Numerical linear algebra in the Canadian hydrological model

Kevin R. Green

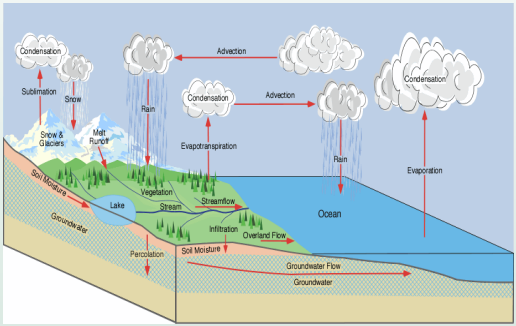
Department of Computer Science
University of Saskatchewan

Math 313
University of Saskatchewan
September 15, 2020

Outline

Hydrology

- The scientific study of the movement, distribution, and quality of water on Earth and other planets.

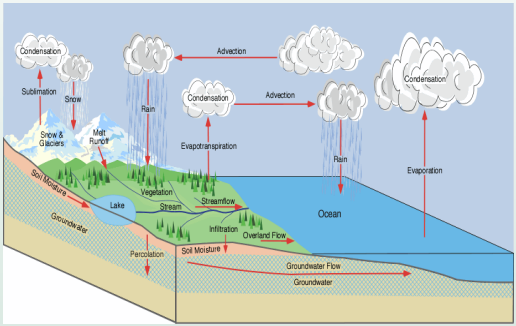


Pidwirny, M. (2006). "The Natural Spheres: The Hydrologic Cycle".

Fundamentals of Physical Geography, 2nd Edition

Hydrology

- The scientific study of the movement, distribution, and quality of water on Earth and other planets.
- Requires complicated physical models that are composed of many smaller components.



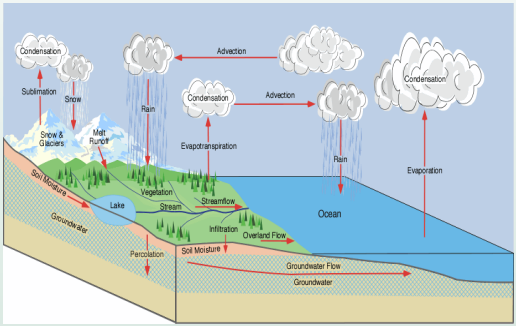
Pidwirny, M. (2006). "The Natural Spheres: The Hydrologic Cycle".

Fundamentals of Physical Geography, 2nd Edition

Hydrology

- The scientific study of the movement, distribution, and quality of water on Earth and other planets.
- Requires complicated physical models that are composed of many smaller components.

The science isn't always settled:



Pidwirny, M. (2006). "The Natural Spheres: The Hydrologic Cycle".

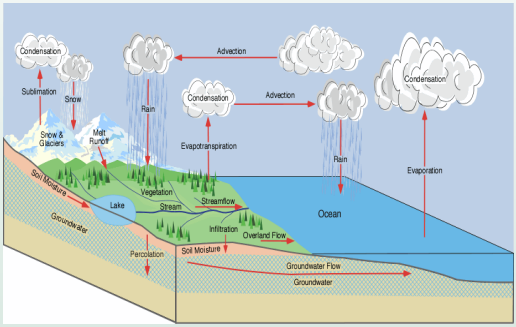
Fundamentals of Physical Geography, 2nd Edition

Hydrology

- The scientific study of the movement, distribution, and quality of water on Earth and other planets.
- Requires complicated physical models that are composed of many smaller components.

The science isn't always settled:

- How to model a given process?



Pidwirny, M. (2006). "The Natural Spheres: The Hydrologic Cycle".

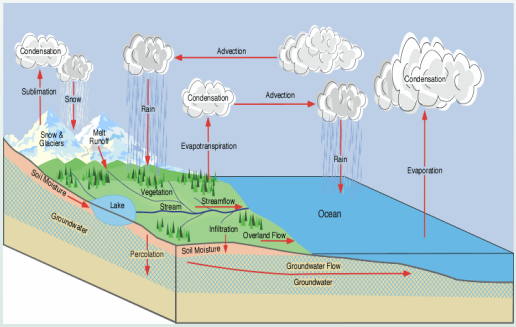
Fundamentals of Physical Geography, 2nd Edition

Hydrology

- The scientific study of the movement, distribution, and quality of water on Earth and other planets.
- Requires complicated physical models that are composed of many smaller components.

The science isn't always settled:

- How to model a given process?
- How do processes interact?



Pidwirny, M. (2006). "The Natural Spheres: The Hydrologic Cycle".

Fundamentals of Physical Geography, 2nd Edition

What is CHM?

The Canadian hydrological model (CHM) is a novel unstructured mesh-based approach to hydrological modelling. It is designed with a plug-in architecture of modular process representations to assist with the development and testing of hydrological models. The combination of unstructured meshes with plug-in processes allows CHM to bridge spatial scales, temporal scales, and spatial extents.

`https://github.com/Chrismarsh/CHM`

CHM

CHM

- is concerned mostly with modelling surface and ground water hydrology.

CHM

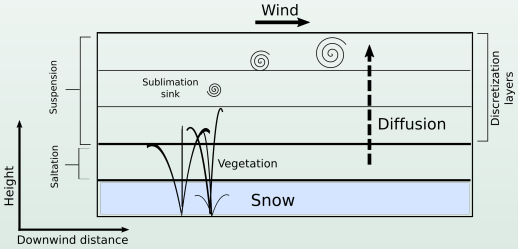
- is concerned mostly with modelling surface and ground water hydrology.
- is intended to handle spatially heterogeneous models required for snow redistribution phenomena (for example).

CHM

- is concerned mostly with modelling surface and ground water hydrology.
- is intended to handle spatially heterogeneous models required for snow redistribution phenomena (for example).
- is designed with the ability to plug in various *types* of other models as well.

Example: Prairie blowing snow model (PBSM3D)

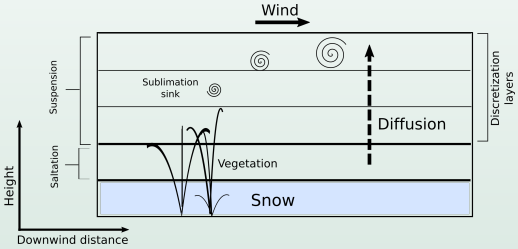
- *Partial differential equation* model for blowing snow.



