Numerical Simulation and Scientific Computing I

Lecture 3:
C++ Optimization, Intro Finite Difference Method

Xaver Klemenschits, Paul Manstetten, and
Josef Weinbub

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

WS 2022

Quiz

- Q1: How to deal with “turbo frequencies” when estimating the peak performance for a CPU?
- Q2: Why did Intel drop AVX-512 from the P-Cores (e.g. i9-13900k)?
- Q3: For very large N, is the performance of a dense matrix-matrix multiplication memory bound or computationally bound?

Quiz

- Q4: What are options to discretize the first and second derivatives of a one-dimensional function using discretization points of distance h?

- Q5: Using the trapezoidal rule to integrate

$$A = \int_{-\pi/2}^{+\pi/2} \cos(x)\,dx$$

does the approximation overestimate or underestimate the integral value A?

Outline

- Manual and Automatic Optimization in C++
- Intro Finite Difference Method

Manual and Automatic Optimization in C++

- Code snippets
- Identify manual/automatic optimizations

Do Less Work

- Compiler can help? No.
- Why not? Too high-level logic.

```
// original
bool vecInvalid = false;
for(const auto& item: vec){
  if(isInvalid(item))
    vecInvalid = true;
}

// break loop on first invalid item
bool vecInvalid = false;
for(const auto& item: vec){
  if(isInvalid(item)){
    vecInvalid = true;
    break;
  }
}
```

Eliminate Scope-Invariant Expressions

- Compiler can do it? Yes, if …
- … foo() does not have any side effects & compiler is allowed to reorder mathematical operators according to associative rules.

```
// original
const double tmp = 2.0;
for(auto& item: vec){
  item = ((item + tmp) + foo(0.5));
}

// remove loop constants from the loop
const double tmp = 2.0;
const double tmp2 = (tmp + foo(0.5));
for(auto& item: vec){
  item = (item + tmp2);
}
```

Memoization

- Compiler can help? No.
- Why not? Too high-level logic to analyze robustly.

```
// original
double eval[2] = {0.3265,0.6533};
for (auto& item: vec){
  switch(item.type)
    case 0:
      item.value = expensiveFoo(eval[0]); break;
    case 1:
      item.value = expensiveFoo(eval[1]); break;
}
// create table for function
double fooTab[2] = {0,0};
fooTab[0] = expensiveFoo(0.3265);
fooTab[1] = expensiveFoo(0.6533);
for (auto& item: vec){
  switch(item.type)
    case 0:
      item.value = fooTab[0]; break;
    case 1:
      item.value = fooTab[1]; break;
}
```

Data Types

- Compiler can help? No.
- Why not? Compiler does not change types.

```
// original, vecB only stores one of 0,1,2,4
std::vector<float> vecA(N,0);
std::vector<int> vecB(N,0);
for(int i=0; i!=N;++i) vecB[i]=i%4;
for(int i=0; i!=N;++i){
  vecA[i] = vecB[i] * vecC[i];
}

// use int8 instead if default int32
std::vector<float> vecA(N,0);
std::vector<uint8_t> vecB(N,0);
for(int i=0; i!=N;++i) vecB[i]=i%4;
for(int i=0; i!=N;++i){
  vecA[i] = vecB[i] * vecC[i];
}
```

Loop Splitting

- Compiler can help? Yes.
- Should I do it by hand? Yes, readability did not suffer. Might not result in speedup.

```
// original
for(int i=0;i!=N;++i){
  if(i<N/2){
    item = func1(item);
  } else {
    item = func2(item);
  }
}
// use two loops instead of if/else
for(int i=0;i<N/2;++i){
  item = func1(item)
}
for(int i=N/2;i<N;++i){
  item = func2(item)
}
```

Loop Unrolling

- Compiler can help? Yes.
- Should I do it by hand? Readability suffers. In most cases, better try to help the compiler do it for you.

```
// original
double sum = 0;
for(int i=0; i!=N; ++i){
  sum += A[i];
}
// 4x unrolled
double sum = 0;
for(int i=0; i<N; i+=4){
  sum += A[i+0];
  sum += A[i+1];
  sum += A[i+2];
  sum += A[i+3];
}
for(int i=4*(N/4); i!=N; ++i){
  sum += A[i];
}
```

Data Access

- Compiler can help? Maybe.
- Should I do it by hand? Yes.

```
// original, stride-N access
double sum;
std::vector<float> A(N*N,1);
for(int i=0; i!=N; ++i){
  for(int j=0; j!=N; ++j){
    sum += A[i+N*j];
  }
}
// stride-1 access
double sum;
std::vector<float> A(N*N,1);
for(int j=0; j!=N; ++j){
  for(int i=0; i!=N; ++i){
    sum += A[i+N*j];
  }
}
```

