x_{\mathrm{n}}, y_{\mathrm{n}}) + O(\Delta h^2)$$

Thus starting from an initial value y_0

$$y_{\mathrm{n+1}} \approx y_{\mathrm{n}} + \Delta h\, \mathrm{f}(x_{\mathrm{n}}, y_{\mathrm{n}}) \quad \textit{with} \;\; O(\Delta h^2) \;\; \text{error}$$

Euler's Method: Example

$$\frac{dy}{dx} = x + y \qquad y(0) = 1$$

$$y_{n+1} \approx y_n + \Delta h \, f(x_n, y_n) = y_n + \Delta h \,(x_n + y_n)$$

$$\Delta h = 0.02$$

x_n	y_n	y'_n	hy'_n	Exact Solution	Error
0	1.00000	1.00000	0.02000	1.00000	0.00000
0.02	1.02000	1.04000	0.02080	1.02040	-0.00040
0.04	1.04080	1.08080	0.02162	1.04162	-0.00082
0.06	1.06242	1.12242	0.02245	1.06367	-0.00126
0.08	1.08486	1.16486	0.02330	1.08657	-0.00171
0.1	1.10816	1.20816	0.02416	1.11034	-0.00218
0.12	1.13232	1.25232	0.02505	1.13499	-0.00267
0.14	1.15737	1.29737	0.02595	1.16055	-0.00318
0.16	1.18332	1.34332	0.02687	1.18702	-0.00370
0.18	1.21019	1.39019	0.02780	1.21443	-0.00425
0.2	1.23799	1.43799	0.02876	1.24281	-0.00482

ODE with boundary value

$$\frac{d^2u}{dr^2}+\frac{1}{r}\frac{d\ u}{dr}-\frac{u}{r^2}=0$$

$$u(5)=0.0038731'',$$
$$u(8)=0.0030769''$$

Solution

Using the approximation of

$$\frac{d^2y}{dx^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2} \quad \text{and} \quad \frac{dy}{dx} \approx \frac{y_{i+1} - y_{i-1}}{2(\Delta x)}$$

Gives you

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta r)^2} + \frac{1}{r_i}\frac{u_{i+1} - u_{i-1}}{2(\Delta r)} - \frac{u_i}{r_i^{\,2}} = 0$$

$$\left(-\frac{1}{2r_i(\Delta r)} + \frac{1}{(\Delta r)^2}\right)u_{i-1} + \left(-\frac{2}{(\Delta r)^2} - \frac{1}{r_i^{\,2}}\right)u_i + \left(\frac{1}{(\Delta r)^2} + \frac{1}{2r_i\Delta r}\right)u_{i+1} = 0$$

Solution Cont

Step 1 At node $i = 0, r_0 = a = 5$

$$u_0 = 0.0038731$$

Step 2 At node $i = 1,\ r_1 = r_0 + \Delta r = 5 + 0.6 = 5.6''$

$$\left(-\frac{1}{2(5.6)(0.6)} + \frac{1}{(0.6)^2}\right)u_0 + \left(-\frac{2}{(0.6)^2} - \frac{1}{(5.6)^2}\right)u_1 + \left(\frac{1}{0.6^2} + \frac{1}{2(5.6)(0.6)}\right)u_2 = 0$$

$$2.6290u_0 - 5.5874u_1 + 2.9266u_2 = 0$$

Step 3 At node $i = 2,\quad r_2 = r_1 + \Delta r = 5.6 + 0.6 = 6.2$

$$\left(-\frac{1}{2(6.2)(0.6)} + \frac{1}{0.6^2}\right)u_1 + \left(-\frac{2}{0.6^2} - \frac{1}{6.2^2}\right)u_2 + \left(\frac{1}{0.6^2} + \frac{1}{2(6.2)(0.6)}\right)u_3 = 0$$

$$2.6434u_1 - 5.5816u_2 + 2.9122u_3 = 0$$

Solution Cont

Step 4 At node $i = 3, \quad r_3 = r_2 + \Delta r = 6.2 + 0.6 = 6.8$

$$\left(-\frac{1}{2(6.8)(0.6)} + \frac{1}{0.6^2}\right)u_2 + \left(-\frac{2}{0.6^2} - \frac{1}{6.8^2}\right)u_3 + \left(\frac{1}{0.6^2} + \frac{1}{2(6.8)(0.6)}\right)u_4 = 0$$

$$2.6552u_2 - 5.5772u_3 + 2.9003u_4 = 0$$

Step 5 At node $i = 4, \quad r_4 = r_3 + \Delta r = 6.8 + 0.6 = 7.4$

$$\left(-\frac{1}{2(7.4)(0.6)} + \frac{1}{0.6^2}\right)u_3 + \left(-\frac{2}{0.6^2} - \frac{1}{(7.4)^2}\right)u_4 + \left(\frac{1}{0.6^2} + \frac{1}{2(7.4)(0.6)}\right)u_5 = 0$$

$$2.6651u_3 - 5.6062u_4 + 2.8903u_5 = 0$$

Step 6 At node $i = 5, \quad r_5 = r_4 + \Delta r = 7.4 + 0.6 = 8$

$$u_5 = u/_{r=b} = 0.0030769$$

Solving system of equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 2.6290 & -5.5874 & 2.9266 & 0 & 0 & 0 \\ 0 & 2.6434 & -5.5816 & 2.9122 & 0 & 0 \\ 0 & 0 & 2.6552 & -5.5772 & 2.9003 & 0 \\ 0 & 0 & 0 & 2.6651 & -5.6062 & 2.8903 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0.0038731 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.0030769 \end{bmatrix}$$

$u_0 = 0.0038731$

$u_1 = 0.0036115$

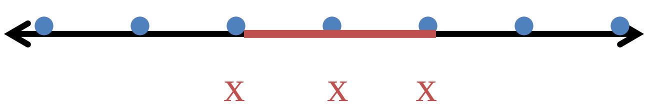
$u_2 = 0.0034159$

$u_3 = 0.0032689$

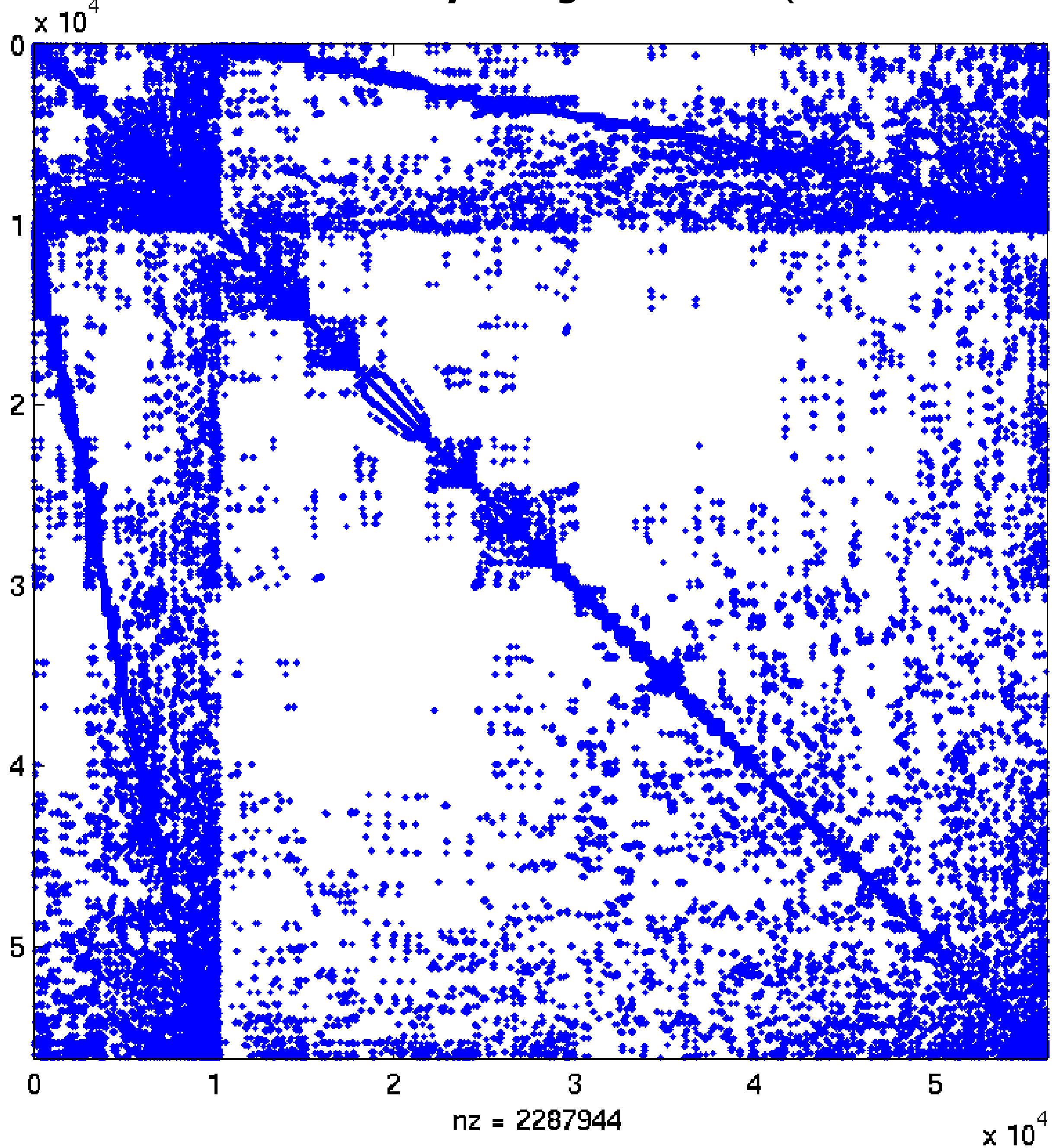
$u_4 = 0.0031586$

$u_5 = 0.0030769$

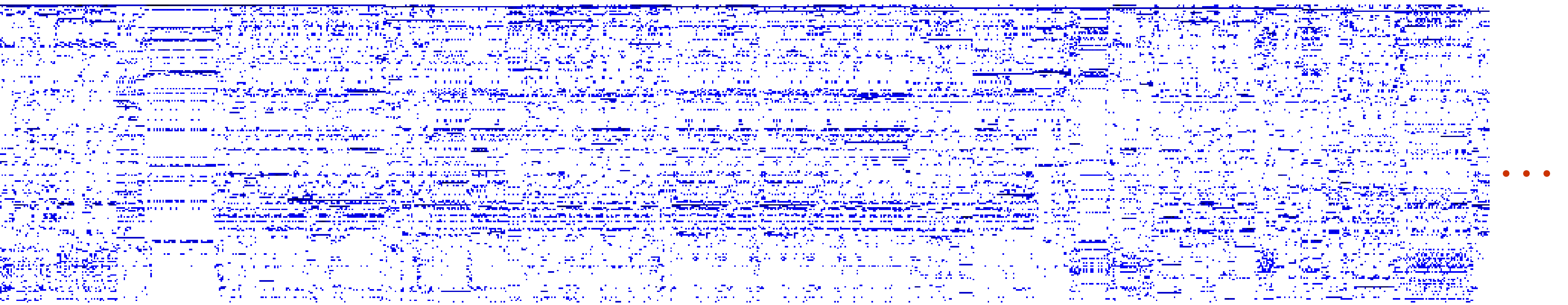
Graph and "stencil"



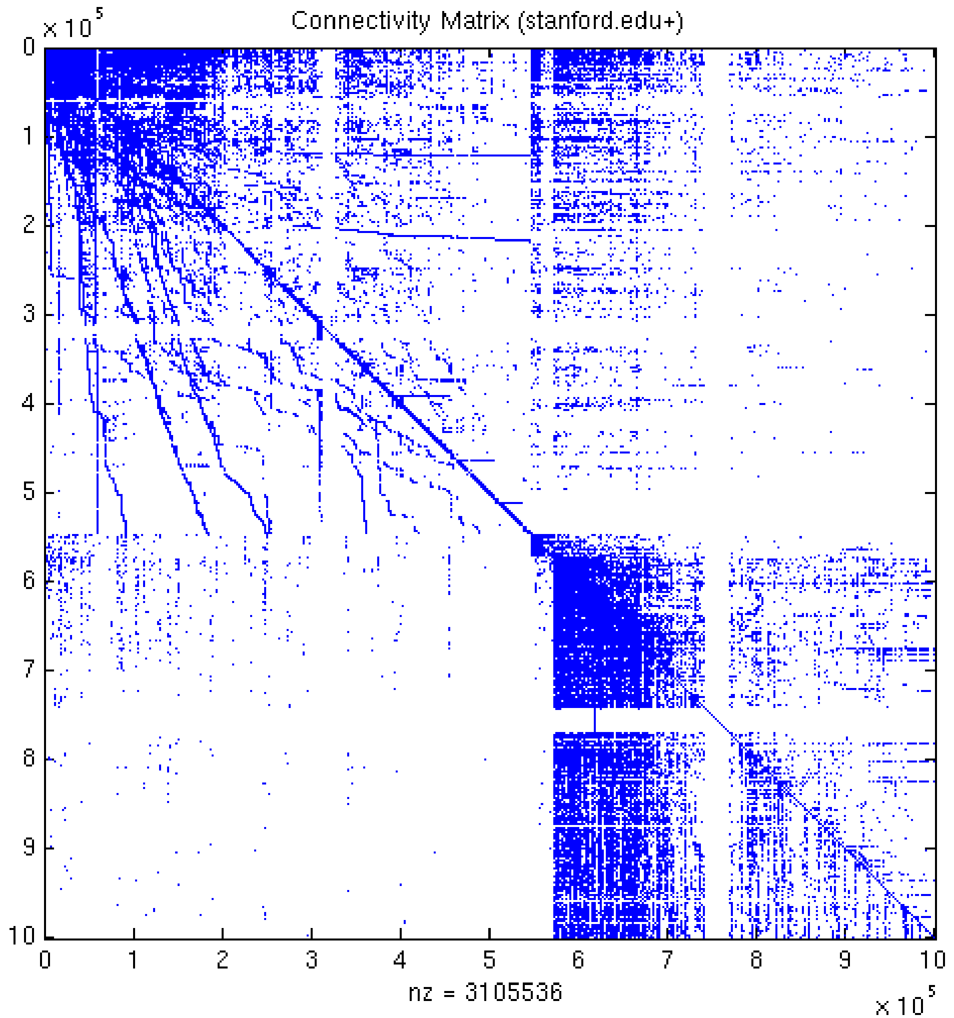
Source: Accelerator Cavity Design Problem (Ko via Husbands)