Marsh et al., 2019.

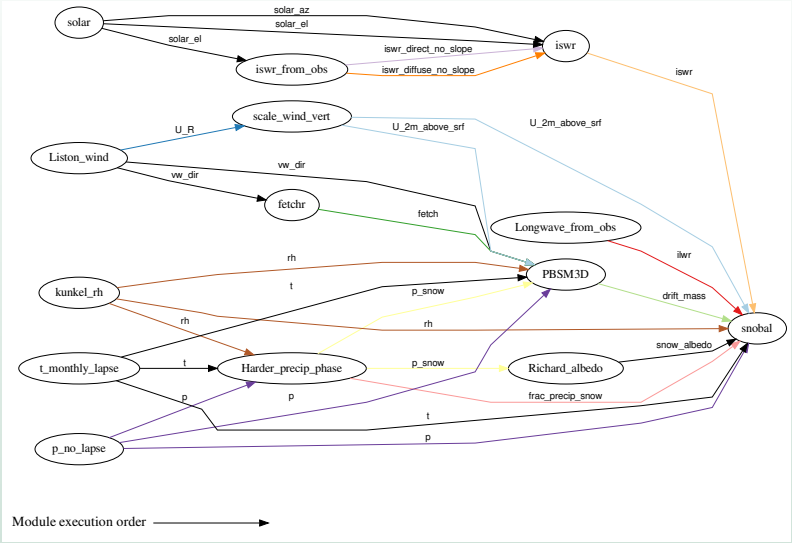
Example: Prairie blowing snow model (PBSM3D)

- *Partial differential equation* model for blowing snow.
- Saltation and suspension processes.
  - Multi-layer suspension.



Marsh et al., 2019.

A *basic* simulation with PBSM3D blowing snow



Space-dependence

- Some processes are influenced by (and influence) model behaviour in nearby regions of a domain.

Space-dependence

- Some processes are influenced by (and influence) model behaviour in nearby regions of a domain.
- Partial differential equations provide a means for describing such interaction with calculus. i.e.,

Space-dependence

- Some processes are influenced by (and influence) model behaviour in nearby regions of a domain.
- Partial differential equations provide a means for describing such interaction with calculus. i.e.,

$$-\nabla^2 u = f(x, y)$$
$$-u_{xx}(x, y) - u_{yy}(x, y) = f(x, y)$$

Space-dependence

- Some processes are influenced by (and influence) model behaviour in nearby regions of a domain.
- Partial differential equations provide a means for describing such interaction with calculus. i.e.,

$$-\nabla^2 u = f(x, y)$$
$$-u_{xx}(x, y) - u_{yy}(x, y) = f(x, y)$$

"Second derivative of u in the x direction plus the second derivative of u in the y direction equals a known function f"

Space-dependence

- Some processes are influenced by (and influence) model behaviour in nearby regions of a domain.
- Partial differential equations provide a means for describing such interaction with calculus. i.e.,

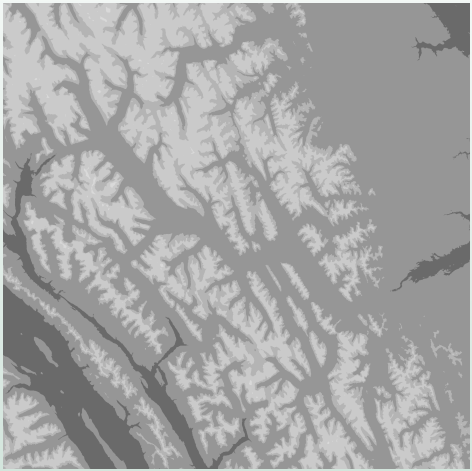
$$-\nabla^2 u = f(x, y)$$
$$-u_{xx}(x, y) - u_{yy}(x, y) = f(x, y)$$

"Second derivative of u in the x direction plus the second derivative of u in the y direction equals a known function f"

$\rightarrow$ What is $u(x, y)$ for all x and y?

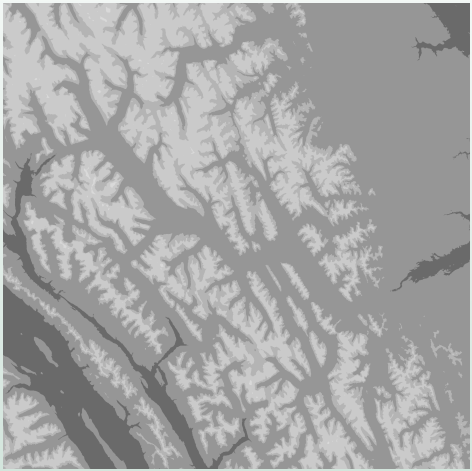
Spatial domains

- Geospatial data: latitude, longitude, elevation *(+ other data)*



Spatial domains

- Geospatial data: latitude, longitude, elevation *(+ other data)*
- Raster format *(data at each "pixel")*



Mesh generation

Goal:

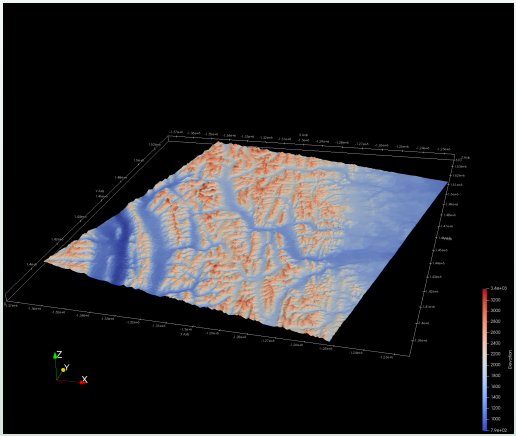
Produce a representation of the spatial domain

Mesh generation

Goal:

Produce a representation of the spatial domain

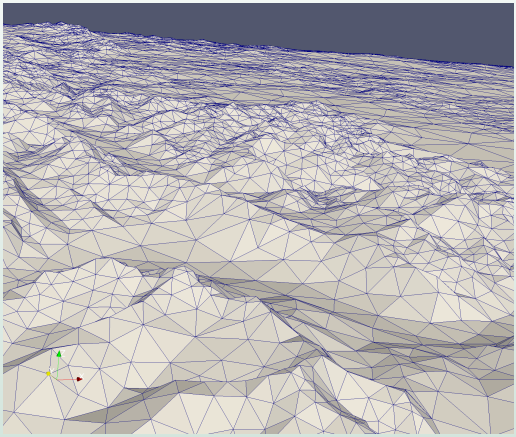
(This domain is $\sim$ 150 km $\times$ 150 km)



Bow Valley, Alberta.