Data Locality

- Compiler can help? Maybe.
- Should I do it by hand? Yes.

```
// original
std::vector<float> A(N,1);
std::vector<float> B(N,1);
for(int i=0; i!=N; ++i){
   A[i] = A[i] - B[i];
}
for(int i=0; i!=N; ++i){
   A[i] = cos(A[i]);
}
// jammed loop
for(int i=0; i!=N; ++i){
   A[i] = cos(A[i]+B[i]);
}
```

Temporaries

- Compiler can help? Partly.
- What can be done? Explicit method, expression templates

```
// vector class
using Vec = std::vector<double>;
Vec operator+(Vec const &a, Vec const &b){
    size_t N = a.size();
    Vec res(N);
    for(size_t i=0;i!=N;++i)
      res.data[i] = a.data[i] + b.data[i];
    return res;
  }
// invocation
Vec a(N), b(N), c(N);
Vec res = ((a + b) + c); // two temporaries of Vec
// ideal implementation w/o temporaries
for(size_t i=0;i!=N;++i)
  res.data[i] = a.data[i] + b.data[i] + c.data[i];
```

Misc

- Investigate effect of compiler flags: Godbolt.org
  - Clang12: “-Ofast -v -Rpass=loop-vectorize -march=haswell”

```
float sdot(const int n, const float *x, const float *y){
  float s = 0.0;
#pragma clang loop vectorize(enable)
  for (int i = 0; i < n; ++i)
  {
    s += x[i] * y[i];
  }
  return s;
}
```

- Talk by developer of Godbolt.org
  - https://youtu.be/bSkpMdDe4g4

Summary: Manual and Automatic Optimization

- “Manual optimization” is advisable in most cases only after …
  - estimating the optimization potential,
  - benchmarking current performance,
  - benchmarking optimization potential, and
  - analysis of optimization attempts/successes/failures in compiler logs.
- Floating point math is not associative
  - Compilation complying to the IEEE floating point standard leads to reproducible results on different platforms (or debugging)
  - Allowing to reorder according to associative rules is required to enable many advanced automatic optimizations
- Data Access / Data Locality
  - Access data sequentially (spatial locality)
  - Reuse data residing in caches (temporal locality)
- Advanced C++ language constructs
  - Tradeoff: Software Design vs. Runtime Overhead
  - Advanced C++ knowledge is required to combine good software design with little runtime overhead

Finite Difference Method

- Example partial differential equation (PDE): Heat Equation
- Domain and constraints
- Finite difference method (FDM)
- Spatial domain discretization
- Finite differences
- Linear equation system

Example Differential Equation

- 0. 1D heat equation
- 1. Generalized spatial dimensions
- 2. Homogeneous material
- 3. Steady-state (Poisson's equation)
- 4. No heat sources (Laplace's equation)
- 5. Reformulation
- 6. Reformulation
- 7. Renaming
- 8. Two spatial dimensions
- 9. Drop k

$$0.\quad \overbrace{\rho c_p \frac{\partial T}{\partial t}}^{\text{Heat Change}} - \overbrace{\frac{d}{dx}\left(k \frac{dT}{dx}\right)}^{\text{Heat Flow}} = \overbrace{q_V}^{\text{Heat Gen.}}$$

$$1.\quad \rho c_p \frac{\partial T}{\partial t} - \nabla \cdot \left(k \nabla T\right) = q_V$$

$$2.\quad \rho c_p \frac{\partial T}{\partial t} - k\nabla \cdot \left(\nabla T\right) = q_V$$

$$3.\quad -k\nabla \cdot \left(\nabla T\right) = q_V$$

$$4.\quad -k\nabla \cdot \left(\nabla T\right) = 0$$

$$5.\quad -k\nabla^2 T = 0$$

$$6.\quad -k\Delta T = 0$$

$$7.\quad -k\Delta u = 0$$

$$8.\quad -k(u_{xx} + u_{yy}) = 0$$

$$9.\quad -(u_{xx} + u_{yy}) = 0$$

Domain and Constraints

- Model: 2D Laplace equation
- Domain
  - Spatial
  - (Temporal)
- Constraints
  - Boundary conditions
    - First-type/Dirichlet: value
    - Second-type/Neumann: derivative
    - Third-type/Robin: combination
  - (Initial conditions)

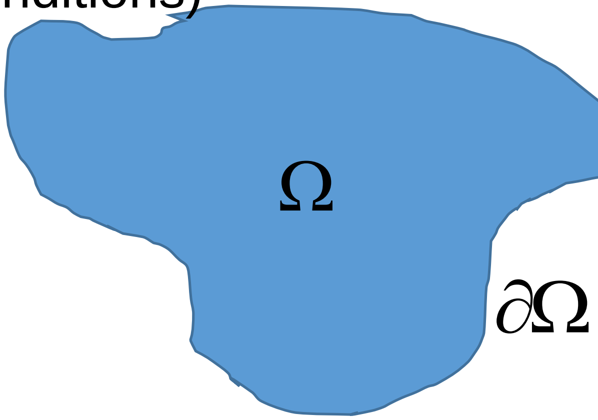
$$-(u_{xx} + u_{yy}) = 0 \quad in \quad \Omega$$