Linear Programming Matrix

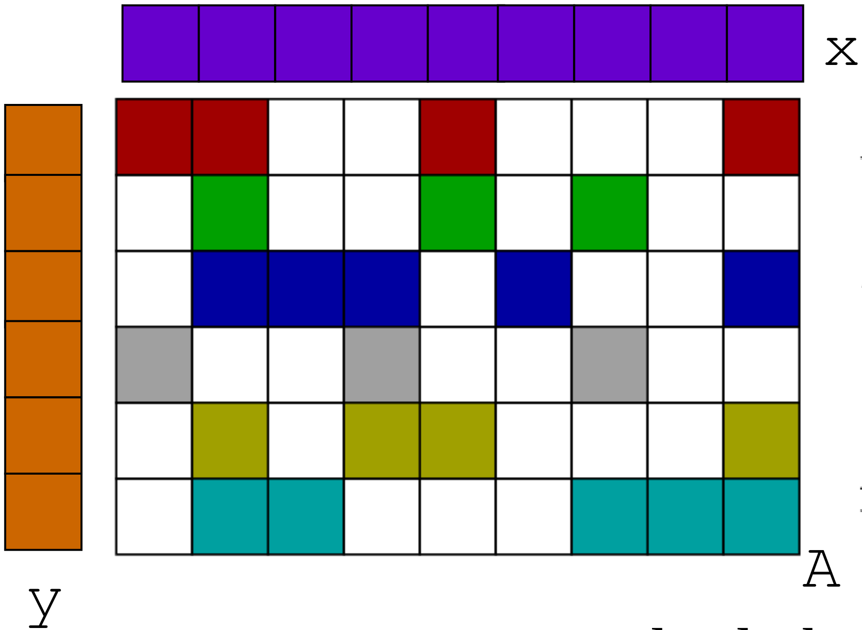


A Sparse Matrix You Encounter Every Day

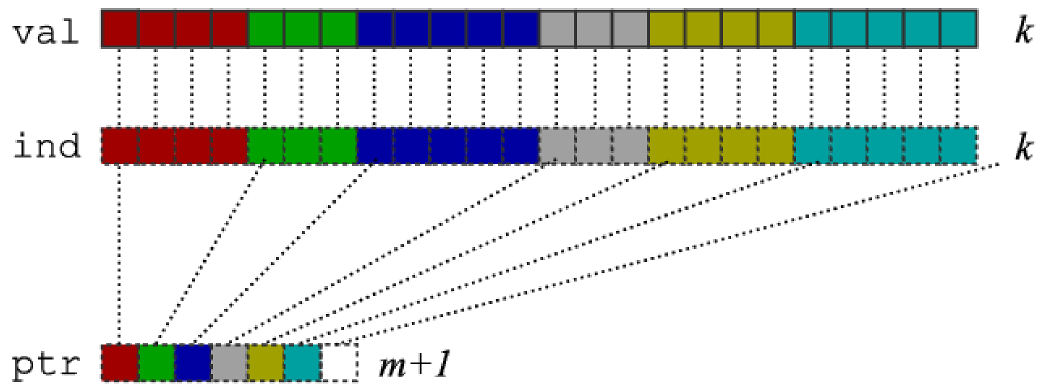


Compressed Sparse Row (CSR) Format

SpMV: y = y + A*x, only store, do arithmetic, on nonzero entries



Representation of A

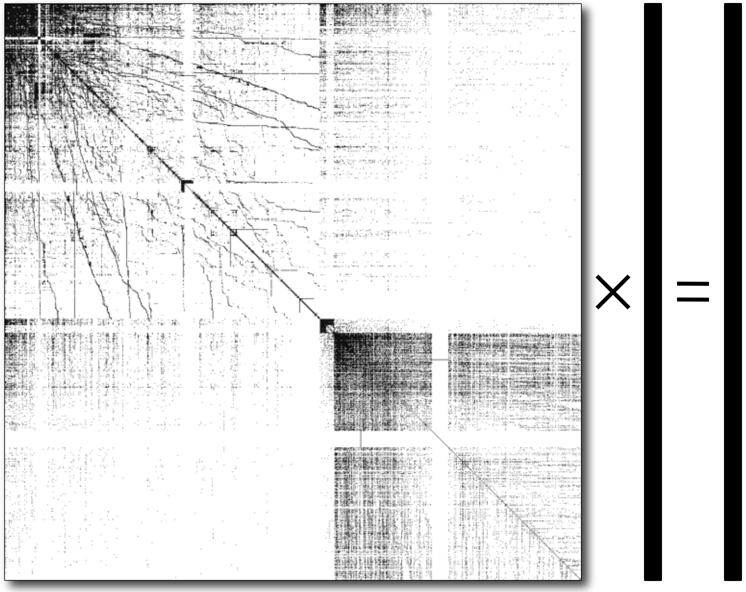


Matrix-vector multiply kernel: $y(i) \leftarrow y(i) + A(i,j) \cdot x(j)$

```
for each row i
   for k=ptr[i] to ptr[i+1]-1 do
         y[i] = y[i] + val[k]*x[ind[k]]
```

SpMV Parallelization

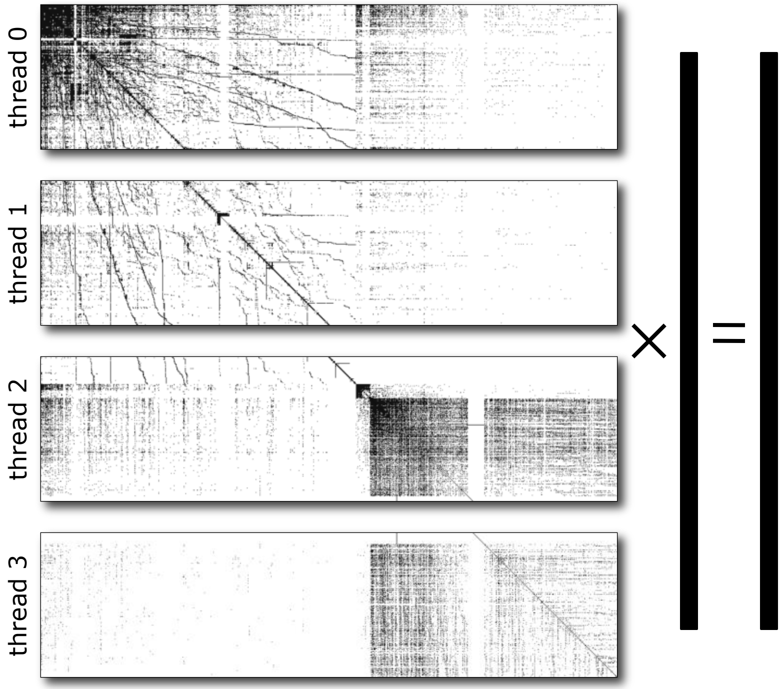
- How do we parallelize a matrix-vector multiplication ?



Source: Sam Williams

SpMV Parallelization

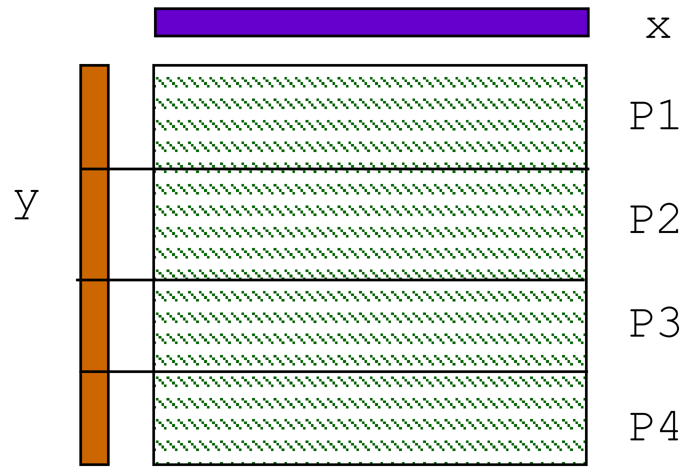
- How do we parallelize a matrix-vector multiplication ?
- By rows blocks
- No inter-thread data dependencies, but random access to x



Source: Sam Williams

Parallel Sparse Matrix-vector multiplication

- y = A*x, where A is a sparse n x n matrix



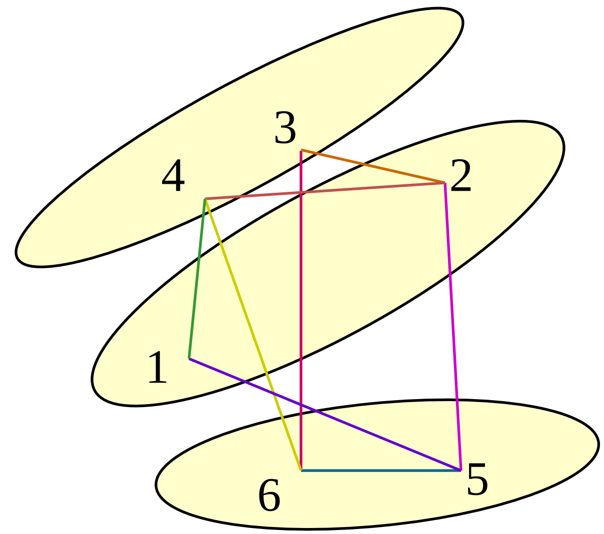
- Questions
  – which processors store
    - y[i], x[i], and A[i,j]
  – which processors compute
    - y[i] = sum (from 1 to n) A[i,j] * x[j]
      = (row i of A) * x   … a sparse dot product
- Partitioning
  – Partition index set {1,…,n} = N1 ∪ N2 ∪ … ∪ Np.
  – For all i in Nk, Processor k stores y[i], x[i], and row i of A
  – For all i in Nk, Processor k computes y[i] = (row i of A) * x
    - "owner computes" rule: Processor k compute the y[i]s it owns.

May require communication

Graph Partitioning and Sparse Matrices

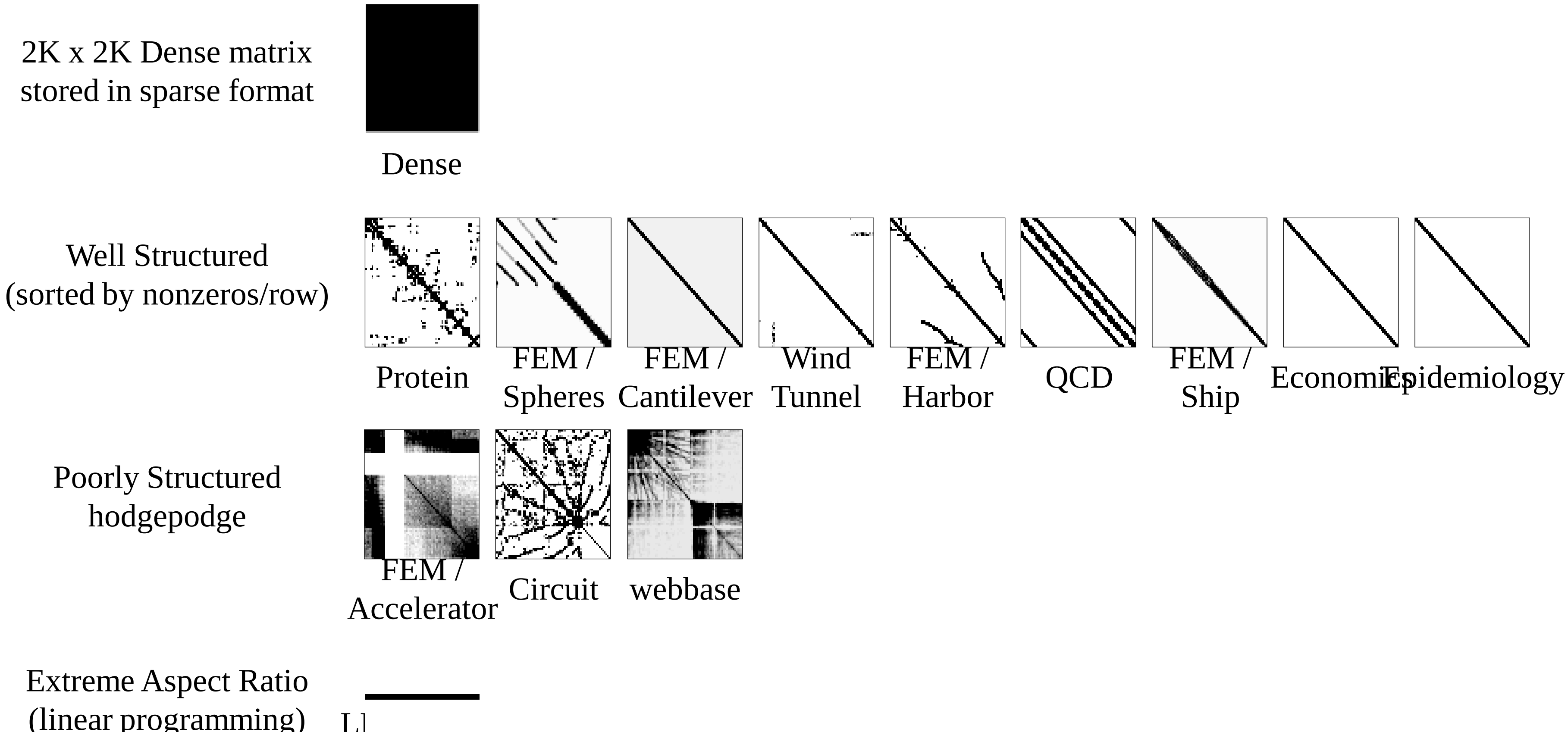
- Relationship between matrix and graph

	1	2	3	4	5	6
1	1			1	1	
2		1	1	1	1	
3		1	1			1
4	1	1		1		1
5	1	1			1	1
6			1	1	1	1



- Edges in the graph are nonzero in the matrix:
- If divided over 3 procs, there are 14 nonzeros outside the diagonal blocks, which represent the 7 (bidirectional) edges