Triangular mesh

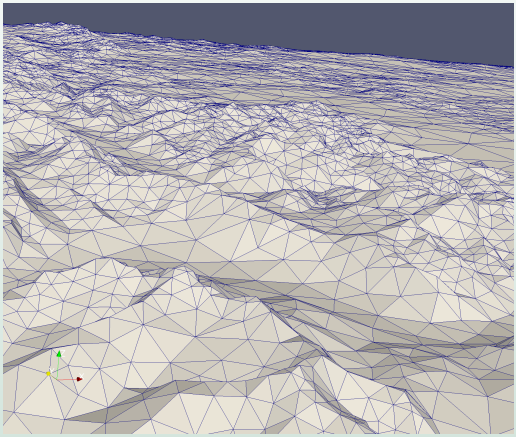
Spatial representation is made up of a bunch of planar triangles



Triangular mesh

Spatial representation is made up of a bunch of planar triangles

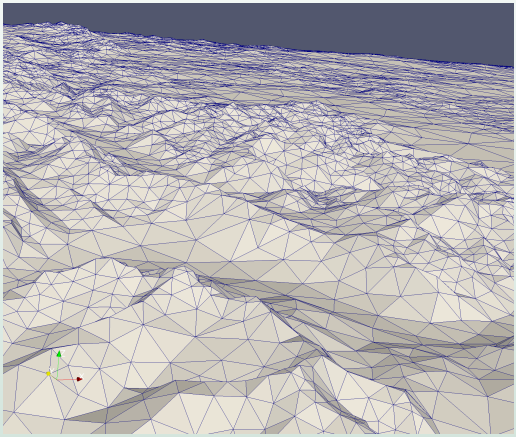
- variable size



Triangular mesh

Spatial representation is made up of a bunch of planar triangles

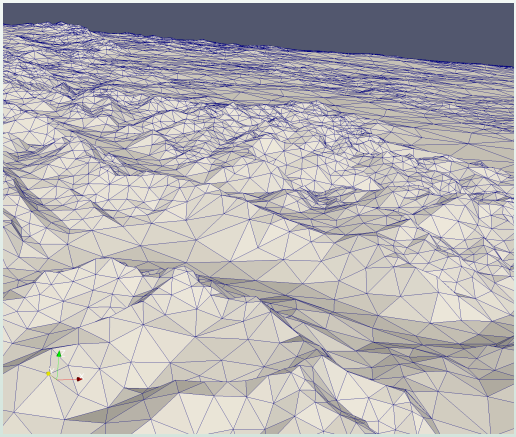
- variable size
- variable orientation



Triangular mesh

Spatial representation is made up of a bunch of planar triangles

- variable size
- variable orientation
- interpolating the input data

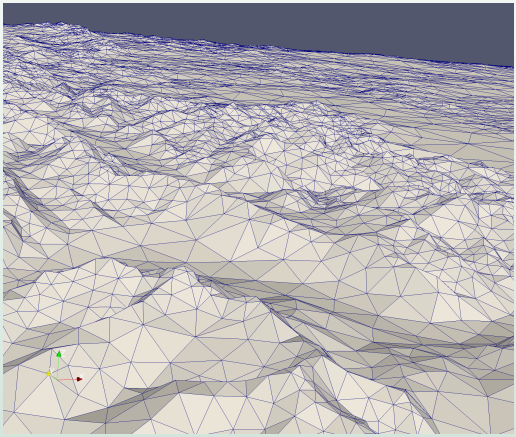


Triangular mesh

Spatial representation is made up of a bunch of planar triangles

- variable size
- variable orientation
- interpolating the input data

How do we get here?



Mesher

https://github.com/Chrismarsh/mesher

Mesher

Iteratively generate a triangular mesh according to various criteria, like elevation accuracy.

https://github.com/Chrismarsh/mesher

Mesher

Iteratively generate a triangular mesh according to various criteria, like elevation accuracy.

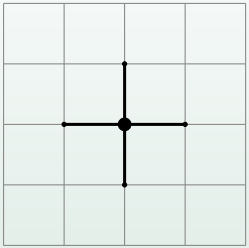
(Computational geometry. . . if you're interested)

`https://github.com/Chrismarsh/mesher`

Discretizing a PDE

Finite difference method

- approximate spatial derivatives with finite differences (only really useful for highly structured meshes)



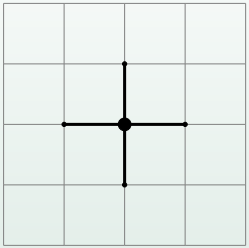
Discretizing a PDE

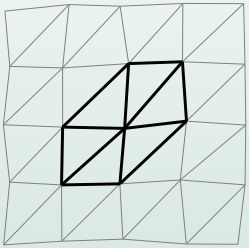
Finite difference method

- approximate spatial derivatives with finite differences (only really useful for highly structured meshes)

Finite element method

- using "nodal" points (vertices of mesh) to approximate function values over the "elements" (triangles)





Discretizing a PDE

Finite difference method

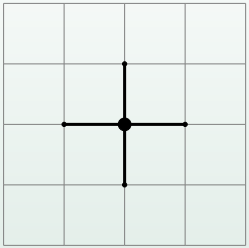
- approximate spatial derivatives with finite differences (only really useful for highly structured meshes)

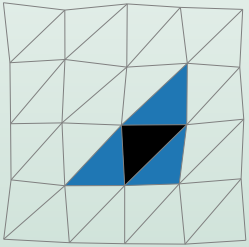
Finite element method

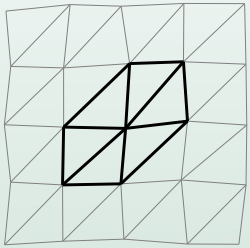
- using "nodal" points (vertices of mesh) to approximate function values over the "elements" (triangles)

Finite volume method

- what flows out of one element (triangle), flows into its neighbour







Calculus to linear algebra

Calculus to linear algebra

Discretizing a linear PDE

Calculus to linear algebra

Discretizing a linear PDE

results in linear algebraic equations.

Calculus to linear algebra

Discretizing a linear PDE

results in linear algebraic equations.