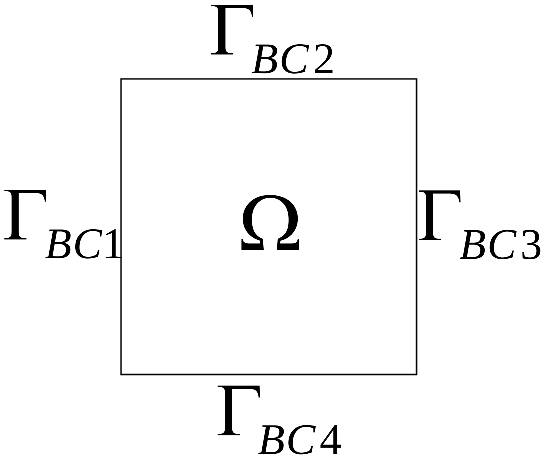
$$u = 0 \quad on \quad \Gamma_{D_{\mathrm{hom}}} \subset \Omega$$

$$u = f \quad on \quad \Gamma_{D_{inh}} \subset \Omega$$

$$\frac{\partial u}{\partial n_\Gamma} = 0 \quad on \quad \Gamma_{N_{\mathrm{hom}}} \subset \Omega$$

$$\frac{\partial u}{\partial n_\Gamma} = g \quad on \quad \Gamma_{N_{inh}} \subset \Omega$$





Discretization Methods

- Popular approaches
  - FDM: Finite Difference Method
  - (FVM: Finite Volume Method)
  - (FEM: Finite Element Method)

$$-(u_{xx} + u_{yy}) = 0 \quad in \quad \Omega$$
$$+ Boundary\ Conditions$$

- All three approaches lead to a system of linear equations
  - Matrix notation

$$A_h u_h = b_h$$

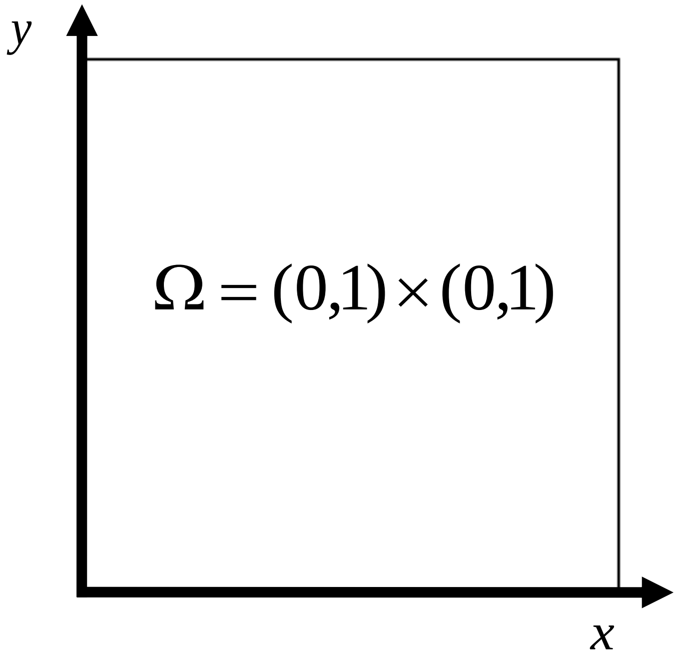
  - Matrix A_h is typically a sparse matrix
  - Vector u_h is the solution to the system and approximates u at discrete points in the domain
  - Vector b_h is constructed according to the boundary conditions
  - The approximation is called *consistent* if the solution u_h approaches the original solution u better and better if the resolution h is refined