Application matrices

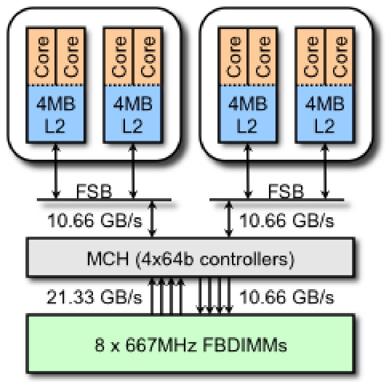


Samuel Williams, Oliker, W. Vuduc, Shalf, A. Yelick, James Demmel, **"Optimization of sparse matrix-vector multiplication on emerging multicore platforms"**, *Parallel Computing*, 2008, 35:38

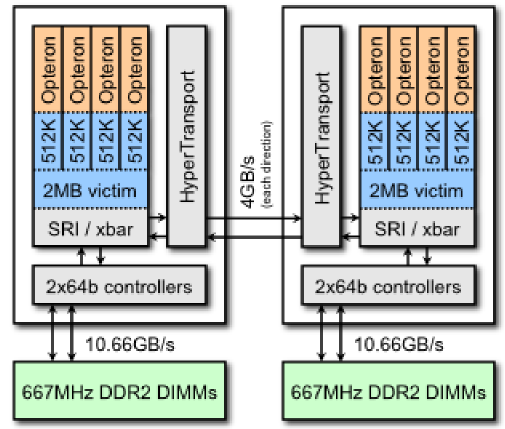
Source: Sam Williams

Multicore SMPs Used

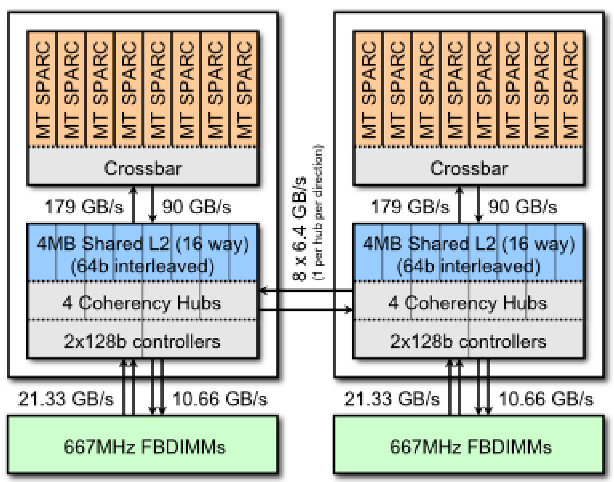
Intel Xeon E5345 (Clovertown)



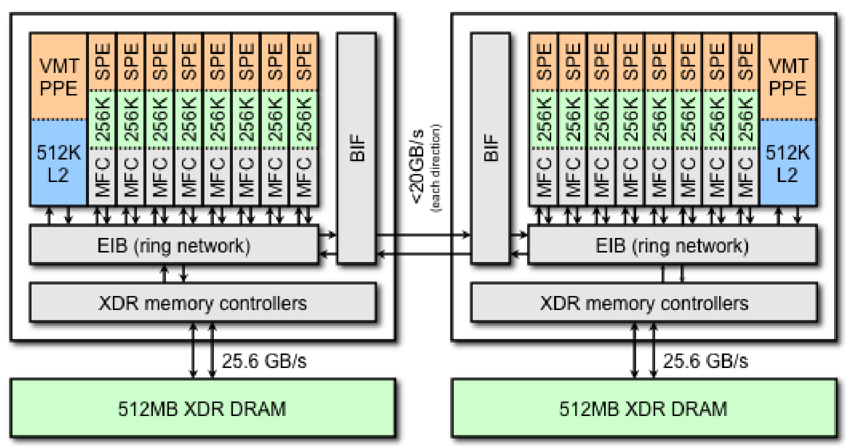
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

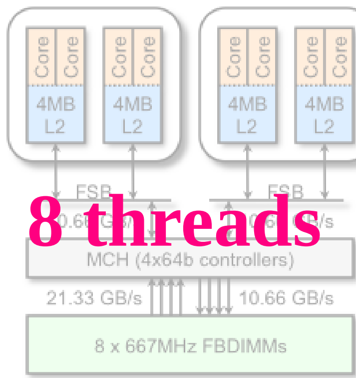


Source: Sam Williams

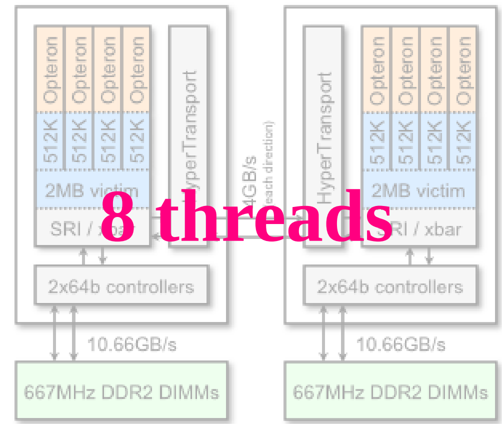
Multicore SMPs Used

(threads)

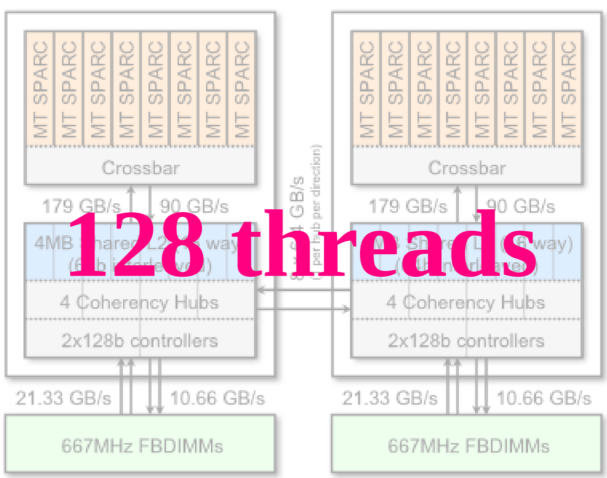
Intel Xeon E5345 (Clovertown)



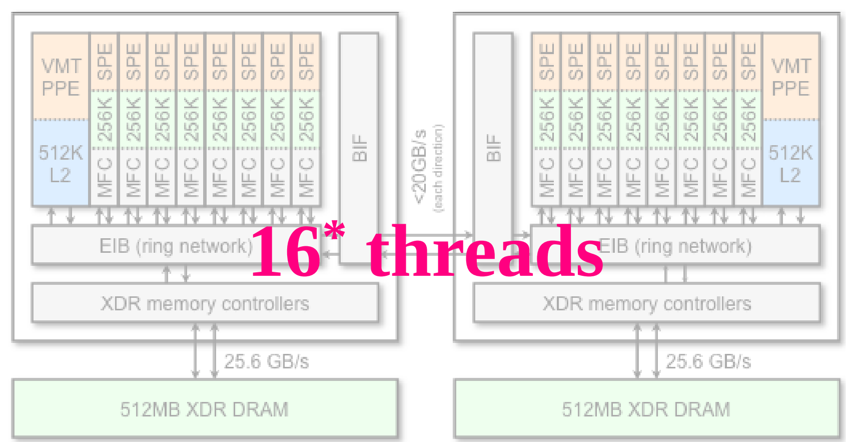
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade



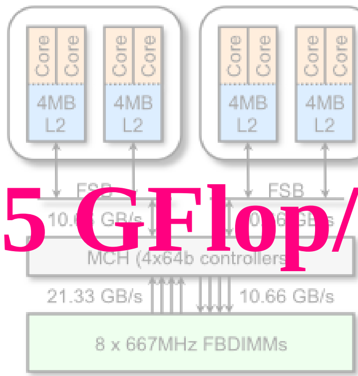
*SPEs only

Source: Sam Williams

Multicore SMPs Used

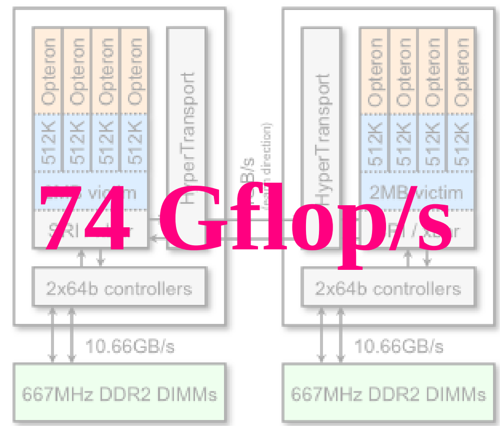
(peak double precision flops)

Intel Xeon E5345 (Clovertown)



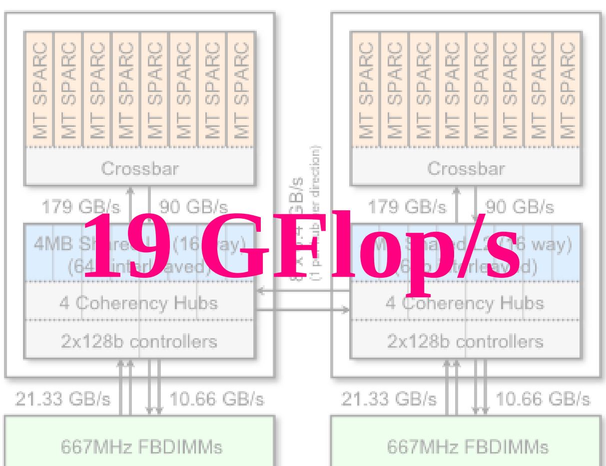
75 GFlop/s

AMD Opteron 2356 (Barcelona)



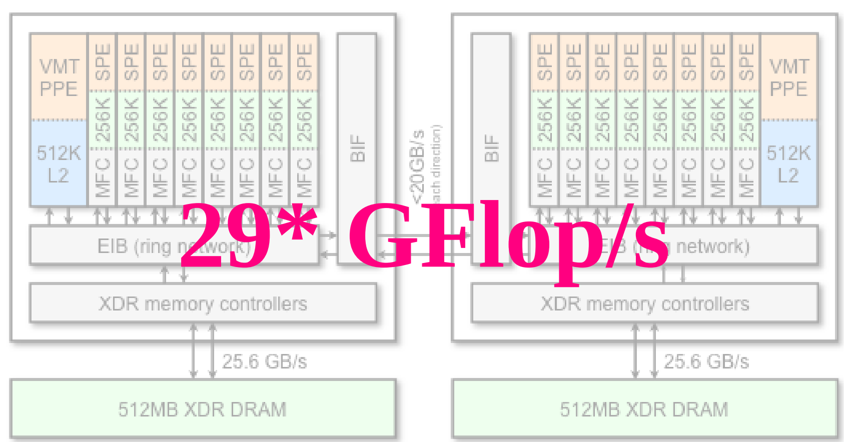
74 Gflop/s

Sun T2+ T5140 (Victoria Falls)



19 GFlop/s

IBM QS20 Cell Blade



29* GFlop/s

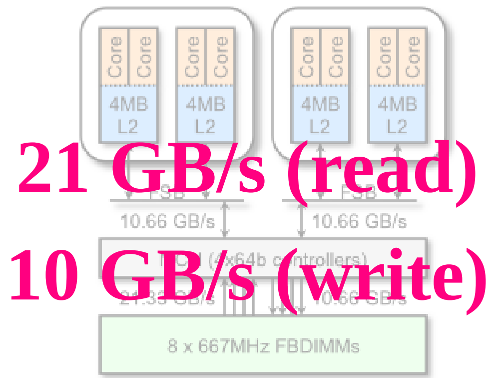
*SPEs only

Source: Sam Williams

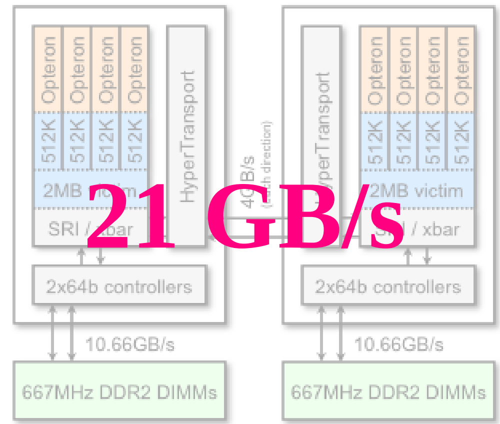
Multicore SMPs Used

(Total DRAM bandwidth)

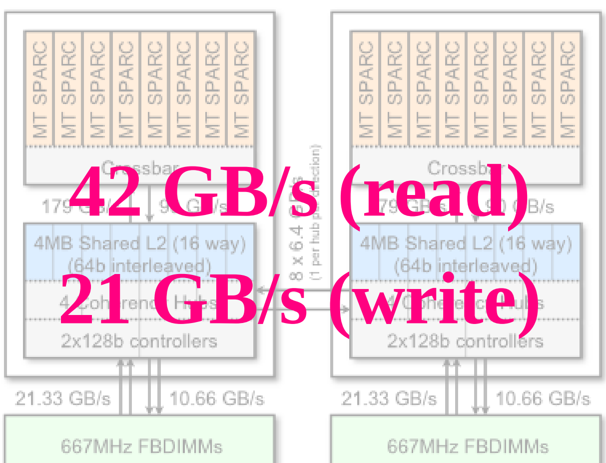
Intel Xeon E5345 (Clovertown)



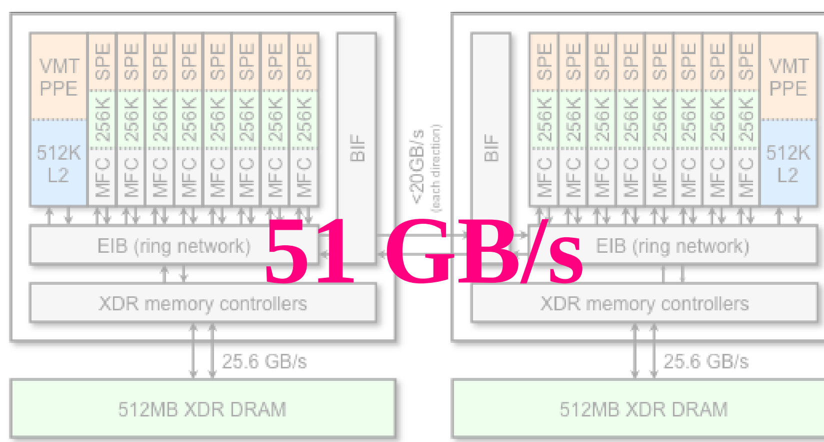
AMD Opteron 2356 (Barcelona)