There are many ways to do this.

Calculus to linear algebra

Discretizing a linear PDE

results in linear algebraic equations.

There are many ways to do this.

The mesh-based approaches listed above result in sparsely coupled equations.

Calculus to linear algebra

Discretizing a linear PDE

results in linear algebraic equations.

There are many ways to do this.

The mesh-based approaches listed above result in sparsely coupled equations.

(Numerical PDEs. . . if you're interested)

Sparse linear algebra

Solving

$$Au = b$$

Sparse linear algebra

Solving

$$Au = b$$

- A - sparse matrix, discrete version of spatial influence

Sparse linear algebra

Solving

$$Au = b$$

- A - sparse matrix, discrete version of spatial influence
- b - known vector, discrete version of $f(x, y)$

Sparse linear algebra

Solving

$$Au = b$$

- A - sparse matrix, discrete version of spatial influence
- b - known vector, discrete version of $f(x, y)$
- u - unknown vector, discrete version of $u(x, y)$

Sparse linear algebra

Solving

$$Au = b$$

There are many ways to do this too.

- A - sparse matrix, discrete version of spatial influence
- b - known vector, discrete version of $f(x, y)$
- u - unknown vector, discrete version of $u(x, y)$

Direct methods

Direct "inversion" of linear systems.

- Obtain numerically precise solution
  $|Au - b| \sim \mathcal{O}(\epsilon_{\text{mach}})$

ϵ_{mach} – *machine epsilon*

Direct methods

Direct "inversion" of linear systems.

- Obtain numerically precise solution $|Au - b| \sim \mathcal{O}(\epsilon_{\text{mach}})$

ϵ_{mach} – *machine epsilon*

Methods:

- Cholesky [Benoit1924]

$$PAP^T = LL^T$$

Direct methods

Direct "inversion" of linear systems.

- Obtain numerically precise solution $|Au - b| \sim \mathcal{O}(\epsilon_{\text{mach}})$

ϵ_{mach} – *machine epsilon*

Methods:

- Cholesky [Benoit1924]

$$PAP^T = LL^T$$

- LU [Banachiewicz1938]

$$PAQ^T = LU$$

Direct methods

Direct "inversion" of linear systems.

- Obtain numerically precise solution $|Au - b| \sim \mathcal{O}(\epsilon_{\text{mach}})$

ϵ_{mach} – *machine epsilon*
P, Q – *permutation matrices*

Methods:

- Cholesky [Benoit1924]

$$PAP^T = LL^T$$

- LU [Banachiewicz1938]

$$PAQ^T = LU$$

Pros and cons of direct methods

Pros:

Cons:

Pros and cons of direct methods

Pros:

- As accurate as you can get numerically

Cons:

Pros and cons of direct methods

Pros:

- As accurate as you can get numerically

Cons:

- Algorithms scale poorly ($\mathcal{O}(m^3)$ in m unknowns, for a general sparse matrix)

Pros and cons of direct methods

Pros:

- As accurate as you can get numerically
- Re-use factorization to solve with different b

Cons:

- Algorithms scale poorly ($\mathcal{O}(m^3)$ in m unknowns, for a general sparse matrix)

Pros and cons of direct methods

Pros:

- As accurate as you can get numerically
- Re-use factorization to solve with different b

Cons:

- Algorithms scale poorly ($\mathcal{O}(m^3)$ in m unknowns, for a general sparse matrix)

Pro/Con:

- Time to solution dependent *entirely* on matrix structure
  - *might* be substantially better than worst-case

Permutations

Free to rearrange our equations and unknowns.

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

Permutations

Free to rearrange our equations and unknowns.

$$Au = b$$

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

Permutations

Free to rearrange our equations and unknowns.

$$Au = b$$
$$PAu = Pb$$

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$Au = b$$
$$PAu = Pb$$
$$PA\left(Q^T Q\right) u = Pb$$

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$Au = b$$
$$PAu = Pb$$
$$PA\left(Q^T Q\right) u = Pb$$
$$\left(PAQ^T\right)(Qu) = Pb$$

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$
\begin{aligned}
Au &= b \\
PAu &= Pb \\
PA\left(Q^T Q\right) u &= Pb \\
\left(PAQ^T\right)(Qu) &= Pb \\
\tilde{A}\tilde{u} &= \tilde{b}
\end{aligned}
$$

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$Au = b$$
$$PAu = Pb$$
$$PA\left(Q^T Q\right) u = Pb$$
$$\left(PAQ^T\right)(Qu) = Pb$$
$$\tilde{A}\tilde{u} = \tilde{b}$$

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$Au = b$$
$$PAu = Pb$$
$$PA\left(Q^T Q\right) u = Pb$$
$$\left(PAQ^T\right)(Qu) = Pb$$
$$\tilde{A}\tilde{u} = \tilde{b}$$

Permutations

Free to rearrange our equations and unknowns.

Within reason:

- Numerical stability considerations

Can be used to alter matrix structure...

view as matrix multiplication:

$$
\begin{aligned}
Au &= b \\
PAu &= Pb \\
PA\left(Q^T Q\right) u &= Pb \\
\left(PAQ^T\right)(Qu) &= Pb \\
\tilde{A}\tilde{u} &= \tilde{b}
\end{aligned}
$$

Want $\tilde{A}$ to have a *better* structure.

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P
- $A \rightarrow PAP^T$

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P
- $A \rightarrow PAP^T$

Nice properties:

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P
- $A \rightarrow PAP^T$

Nice properties:

- If A is symmetric positive definite (SPD), so is PAP^T.

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P
- $A \rightarrow PAP^T$

Nice properties:

- If A is symmetric positive definite (SPD), so is PAP^T.
- If A is diagonally dominant, so is PAP^T.

Symmetric permutations

Changing equation *and* unknown ordering simultaneously.

- Permutation matrix P
- $A \rightarrow PAP^T$

Nice properties:

- If A is symmetric positive definite (SPD), so is PAP^T.
- If A is diagonally dominant, so is PAP^T.

$\rightarrow$ No loss of stability for certain PDE operators.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}`

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

How? *(NP-Hard) problem*

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$

How? *(NP-Hard) problem*

Heuristic algorithms

- Cuthill–McKee (CM), [Cuthill1969]

Double precision, general banded, triangular, factor/solve.