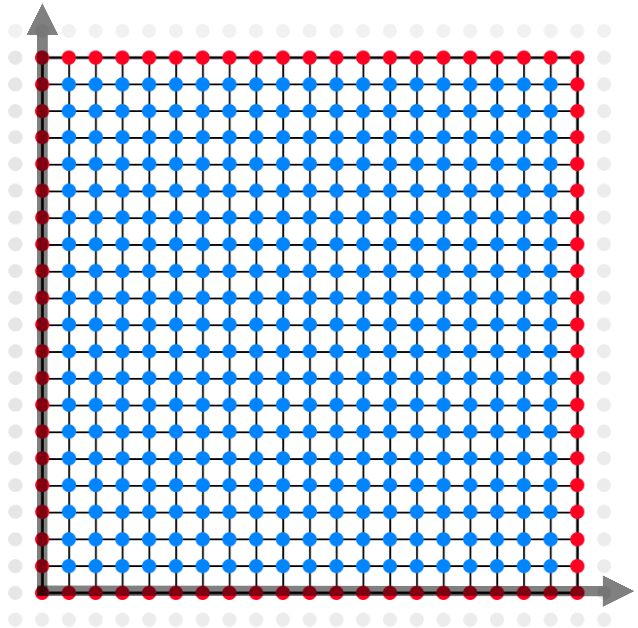
Finite Difference Method

- Approximate solution at a discrete set of points in the domain
  - Spatial domain: typically, a regular rectangular grid is used
  - (Temporal domain: implicit and explicit methods, adaptive time steps)
- Approximate derivatives with differences
  - Local neighbors of a grid point are used to construct approximations
- Attractive scheme mostly if
  - the spatial domain is a rectangular region, and
  - a regular rectangular grid is suitable for the application
- because then
  - the difference approximations of derivatives are straightforward to construct,
  - boundary conditions can be incorporated easily, and
  - the discretization error is nice to analyze.

Spatial Domain Discretization

- 2D Laplace equation $\quad -\Delta u = -(u_{xx} + u_{yy}) = 0 \quad in \quad \Omega$
- Unit square domain $\quad + \, Dirichlet \;\; Boundary \; Conditions$
- # grid points in one dimension: N
- Resulting grid spacing: $h = 1/(N-1)$
- # unknowns: $N^2 - (4N-4)$





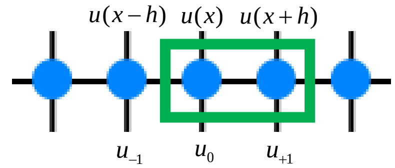
Finite Differences

- First derivative in x (analog for y)

$$-(u_{xx}+u_{yy})=0 \quad in \quad \Omega$$

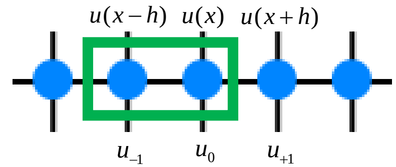
  - Forward

$$\frac{\partial u}{\partial x}=u_{+x}\approx\frac{u_{+1}-u_0}{h}$$



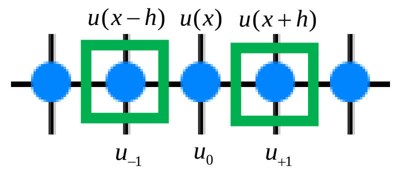
  - Backward

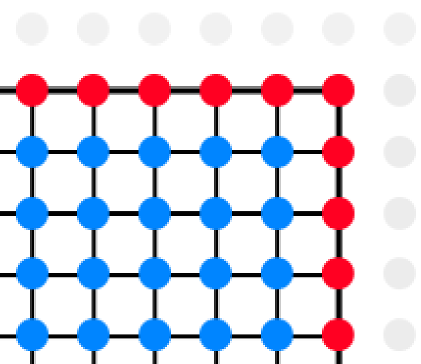
$$\frac{\partial u}{\partial x}=u_{-x}\approx\frac{u_0-u_{-1}}{h}$$