Sun T2+ T5140 (Victoria Falls)



IBM QS20 Cell Blade

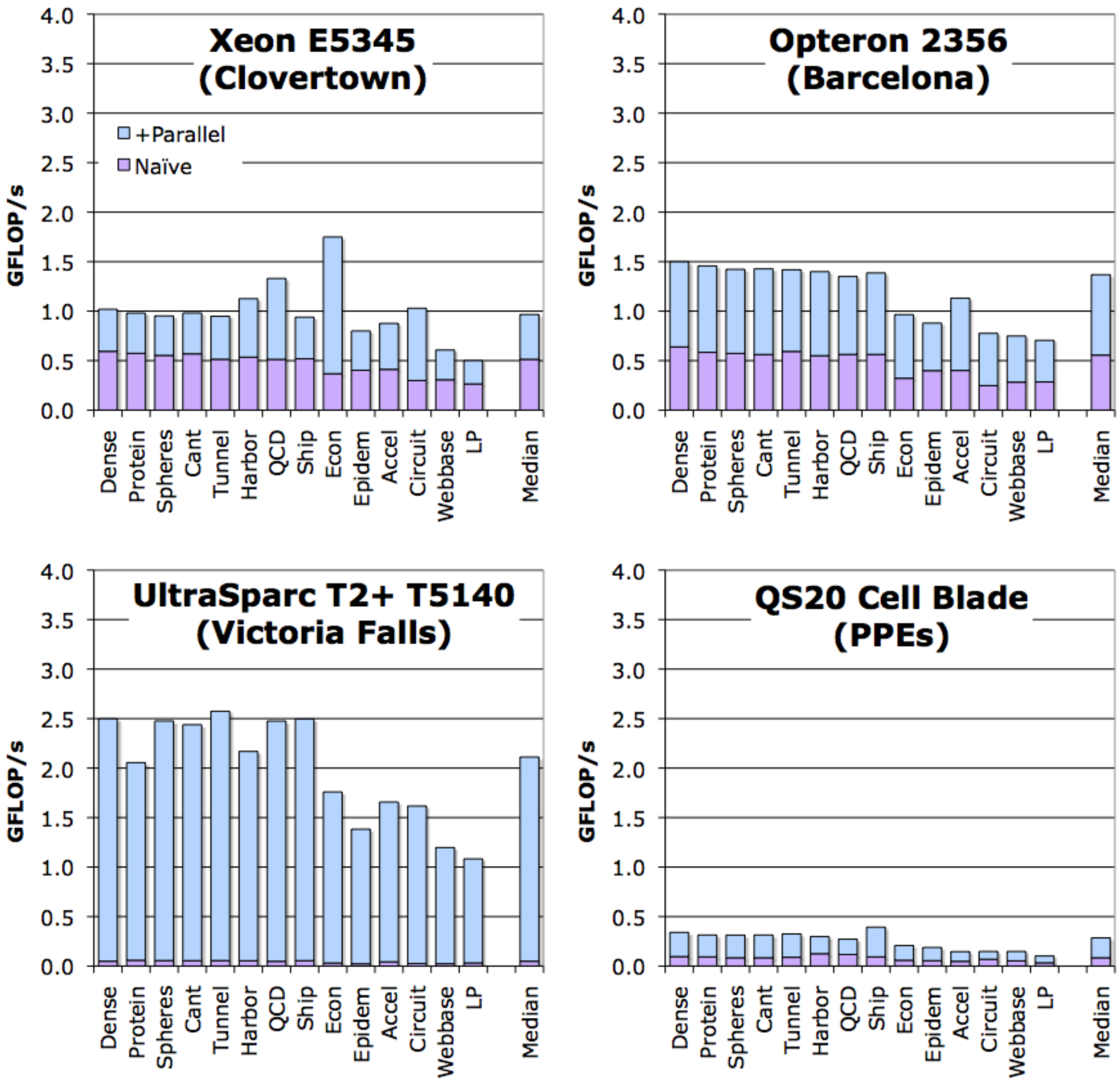


Source: Sam Williams

*SPEs only

SpMV Performance

(simple parallelization)



- Out-of-the box SpMV performance on a suite of 14 matrices
- Simplest solution = parallelization by rows (solves data dependency challenge)

- **Scalability isn't great**
- **Is this performance good?**

Naïve Pthreads

Naïve

Source: Sam Williams

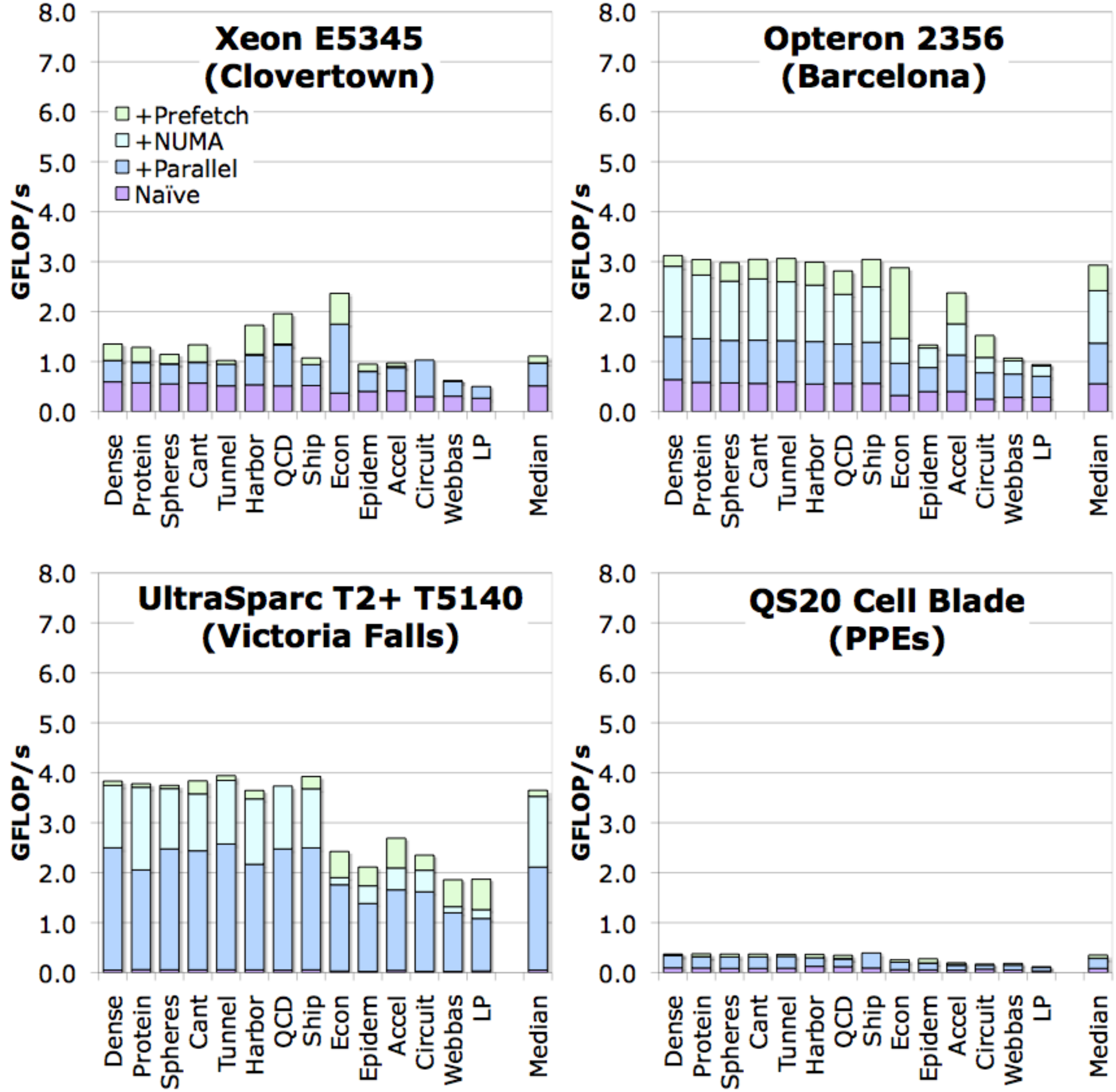
Prefetch for SpMV

- SW prefetch injects more MLP into the memory subsystem.
- Supplement HW prefetchers
- Can try to prefetch the
  - values
  - indices
  - source vector
  - *or any combination thereof*
- In general, should only insert one prefetch per cache line (works best on unrolled code)

```
for(all rows){
    y0 = 0.0;
    y1 = 0.0;
    y2 = 0.0;
    y3 = 0.0;
for(all tiles in this row){
  PREFETCH(V+i+PFDistance);
    y0+=V[i  ]*X[C[i]]
    y1+=V[i+1]*X[C[i]]
    y2+=V[i+2]*X[C[i]]
    y3+=V[i+3]*X[C[i]]
        }
    y[r+0] = y0;
    y[r+1] = y1;
    y[r+2] = y2;
    y[r+3] = y3;
       }
```

Source: Sam Williams

SpMV Performance

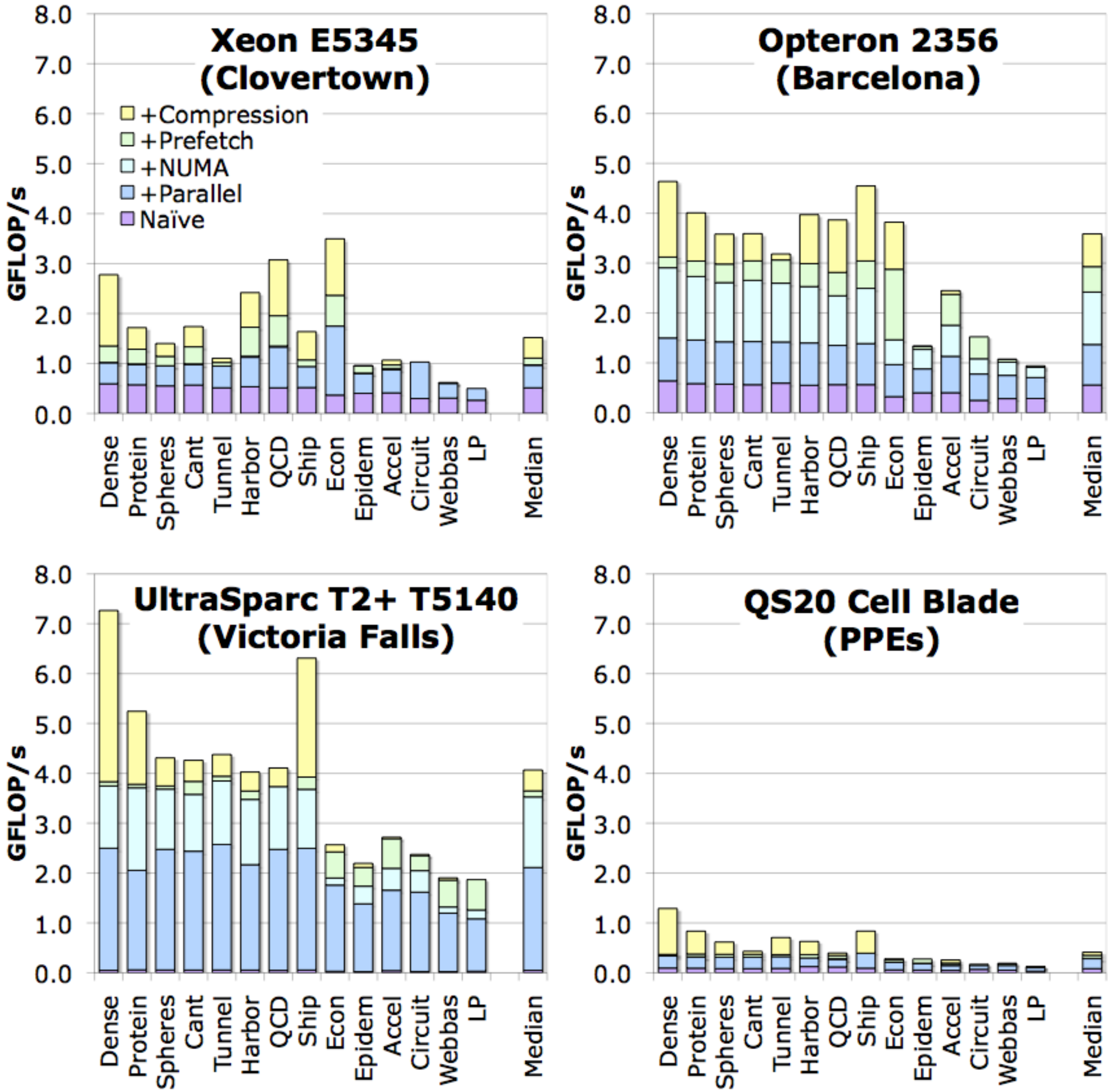


ng)

- NUMA-aware allocation is essential on memory-bound NUMA SMPs.
  - Explicit software prefetching can boost bandwidth and change cache replacement policies
  - Cell PPEs are likely latency-limited.
- used **exhaustive** search for best prefetch distance

Source: Sam Williams

SpMV Performance



- After maximizing memory bandwidth, the only hope is to minimize memory traffic.
  - exploit:
    - register blocking
    - other formats
    - smaller indices
- Use a traffic minimization **heuristic** rather than search
  - Benefit is clearly matrix-dependent.
- Register blocking enables efficient software prefetching (one per cache line)

Source: Sam Williams

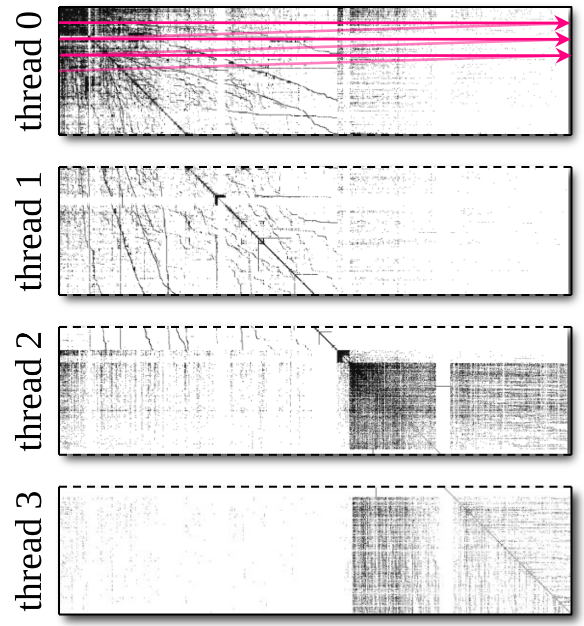
Cache blocking for SpMV

(Data Locality for Vectors)