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

How? *(NP-Hard) problem*

Heuristic algorithms

- Cuthill–McKee (CM), [Cuthill1969]
- Reverse Cuthill–McKee (RCM), [George1971]

Bandwidth-reducing order

Goal: Minimize bandwidth of A.

Why?

- Bandwidth of factors is less than or equal to bandwidth of original matrix
- Less chance for "fill-in" to occur in factors.
- "Banded" solvers can exploit this structure for efficiency.
  - LAPACK routines like `dgbtr{fs}` $\rightarrow$ Double precision, general banded, triangular, factor/solve.

How? *(NP-Hard) problem*

Heuristic algorithms

- Cuthill–McKee (CM), [Cuthill1969]
- Reverse Cuthill–McKee (RCM), [George1971]

We use RCM for reduced bandwidth order.

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Why?

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Why?

- Sparse forward/backward substitution only operates on entries that are there.
  - $\mathcal{O}(nnz)$, nnz - number of non-zeros

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Why?

- Sparse forward/backward substitution only operates on entries that are there.
  - $\mathcal{O}(nnz)$, nnz - number of non-zeros
- Fewer operations to perform in forward/backward substitution.

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Why?

- Sparse forward/backward substitution only operates on entries that are there.
  - $\mathcal{O}(nnz)$, nnz - number of non-zeros
- Fewer operations to perform in forward/backward substitution.

How? *(Heuristics, again)*

Fill-reducing ordering

Goal: Minimize amount of fill-in in factors. *(Also NP-Hard)*

Why?

- Sparse forward/backward substitution only operates on entries that are there.
  - $\mathcal{O}(nnz)$, nnz - number of non-zeros
- Fewer operations to perform in forward/backward substitution.

How? *(Heuristics, again)*

- Algorithms based on Multilevel Nested Dissection (ND) [George1973]

Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

- D – smoothing coefficient
- $u(x, y)$ – rate of snow mass transport at x, y coordinate
- $f(x, y)$ – known quantity dependent on other physical parameters

Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

(Requires much more, but we're just going to stick to this)

- D – smoothing coefficient
- $u(x, y)$ – rate of snow mass transport at x, y coordinate
- $f(x, y)$ – known quantity dependent on other physical parameters

Discretization of specific problem

$$Au = b$$

Discretization of specific problem

$$Au = b$$

A represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$.
(A is symmetric positive definite.)

Discretization of specific problem

$$Au = b$$

A represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$.
(A is symmetric positive definite.)

Components of u represent the rate of snow mass transport, constant on each element.

Discretization of specific problem

$$Au = b$$

A represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$. (A is symmetric positive definite.)

Components of u represent the rate of snow mass transport, constant on each element.

Components of b represent known quantities dependent on current snow mass (among other things), constant on each element.

Discretization of specific problem

$$Au = b$$

A represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$. (A is symmetric positive definite.)

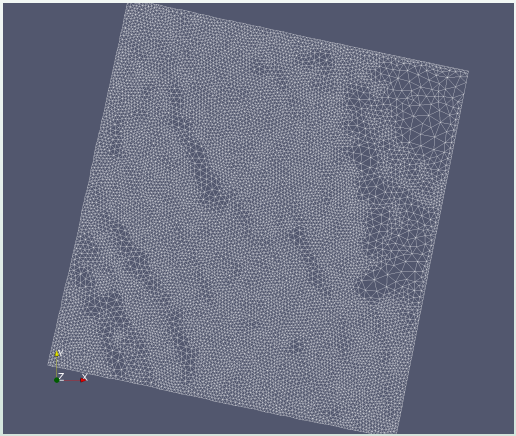
Components of u represent the rate of snow mass transport, constant on each element.

Components of b represent known quantities dependent on current snow mass (among other things), constant on each element.

Use known solution $u^* = 1^{N\times 1}$, $b^* = Au^*$ for testing, i.e., solve $Au = b^*$.

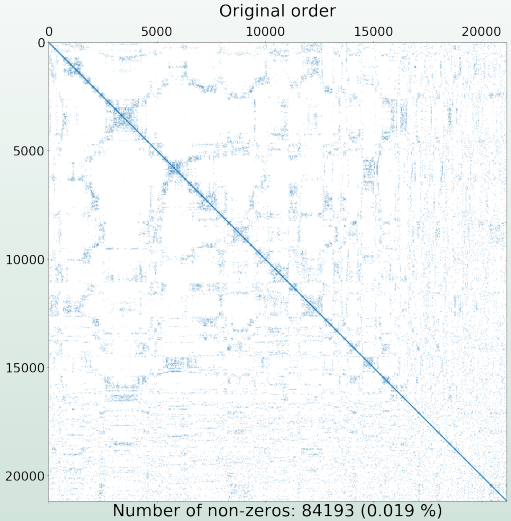
Mesh: coarse resolution

Low resolution (21168 elements)



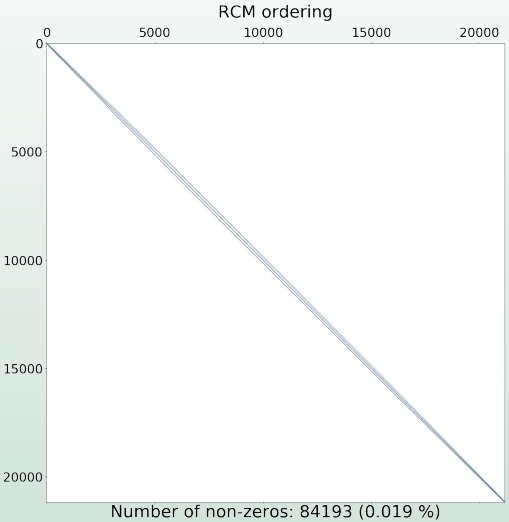
Matrix structure: original order

Bandwidth: 21062



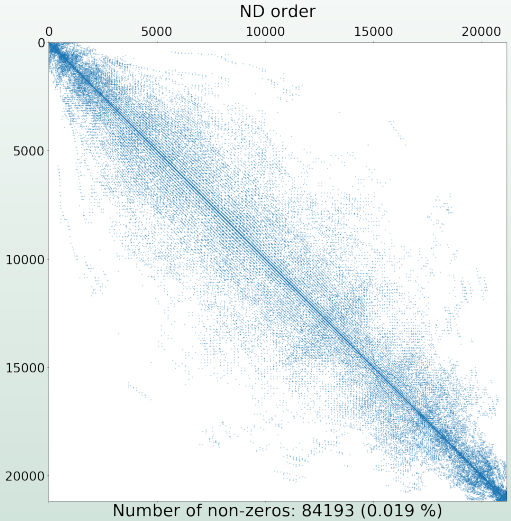
Matrix structure: RCM order

Bandwidth: 168



Matrix structure: ND order

Bandwidth: 12269



Performance: coarse mesh

Not shown here.