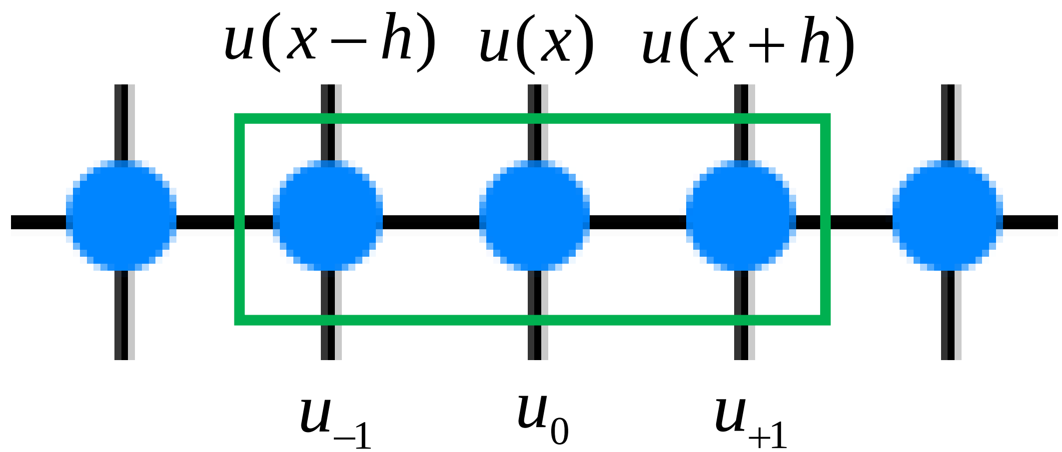


  - Center

$$\frac{\partial u}{\partial x}=u_{\pm x}\approx\frac{u_{+1}-u_{-1}}{2h}$$







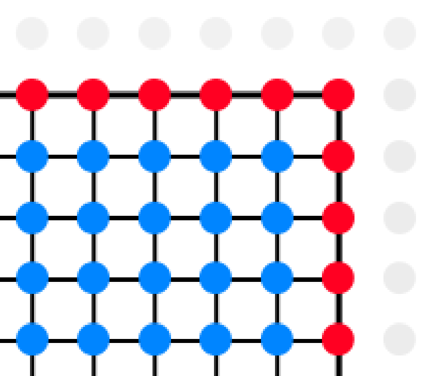
Finite Differences

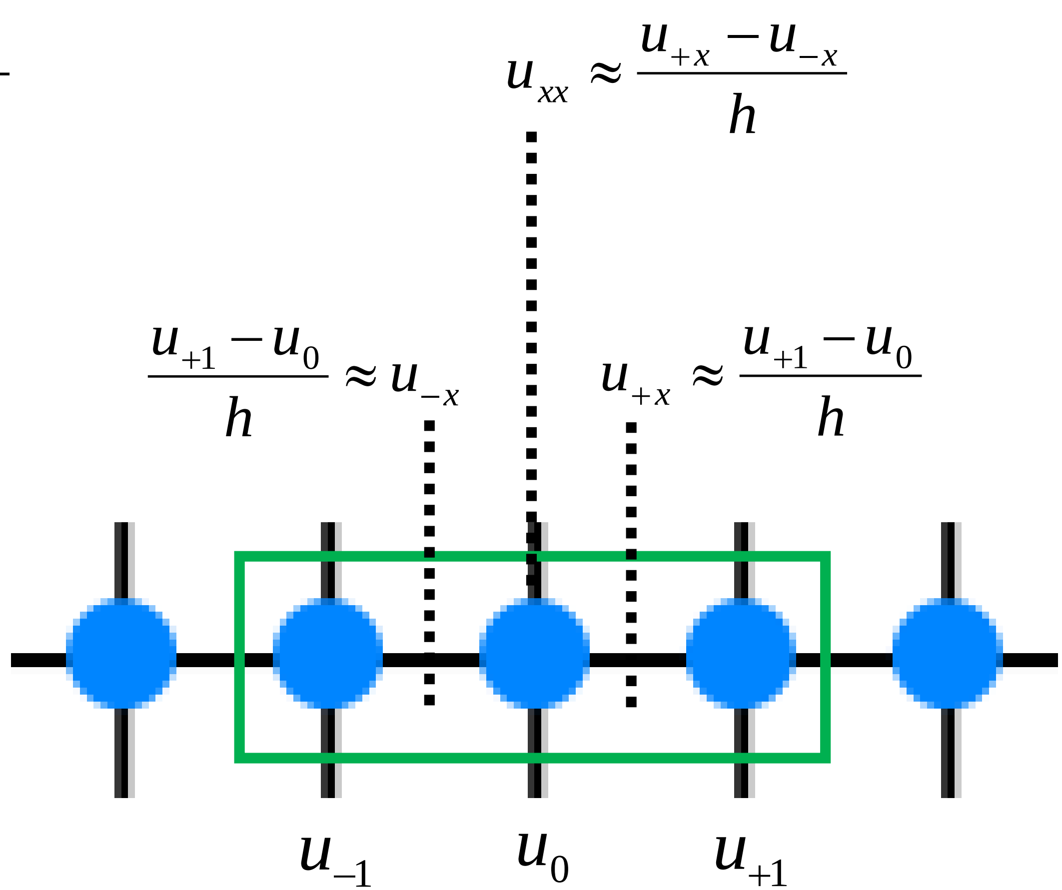
- Second derivative in x (analog for y) $\quad -(u_{xx}+u_{yy})=0 \quad in \quad \Omega$