- Store entire submatrices contiguously

- The columns spanned by each cache block are selected to use same space in cache, i.e. access same number of x(i)

- TLB blocking is a similar concept but instead of on 8 byte granularities, it uses 4KB granularities

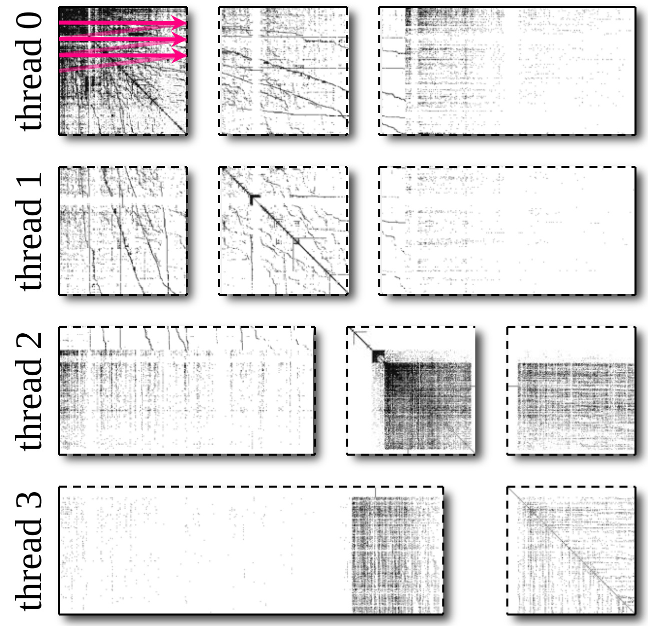


Source: Sam Williams

Cache blocking for SpMV

(Data Locality for Vectors)

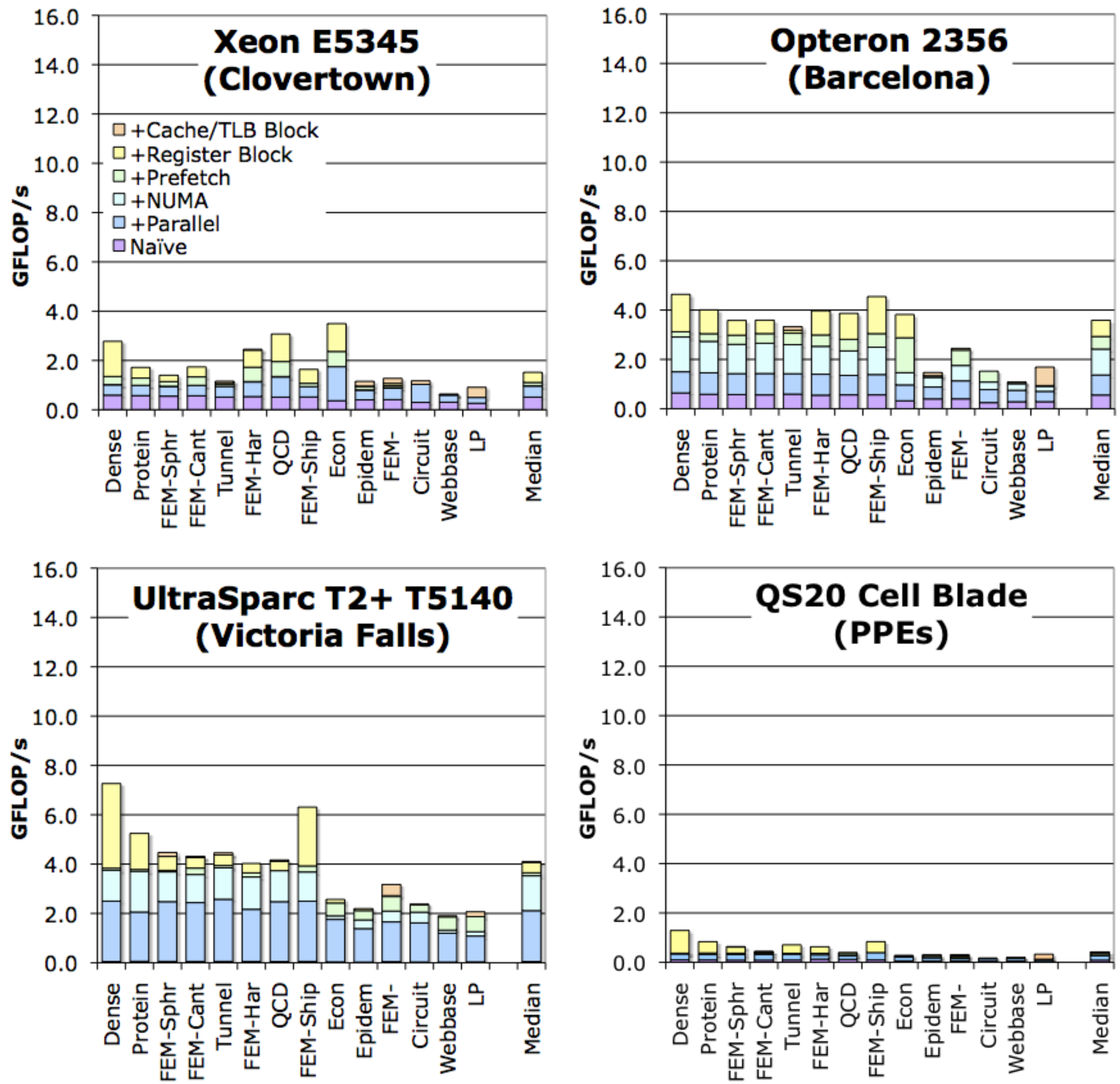
- Store entire submatrices contiguously
- The columns spanned by each cache block are selected to use same space in cache, i.e. access same number of x(i)
- TLB blocking is a similar concept but instead of on 8 byte granularities, it uses 4KB granularities



Source: Sam Williams

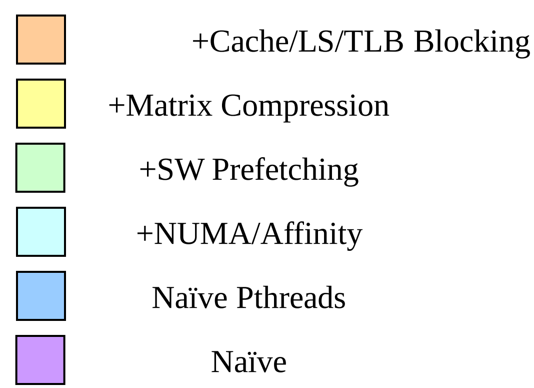
Auto-tuned SpMV Performance

(cache and TLB blocking)



- Fully auto-tuned SpMV performance across the suite of matrices
- Why do some optimizations work better on some architectures?

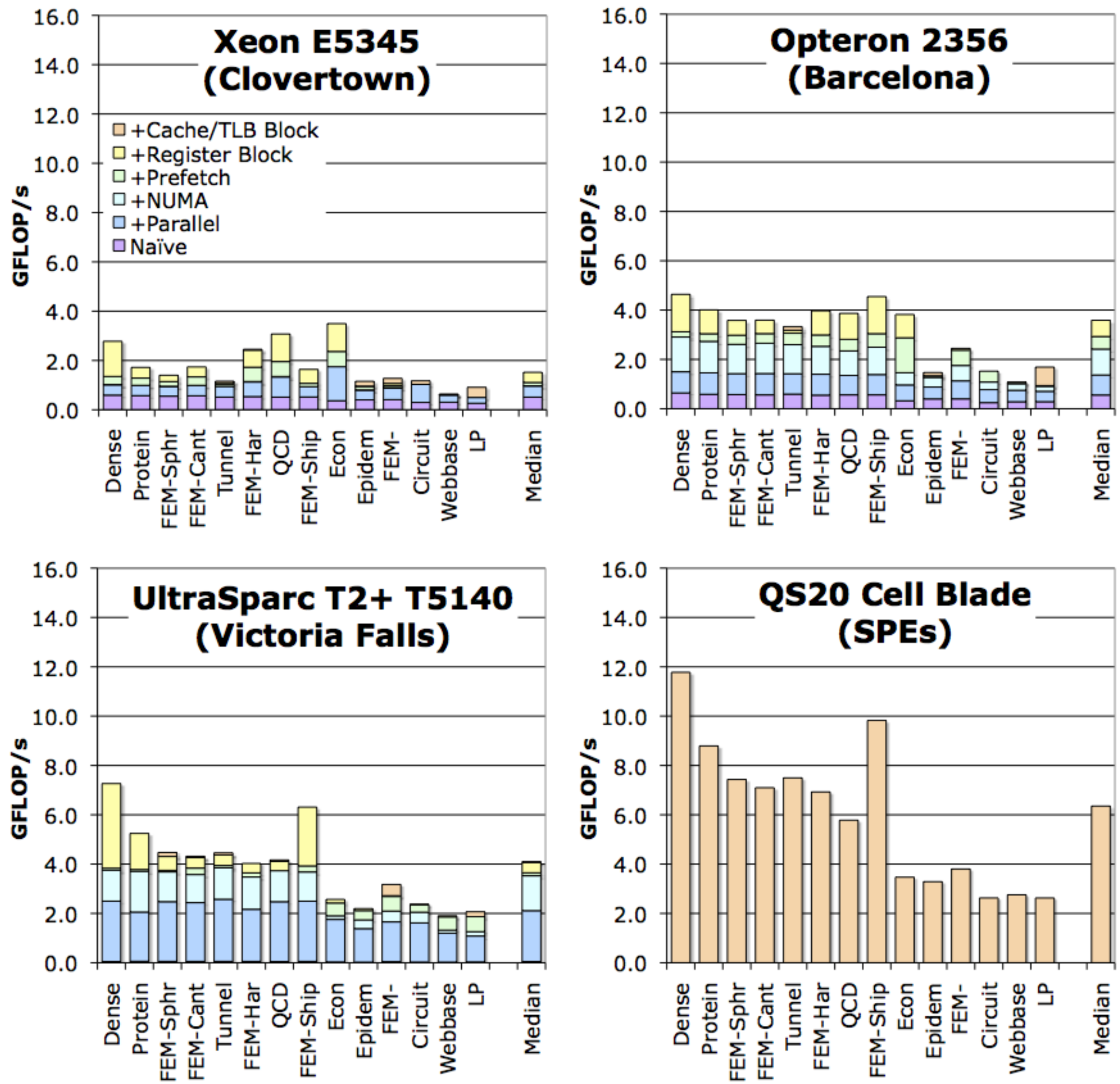
- **matrices with naturally small working sets**
- **architectures with giant caches**



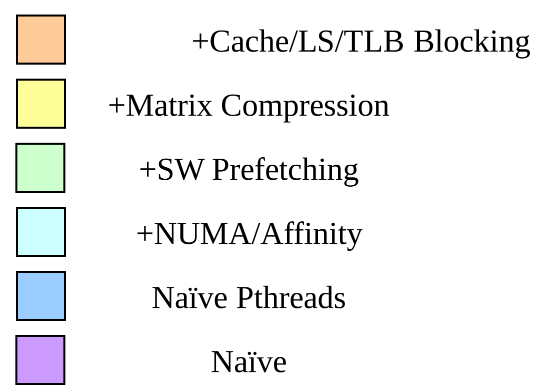
Source: Sam Williams

Auto-tuned SpMV Performance

(architecture specific optimizations)



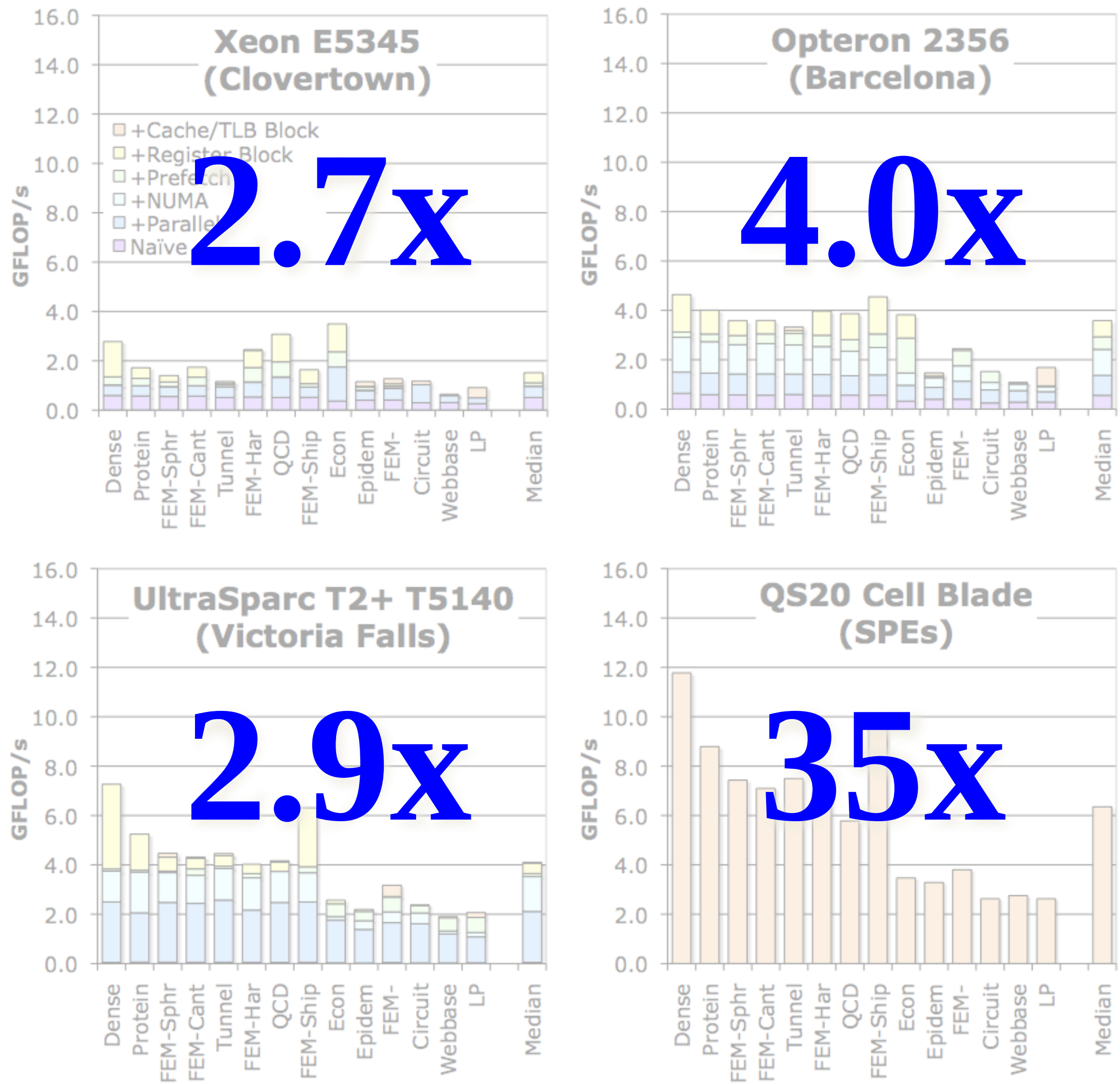
- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?