Performance: coarse mesh

Not shown here.

Jupyter notebook

- `https:github.com/uofs-simlab/Math313`
- Can run easily from `colab.research.google.com` if you have a Google account.

Performance: coarse mesh

Not shown here.

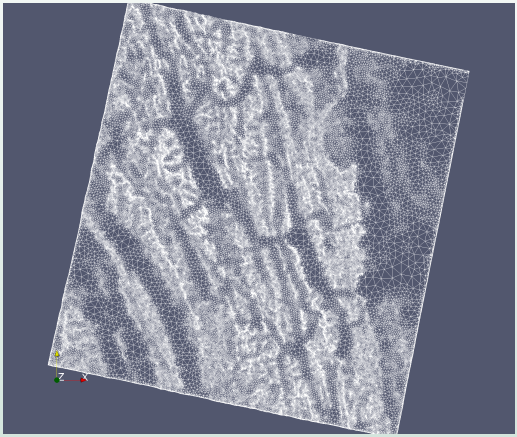
Jupyter notebook

- `https:github.com/uofs-simlab/Math313`
- Can run easily from `colab.research.google.com` if you have a Google account.

I went bigger...

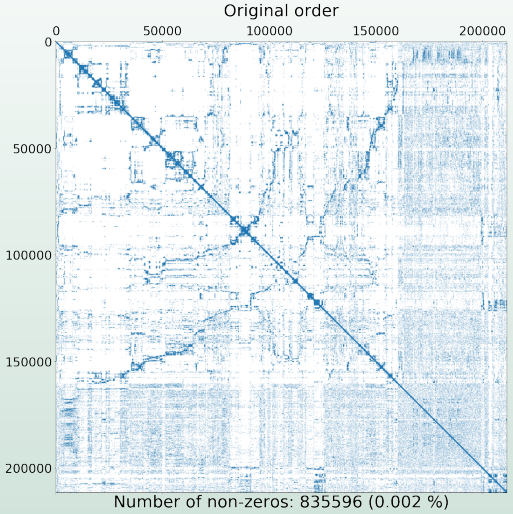
Mesh: fine resolution

High resolution (211415 elements)



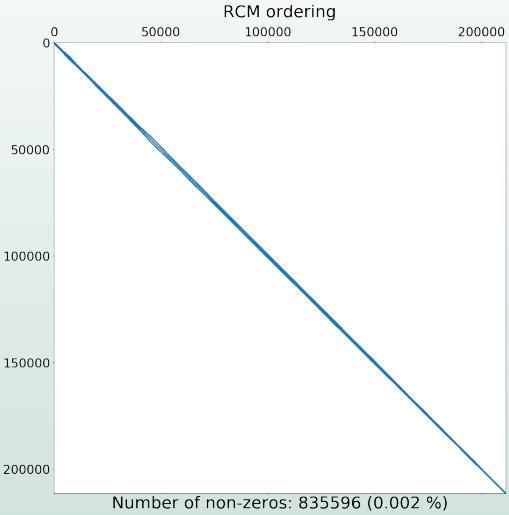
Matrix structure: original order

Bandwidth: 210855



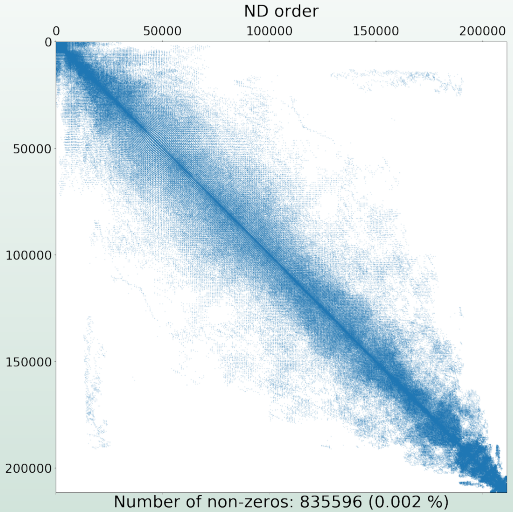
Matrix structure: RCM order

Bandwidth: 1320



Matrix: ND order

Bandwidth: 173136

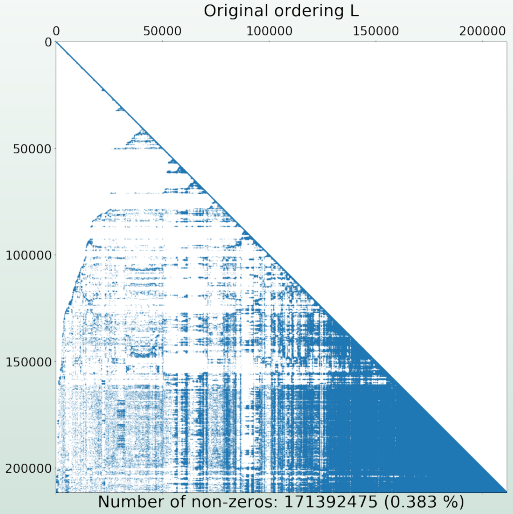


Performance: LU – Original order

LU factorization

```
CPU times:
  user:  8d 12h 41min 15s,
  sys:       18h  1min 11s,
  total: 9d  6h 42min 26s

Wall time:  6h 58min 57s
```

Solve

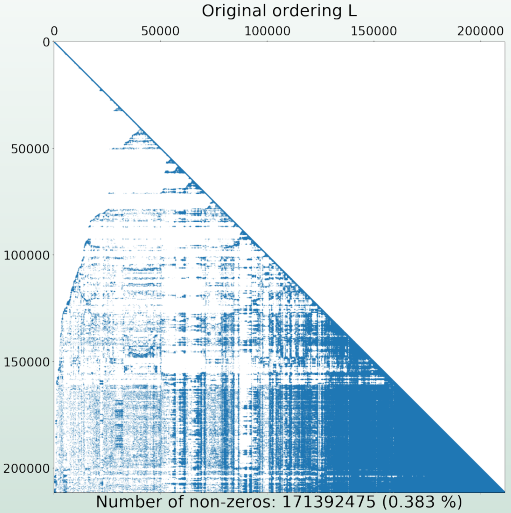


Performance: LU – Original order

LU factorization

```
CPU times:
  user:  8d 12h 41min 15s,
  sys:       18h  1min 11s,
  total: 9d  6h 42min 26s

Wall time:  6h 58min 57s
```