$$\frac{\partial^2 u}{\partial x^2} = u_{xx} \approx \frac{u_{+x}-u_{-x}}{h}$$

$$= \frac{\dfrac{u_{+1}-u_0}{h} - \dfrac{u_0-u_{-1}}{h}}{h}$$

$$= \frac{u_{+1}-2u_0+u_{-1}}{h^2}$$



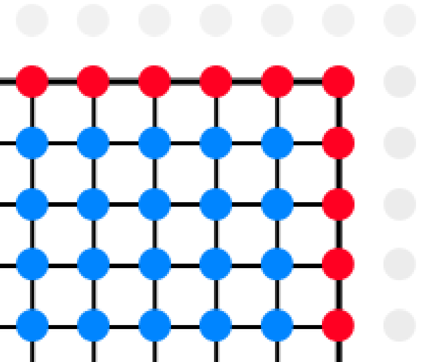


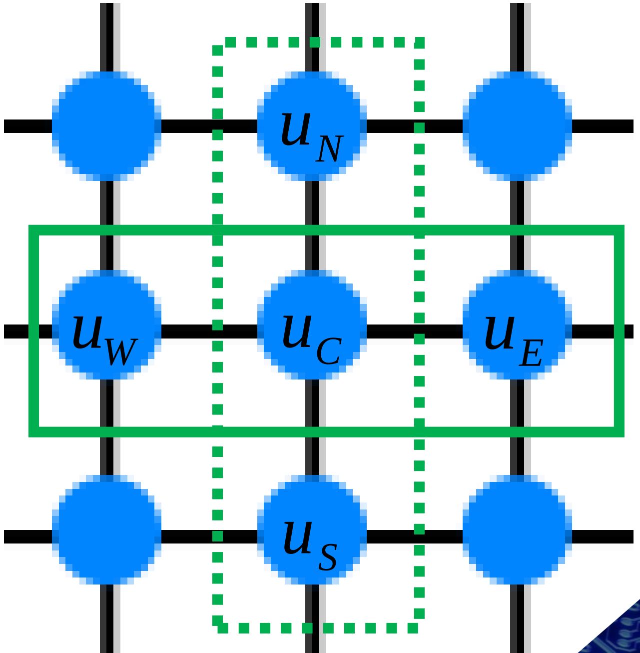
Finite Differences

- Full equation $$-(u_{xx}+u_{yy})=0 \quad in \quad \Omega$$

$$-(u_{xx}+u_{yy}) \approx -\left(\frac{(u_N - 2u_C + u_S)}{h^2} + \frac{(u_W - 2u_C + u_E)}{h^2} \right)$$

$$= -\frac{1}{h^2}(u_N + u_S + u_E + u_W - 4u_C)$$

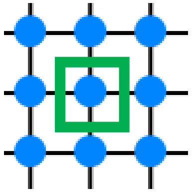




Linear Equation System

- For interior points with interior neighbors

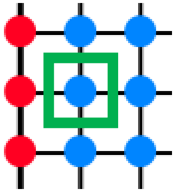
$$-\frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = 0$$



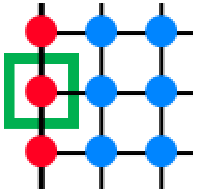
- For interior points with interior and boundary neighbors

$$-\frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = 0$$

$$-\frac{1}{h^2}\left(u_N + u_S + u_E \qquad\qquad - 4u_C\right) = \frac{1}{h^2}u_W$$