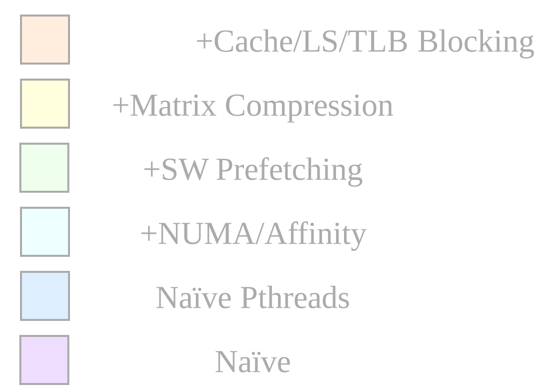
Source: Sam Williams

Auto-tuned SpMV Performance

(max speedup)



- Fully auto-tuned SpMV performance across the suite of matrices
- Included SPE/local store optimized version
- Why do some optimizations work better on some architectures?



Source: Sam Williams

Solving PDEs

- Finite element method
- Finite difference method (our focus)
  - Converts PDE into matrix equation
    - Linear system over discrete basis elements
  - Result is usually a sparse matrix

Class of Linear Second-order PDEs

- Linear second-order PDEs are of the form

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Eu_x + Fu_y + Gu = H$$

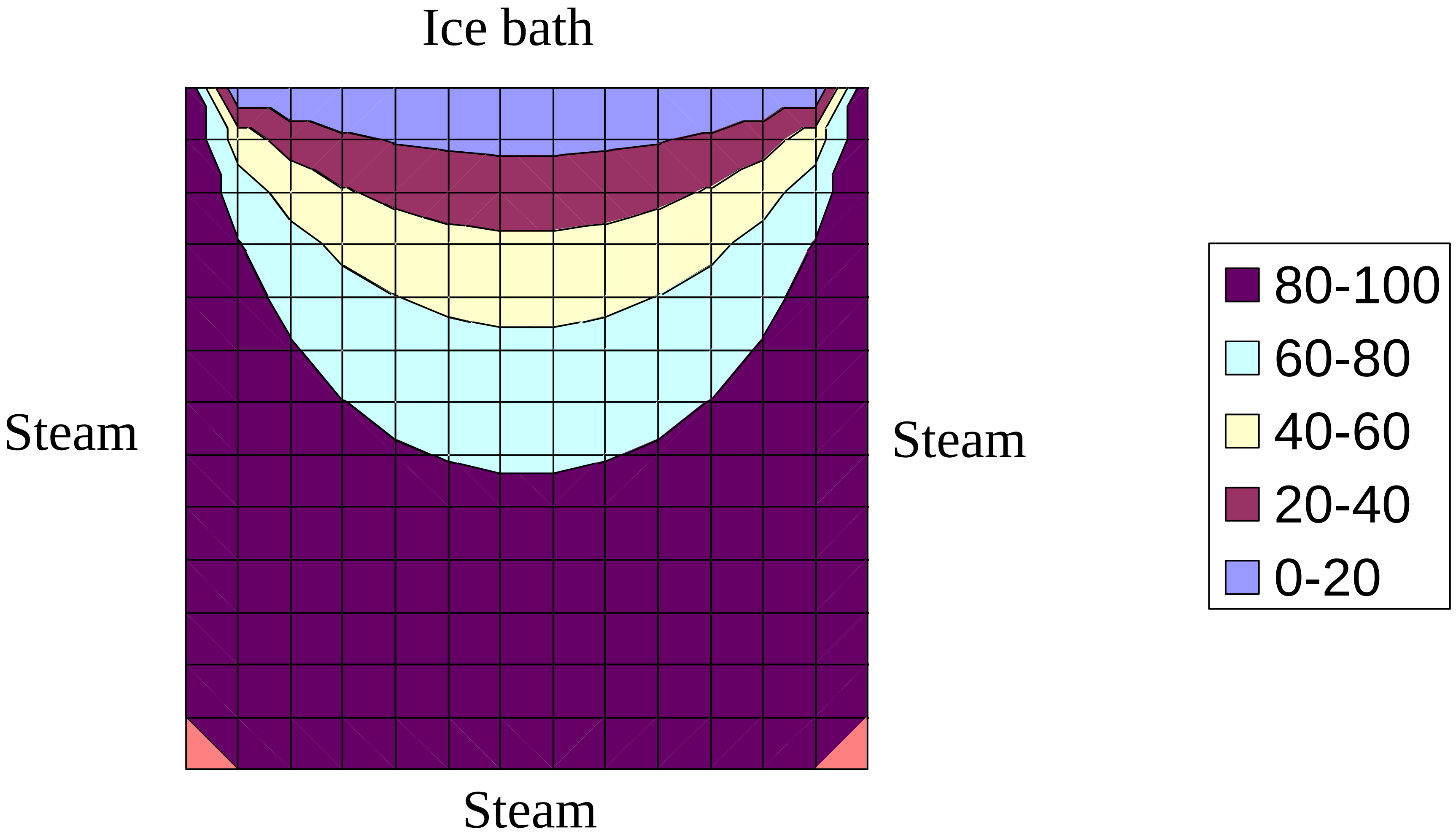
  where *A* - *H* are functions of *x* and *y* only
- Elliptic PDEs: $B^2 - AC < 0$

  (steady state heat equations)
- Parabolic PDEs: $B^2 - AC = 0$

  (heat transfer equations)
- Hyperbolic PDEs: $B^2 - AC > 0$

  (wave equations)

PDE Example: Steady State Heat Distribution Problem



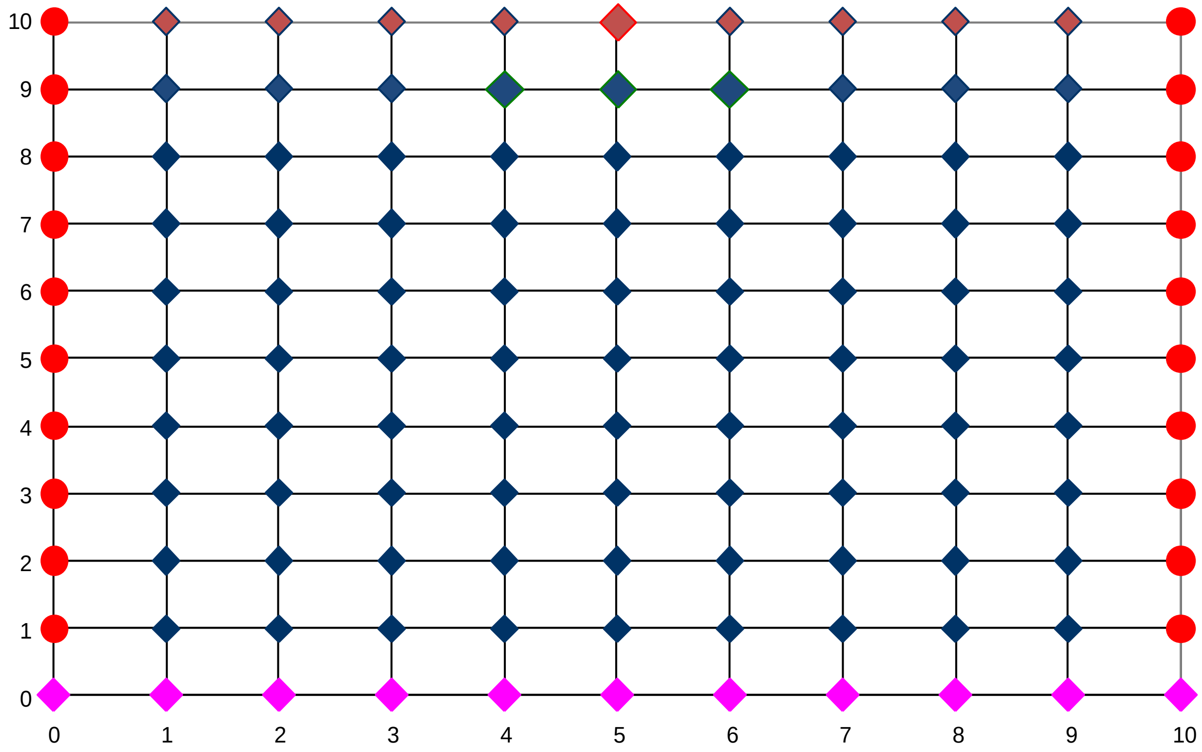
Solving the Problem

- Underlying PDE is the Poisson equation

$$u_{xx} + u_{yy} = f(x, y)$$

- This is an example of an elliptical PDE
- Will create a 2-D grid
- Each grid point represents value of state state solution at particular (*x, y*) location in plate

Discrete 2D grid space



$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Finite-difference

- **Special case: f(x,y)=0**

$$\frac{1}{h^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

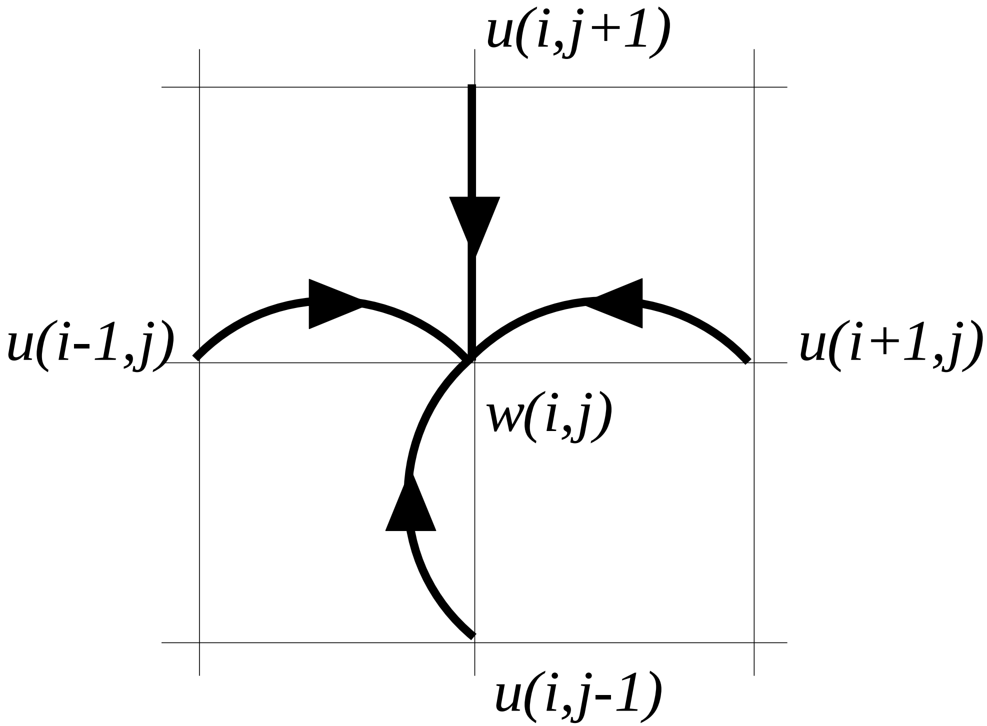
$$+\frac{1}{h^2}(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) = 0$$

- **Namely**

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

Heart of Iterative Program

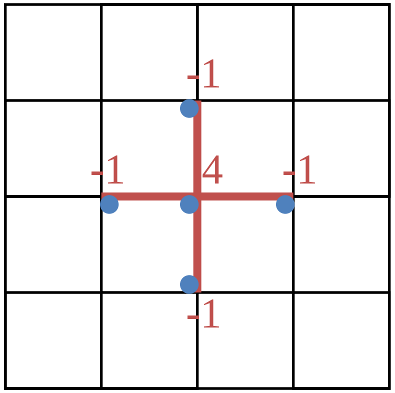
```
w[i][j] = (u[i-1][j] + u[i+1][j] +
          u[i][j-1] + u[i][j+1]) / 4.0;
```



Matrx vs. graph representation

$$L = \left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)$$

Dependence: "5 point stencil"



3D case is analogous (7 point stencil)

Jacobi method

- Jacobi method allows full parallelism, but slower convergence

Repeat

  For i=1 to n

    For j=1 to n

$$u_{i,j}^{new} = 0.25(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}).$$

    EndFor

  EndFor

Until $\| u_{ij}^{new} - u_{ij} \| < \epsilon$

Gauss-Seidel Method

- Faster convergence

Repeat

$u^{old} = u.$

For i=1 to n

For j=1 to n

$$u_{i,j} = 0.25(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}).$$

EndFor

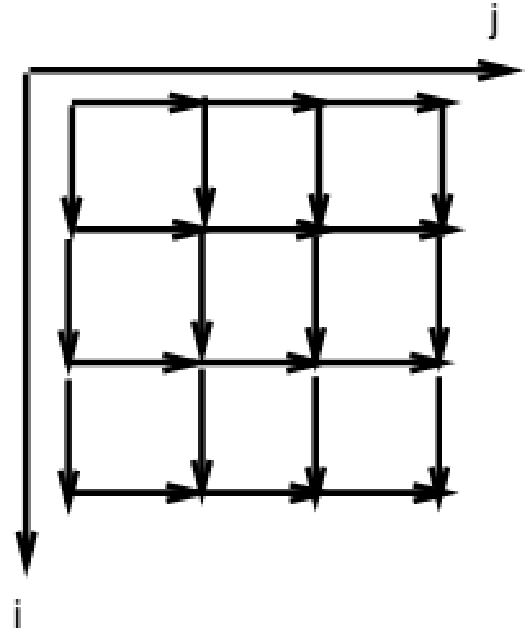
EndFor

Until $\| u_{ij} - u_{ij}^{old} \| < \epsilon$

Parallelism and program transformation

- Code structure with Gauss-Seidel method

```
do i = 2, n-1
 do j=2, m-1
   a[i,j]=a[i-1,j]+a[i,j-1]+a[i+1,j]+a[i,j+1];
 enddo
enddo
```