Solve



Performance: LU – Original order

LU factorization

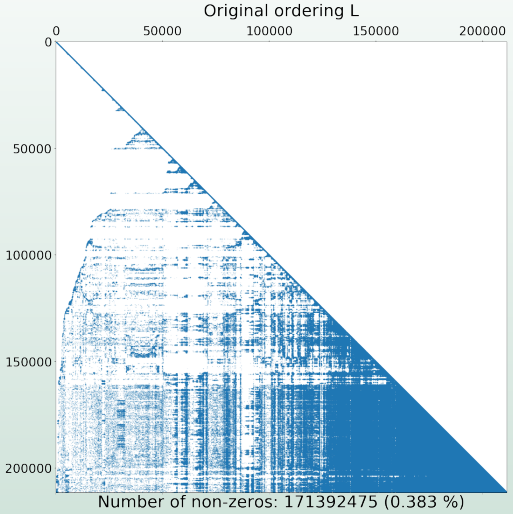
```
CPU times:
  user:  8d 12h 41min 15s,
  sys:       18h  1min 11s,
  total: 9d  6h 42min 26s

Wall time:  6h 58min 57s
```

Solve

```
Wall time:  8.74s
```

Total

```
Wall time:  6h 59min 5s
```



Performance: LU – Original order

LU factorization

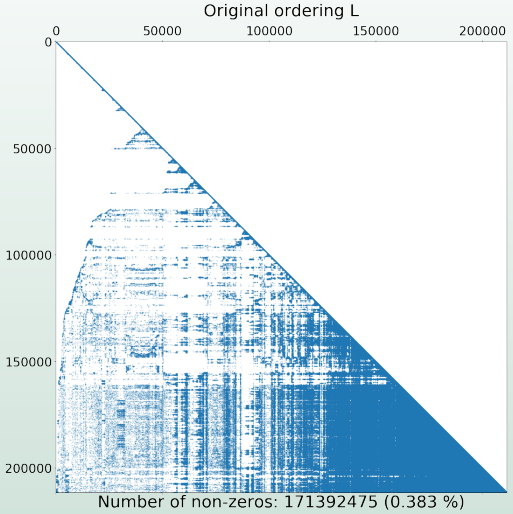
```
CPU times:
  user:  8d 12h 41min 15s,
  sys:       18h  1min 11s,
  total: 9d  6h 42min 26s

Wall time:  6h 58min 57s
```

Solve

```
Wall time:  8.74s
```

Total

```
Wall time:  6h 59min 5s
```

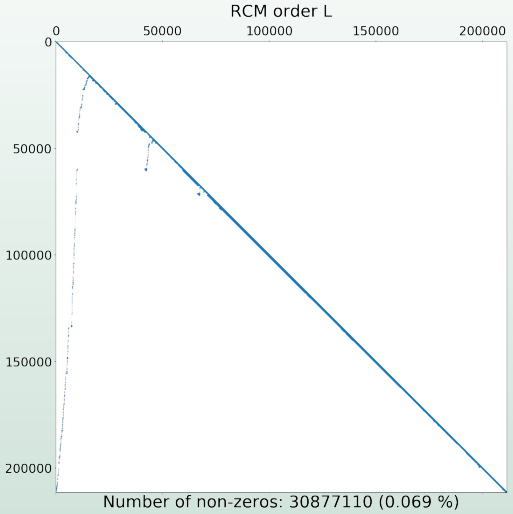


Performance: LU – RCM order

LU factorization

```
CPU times:
  user:  56.2  s,
  sys:    2.71 s,
  total:  58.9 s

Wall time:  52.6 s
```

Solve

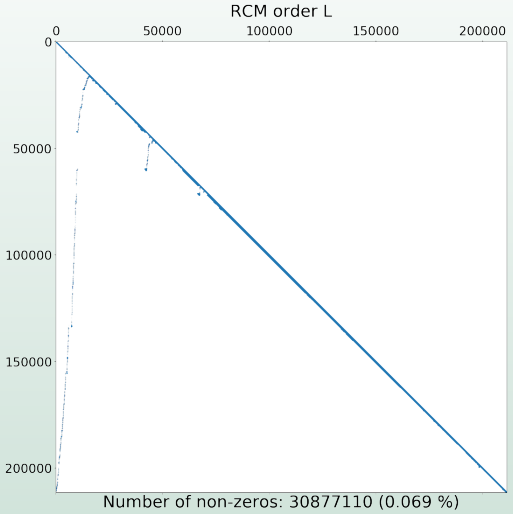


Performance: LU – RCM order

LU factorization

```
CPU times:
  user:  56.2  s,
  sys:    2.71 s,
  total:  58.9 s

Wall time:  52.6 s
```

Solve



Performance: LU – RCM order

LU factorization

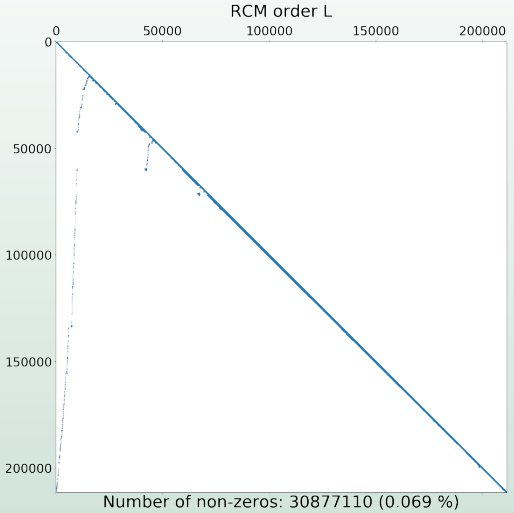
```
CPU times:
  user:  56.2  s,
  sys:    2.71 s,
  total:  58.9 s

Wall time:  52.6 s
```