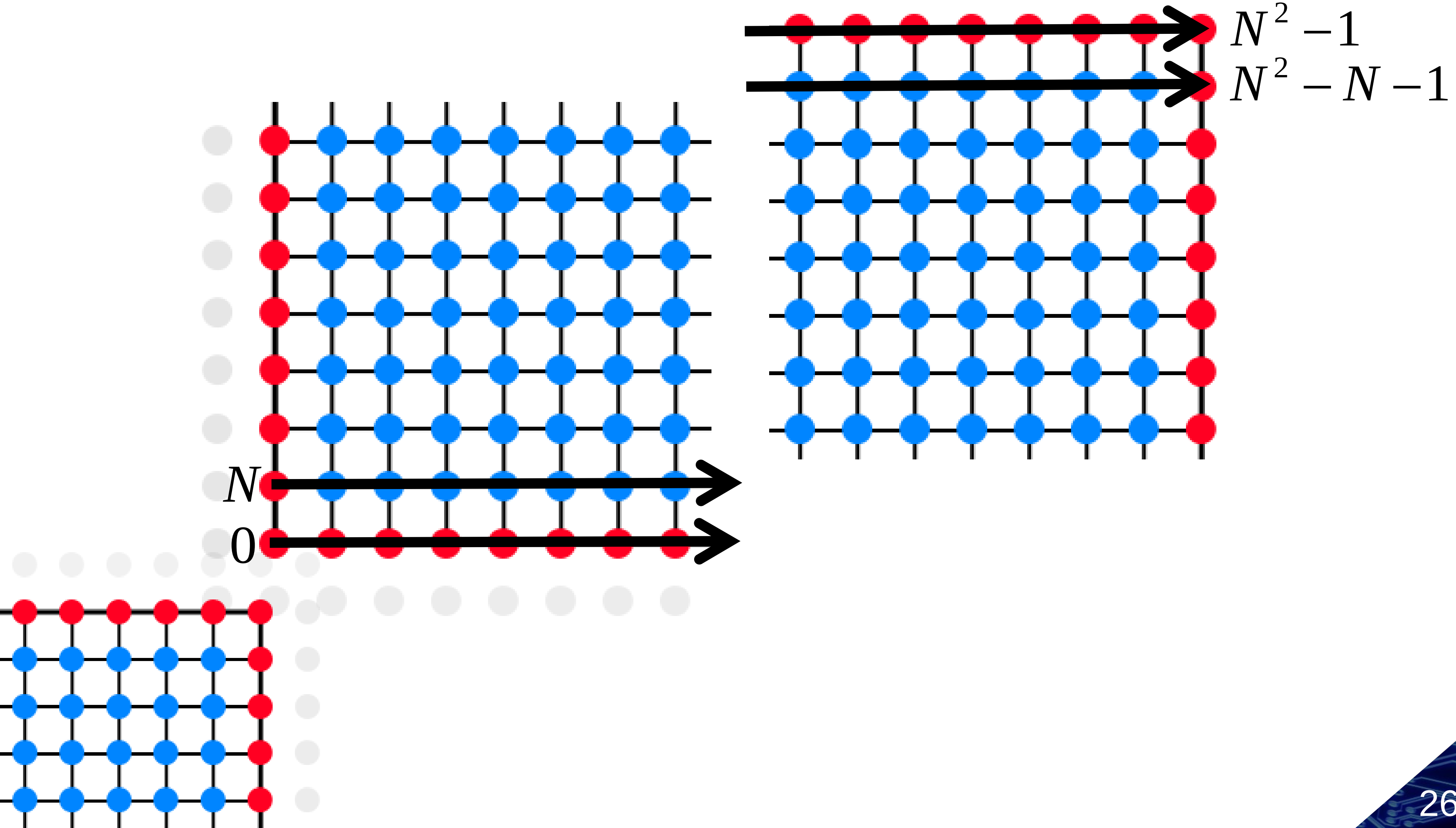


- For boundary points
  - Value prescribed by Dirichlet boundary condition



Linear Equation System

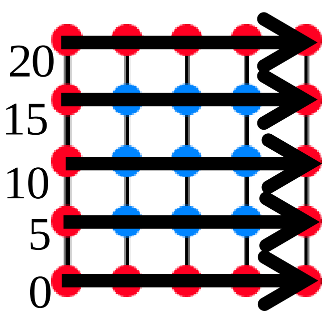
- Ordering scheme for points influences arrangement of equations in the system



Linear Equation System

- Using ordering scheme illustrated below

$$
\underbrace{\frac{1}{h^2}\begin{bmatrix}
+4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & +4 & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & +4 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & +4 & -1 & 0 & -1 & 0 \\
0 & 0 & -1 & 0 & -1 & +4 & 0 & 0 & -1 \\
0 & 0 & 0 & -1 & 0 & 0 & +4 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4
\end{bmatrix}}_{A_h}
\cdot
\underbrace{\begin{bmatrix} u_6 \\ u_7 \\ u_8 \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{16} \\ u_{17} \\ u_{18} \end{bmatrix}}_{x_h}
=
\underbrace{\frac{1}{h^2}\begin{bmatrix} u_1 + u_5 \\ u_2 \\ u_3 + u_9 \\ u_{10} \\ 0 \\ u_{14} \\ u_{15} + u_{21} \\ u_{22} \\ u_{19} + u_{23} \end{bmatrix}}_{b_h}
$$



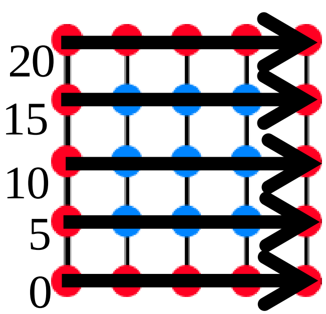
$$-\frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = 0$$

$$\frac{1}{h^2}\left((-1)u_N + (-1)u_S + (-1)u_E + (-1)u_W + (+4)u_C\right) = 0$$

Linear Equation System

- Patterns

$$\frac{1}{h^2}\underbrace{\begin{bmatrix} +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & +4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & +4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & +4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & +4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 \end{bmatrix}}_{A_h} \cdot \underbrace{\begin{bmatrix} u_6 \\ u_7 \\ u_8 \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{16} \\ u_{17} \\ u_{18} \end{bmatrix}}_{x_h} = \underbrace{\frac{1}{h^2}\begin{bmatrix} u_1 + u_5 \\ u_2 \\ u_3 + u_9 \\ u_{10} \\ 0 \\ u_{14} \\ u_{15} + u_{21} \\ u_{22} \\ u_{19} + u_{23} \end{bmatrix}}_{b_h}$$



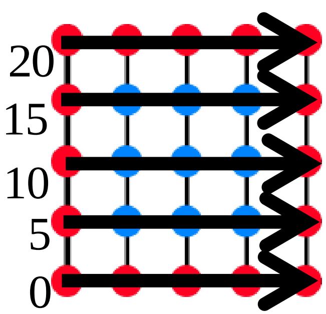
$$-\frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = 0$$