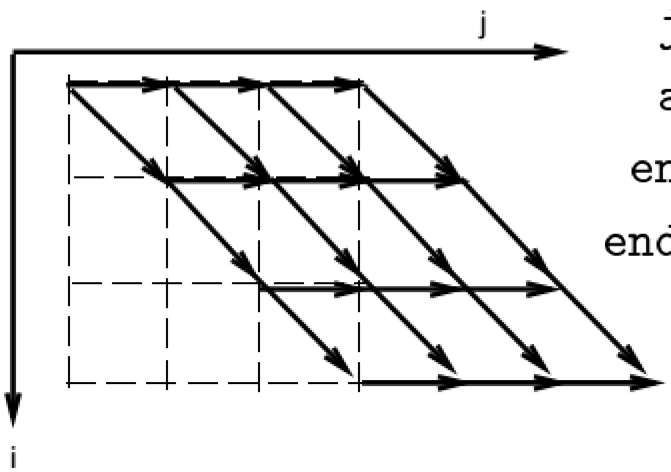


Loop Skewing

```
do i = 2, n-1
 do j=2, m-1
   a[i,j]=a[i-1,j]+a[i,j-1]+a[i+1,j]+a[i,j+1];
 enddo
enddo
```

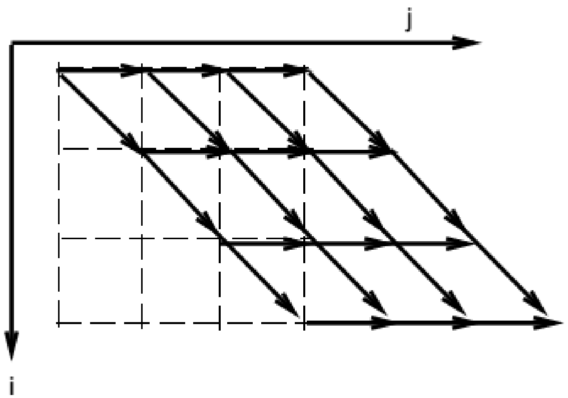
Loop skewing

```
do i = 2, n-1
 do j=i+2, i+m-1
  j'=j-i;
  a[i,j']=a[i-1,j']+a[i,j'-1]+a[i+1,j']+a[i,j'+1]
 enddo
enddo
```



Loop Skewing

```
do i = 2, n-1
 do j=i+2, i+m-1
  j'=j-i;
  a[i,j']=a[i-1,j']+a[i,j'-1]+a[i+1,j']+a[i,j'+1]
 enddo
enddo
```



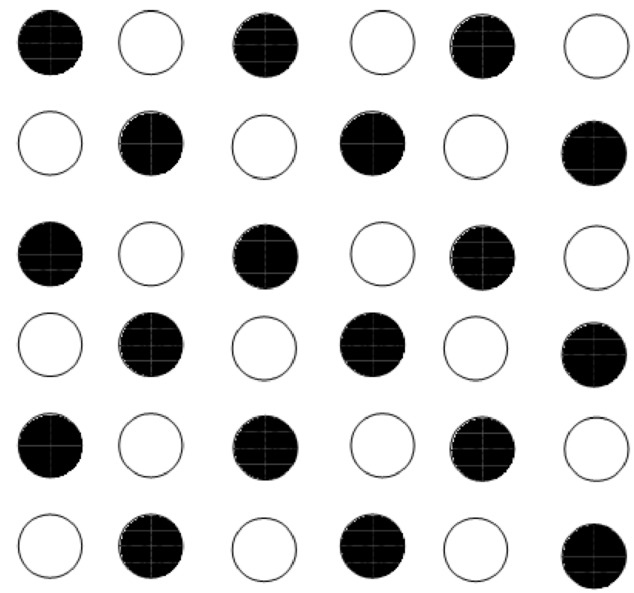
Loop interchange

Loop i can run in parallel

```
do j=4, m+n-2
 do i = max(2,j-m+1), min(n-1,j-2)
  j'=j-i;
  a[i,j']=a[i-1,j']+a[i,j'-1]+a[i+1,j']+a[i,j'+1]
 enddo
enddo
```

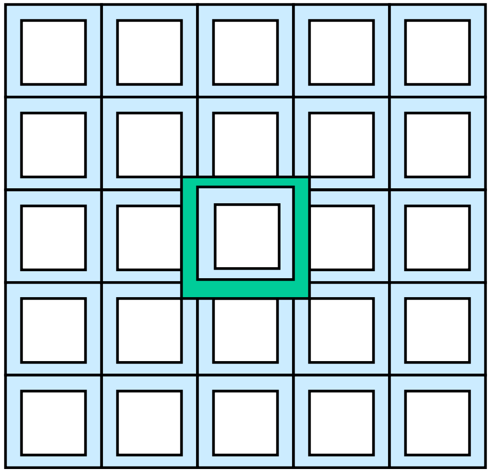
Matrix Reordering for More Parallelism

- Reordering variables to eliminate most of data dependence in the Gauss Seidel algorithm.
- Points are divided into white and black points.
- First, black points are computed using the old red point values.
- Second, while points are computed (using the new black point values).



Parallelism in Regular meshes

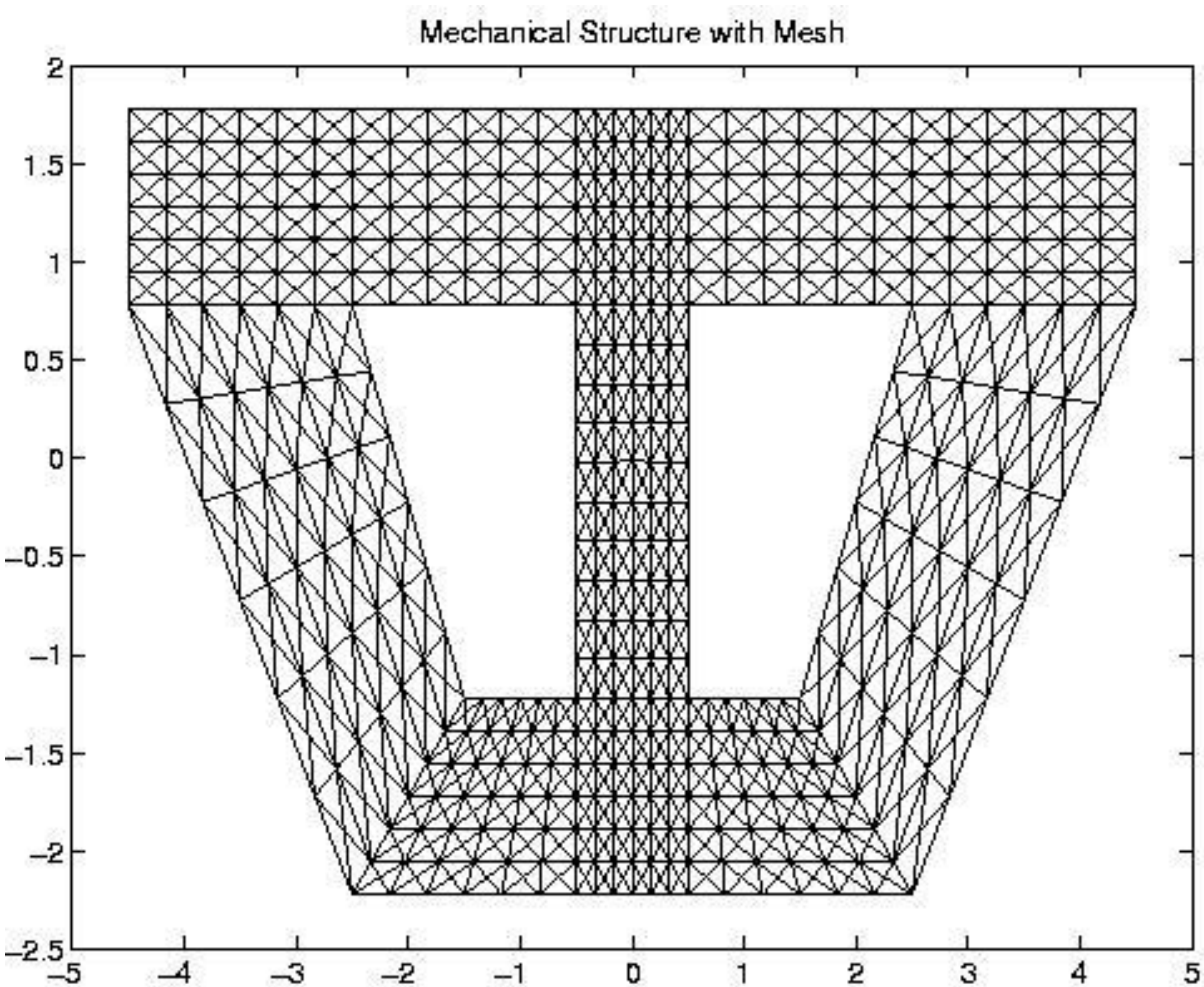
- Computing a Stencil on a regular mesh
  - need to communicate mesh points near boundary to neighboring processors.
    - Often done with ghost regions
  - Surface-to-volume ratio keeps communication down, but
    - Still may be problematic in practice



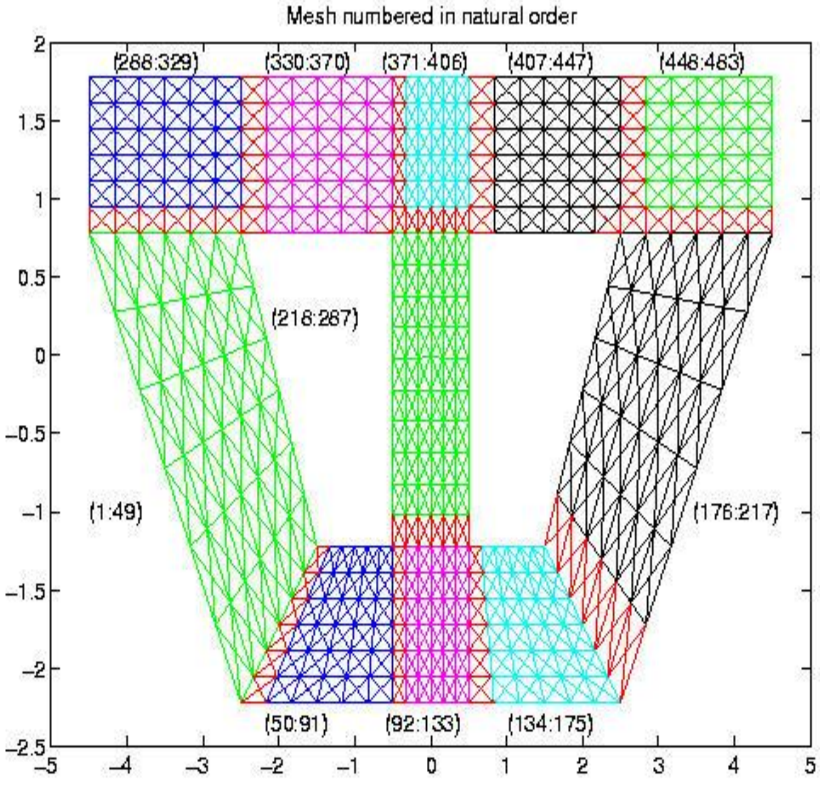
Implemented using "ghost" regions.