Solve

```
Wall time:  0.741 s
```

Total

```
Wall time:  53.3s
```



Performance: LU – RCM order

LU factorization

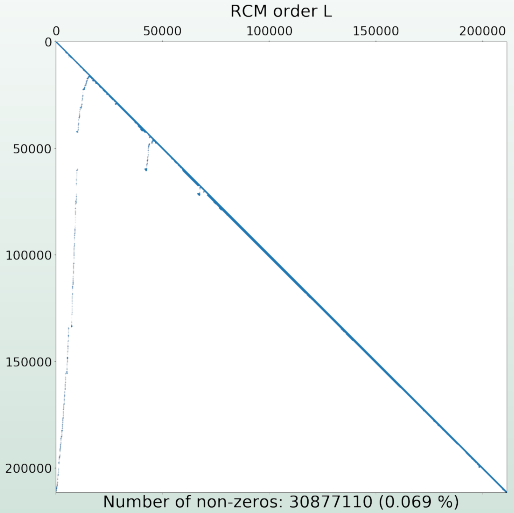
```
CPU times:
  user:  56.2  s,
  sys:    2.71 s,
  total:  58.9 s

Wall time:  52.6 s
```

Solve

```
Wall time:  0.741 s
```

Total

```
Wall time:  53.3s
```

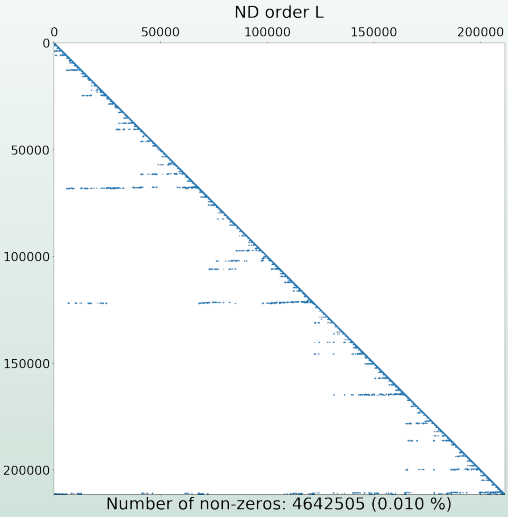


Performance: LU – ND order

LU factorization

```
CPU times:
  user:  6.67  s,
  sys:   0.021 s,
  total: 6.67  s

 Wall time:  1.25 s
```

Solve

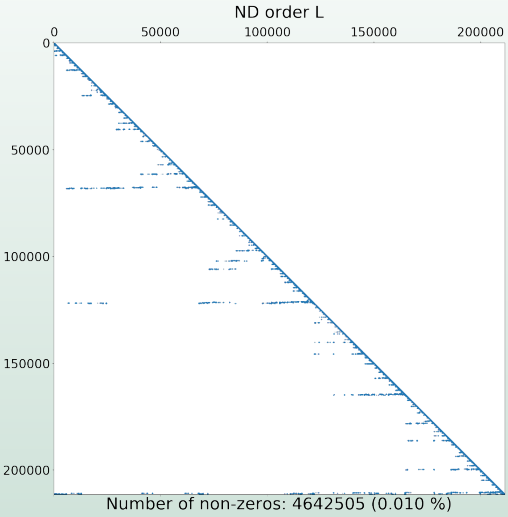


Performance: LU – ND order

LU factorization

```
CPU times:
  user:  6.67  s,
  sys:   0.021 s,
  total: 6.67  s

 Wall time:  1.25 s
```

Solve



Performance: LU – ND order

LU factorization

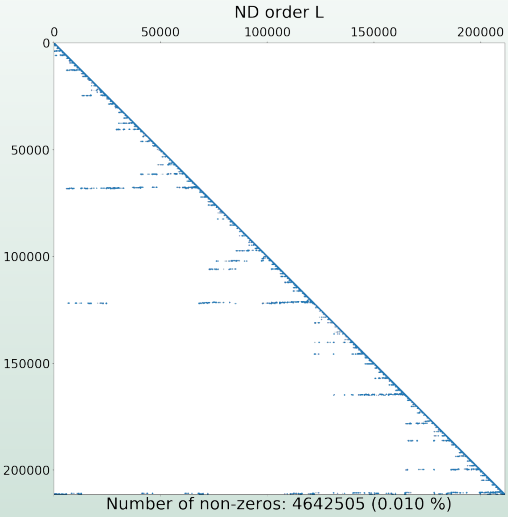
```
CPU times:
  user:  6.67  s,
  sys:   0.021 s,
  total: 6.67  s

 Wall time:  1.25 s
```

Solve

```
 Wall time:  0.049 s
```

Total

```
 Wall time:  1.30s
```



Performance: LU – ND order

LU factorization

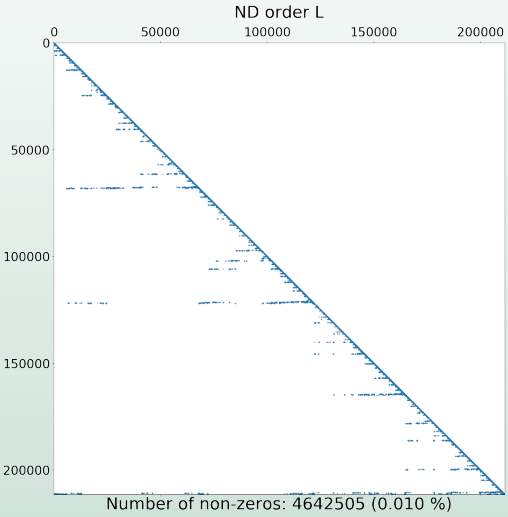
```
CPU times:
  user:  6.67  s,
  sys:   0.021 s,
  total: 6.67  s

 Wall time:  1.25 s
```

Solve

```
 Wall time:  0.049 s
```

Total

```
 Wall time:  1.30s
```



Fine mesh summary

211415 elements $\rightarrow$ 835596 non-zero matrix entries

LU		
Order	Time (s)	Accuracy
original	25145	1.2e-13
RCM	53.3	1.6e-13
ND	1.3	1.5e-13

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

- Substantial performance improvement observed in direct solve simply by reordering.

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

- Substantial performance improvement observed in direct solve simply by reordering.

- We haven't even talked about:

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

- Substantial performance improvement observed in direct solve simply by reordering.

- We haven't even talked about:
  - iterative algorithms *(next guest lecture!)*

Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

- Substantial performance improvement observed in direct solve simply by reordering.

- We haven't even talked about:
  - iterative algorithms *(next guest lecture!)*
  - machine-dependent things *(High-performance computing... if you're interested)*