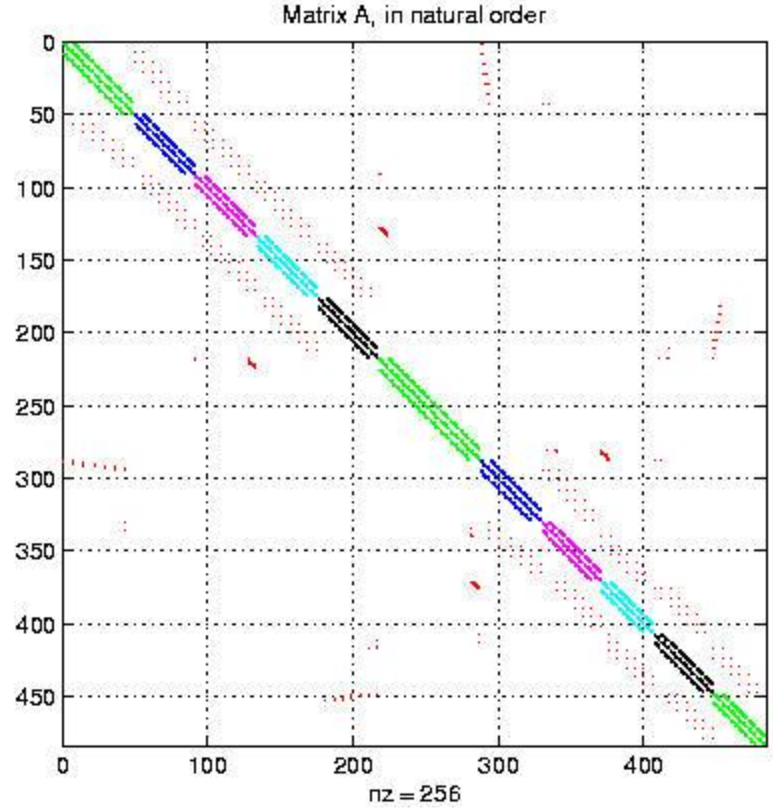
Adds memory overhead

Composite mesh from a mechanical structure



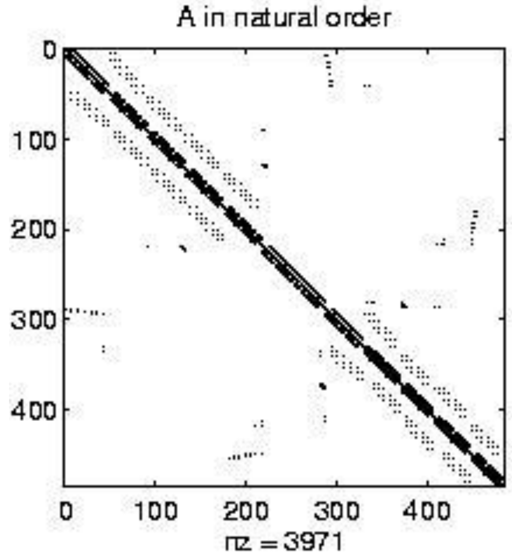
Converting the mesh to a matrix

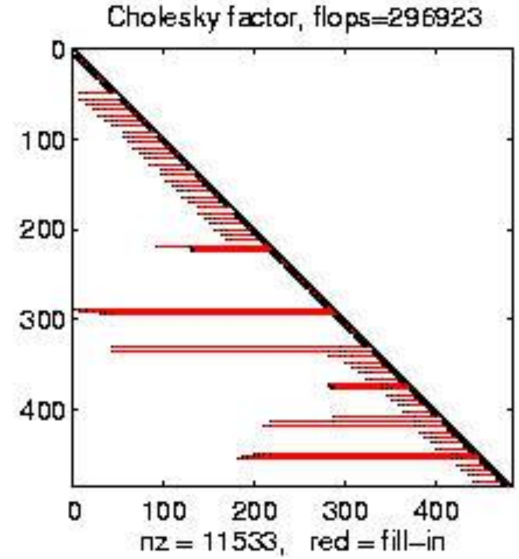




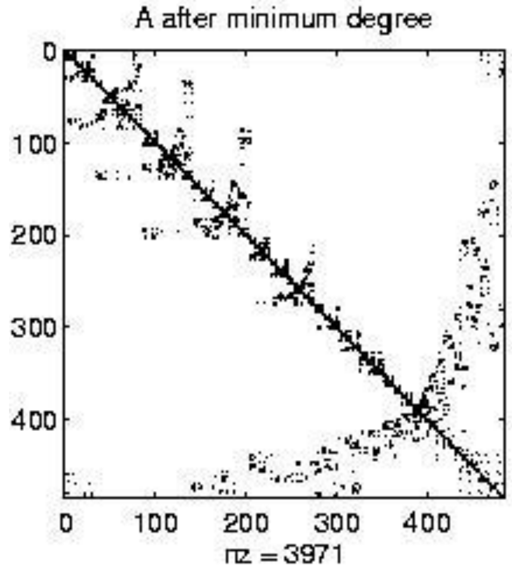
Example of Matrix Reordering Application

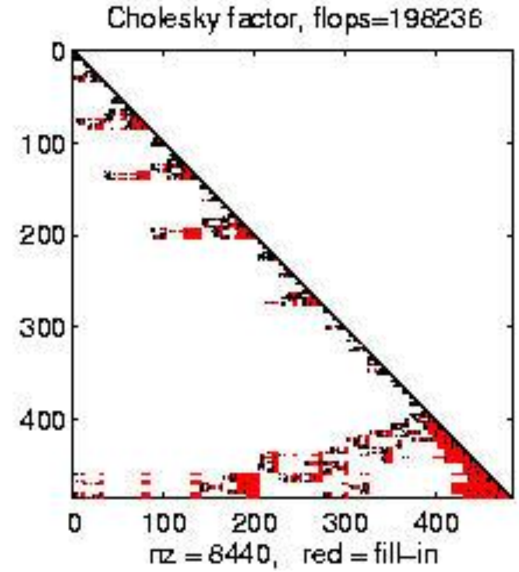
When performing Gaussian Elimination Zeros can be filled ☹



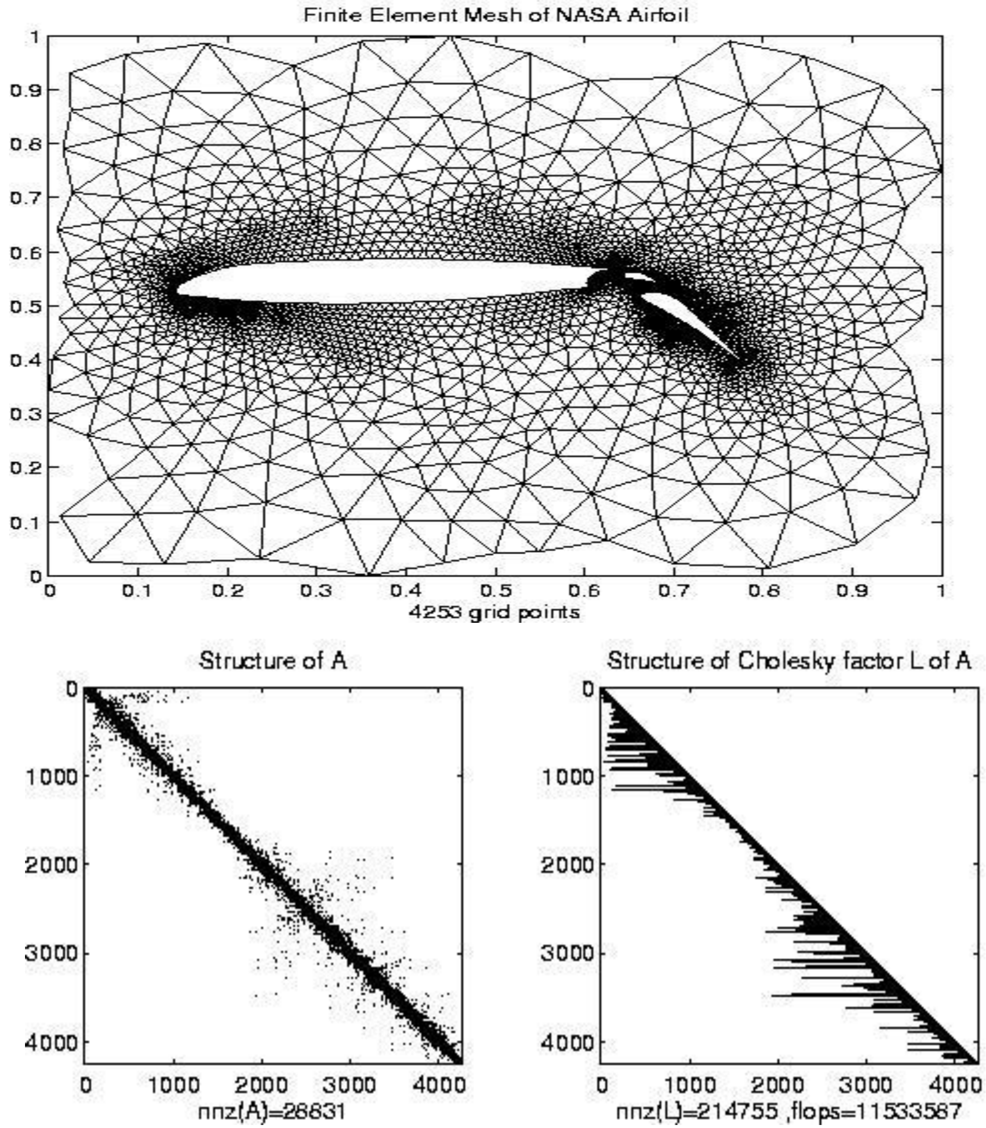


Matrix can be reordered to reduce this fill
But it’s not the same ordering as for parallelism





Irregular mesh: NASA Airfoil in 2D (direct solution)



Irregular mesh: Tapered Tube (multigrid)

Example of Prometheus meshes

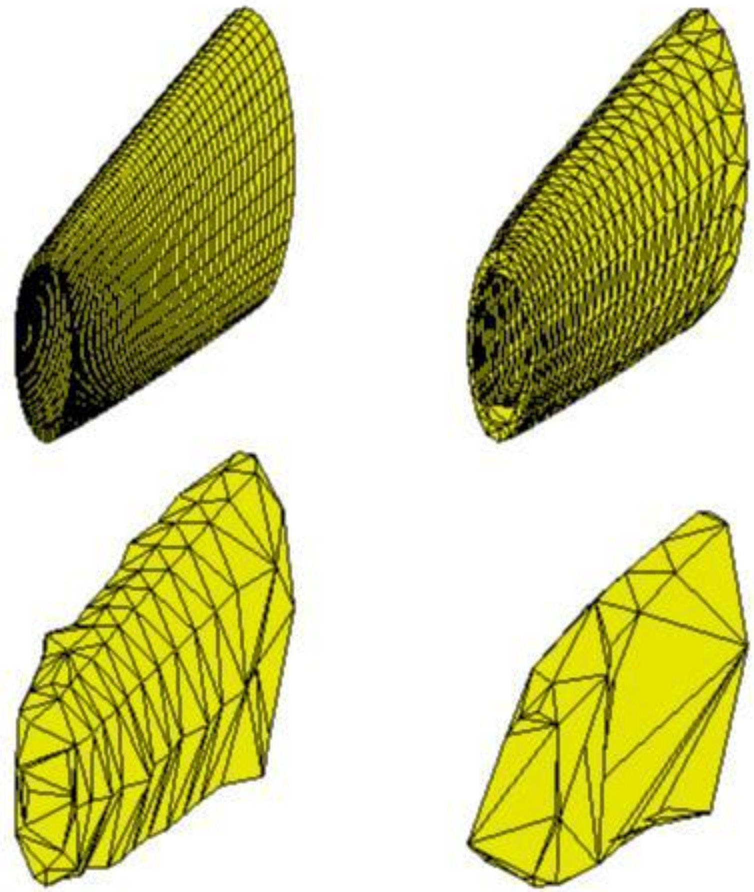
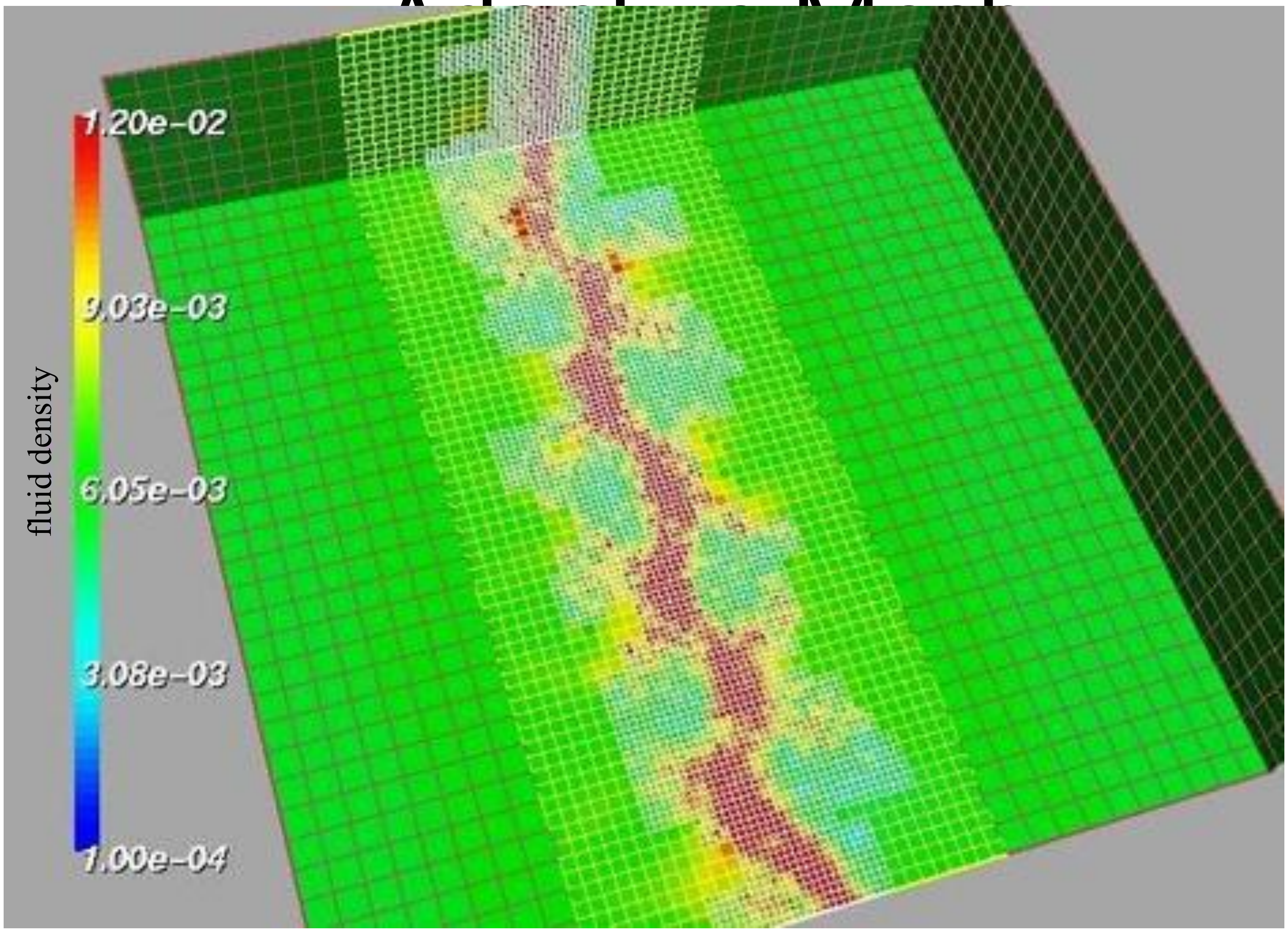


Figure 6: Sample input grid and coarse grids

Adaptive Mesh



Shock waves in gas dynamics using AMR (Adaptive Mesh Refinement) See: http://www.llnl.gov/CASC/SAMRAI/

Challenges of Irregular Meshes

- How to generate them in the first place
  - Start from geometric description of object
  - Triangle, a 2D mesh partitioner by Jonathan Shewchuk
  - 3D harder!
- How to partition them
  - ParMetis, a parallel graph partitioner
- How to design iterative solvers
  - PETSc, a Portable Extensible Toolkit for Scientific Computing
  - Prometheus, a multigrid solver for finite element problems on irregular meshes
- How to design direct solvers
  - SuperLU, parallel sparse Gaussian elimination