Linear Equation System

- Block matrix notation

$$D=\begin{bmatrix}+4 & -1 & 0\\ -1 & +4 & -1\\ 0 & -1 & +4\end{bmatrix}\quad I=\begin{bmatrix}+1 & 0 & 0\\ 0 & +1 & 0\\ 0 & 0 & +1\end{bmatrix}\quad A_h=\frac{1}{h^2}\begin{bmatrix}D & -I & 0\\ -I & D & -I\\ 0 & -I & D\end{bmatrix}$$

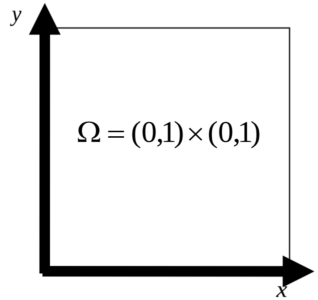
$$\underbrace{\frac{1}{h^2}\begin{bmatrix}+4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0\\ -1 & +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0\\ 0 & -1 & +4 & 0 & 0 & -1 & 0 & 0 & 0\\ -1 & 0 & 0 & +4 & -1 & 0 & -1 & 0 & 0\\ 0 & -1 & 0 & -1 & +4 & -1 & 0 & -1 & 0\\ 0 & 0 & -1 & 0 & -1 & +4 & 0 & 0 & -1\\ 0 & 0 & 0 & -1 & 0 & 0 & +4 & -1 & 0\\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 & -1\\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4\end{bmatrix}}_{A_h}\cdot\underbrace{\begin{bmatrix}u_6\\ u_7\\ u_8\\ u_{11}\\ u_{12}\\ u_{13}\\ u_{16}\\ u_{17}\\ u_{18}\end{bmatrix}}_{x_h}=\underbrace{\frac{1}{h^2}\begin{bmatrix}u_1+u_5\\ u_2\\ u_3+u_9\\ u_{10}\\ 0\\ u_{14}\\ u_{15}+u_{21}\\ u_{22}\\ u_{19}+u_{23}\end{bmatrix}}_{b_h}$$

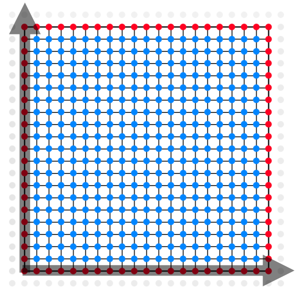


$$-\frac{1}{h^2}\left(u_N+u_S+u_E+u_W-4u_C\right)=0$$

Summary: Finite Difference Method

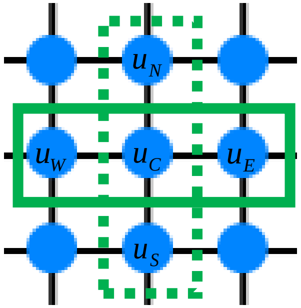
- Starting point $\quad -\Delta u = -(u_{xx} + u_{yy}) = 0 \quad in \quad \Omega = (0,1)\times(0,1) \quad + Dirichlet\ \ BCs$
- Domain discretization
  - N points per dimension
  - ~N*N unknowns





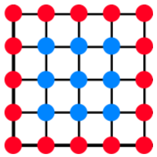
- Approximation of (second) derivatives using FD

$$-\left(u_{xx} + u_{yy}\right) \approx -\frac{1}{h^2}\left(u_N + u_S + u_E + u_W - 4u_C\right) = 0$$



- Construction of linear equation system
  - A_h has size ~ (N*N x N*N)

$$A_h \cdot x_h = b_h$$



$$\underbrace{\frac{1}{h^2}\begin{bmatrix} +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & +4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & +4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & +4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & +4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & +4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & +4 \end{bmatrix}}_{A_h} \cdot \underbrace{\begin{bmatrix} u_6 \\ u_7 \\ u_8 \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{16} \\ u_{17} \\ u_{18} \end{bmatrix}}_{x_h} = \underbrace{\frac{1}{h^2}\begin{bmatrix} u_1 + u_5 \\ u_2 \\ u_3 + u_9 \\ u_{10} \\ 0 \\ u_{14} \\ u_{15} + u_{21} \\ u_{22} \\ u_{19} + u_{23} \end{bmatrix}}_{b_h}$$

Quiz

Q1: Does C++ abstraction (classes, operator overloads, …) lead to run time overhead?

Q2: How many choices are there to approximate a first derivative on a regular finite difference grid?

Q3: How big is the memory footprint of a 3D finite difference grid with uniform resolution of 1024 along each dimension?

Q4: Assume someone provides you a solution u_h to a specific linear equation system arising from applying the FDM to a problem, what could you do to check if it is indeed a solution?

Q5: Assume you were able to check that u_h in Q4 is indeed a solution to the discretized problem, do you expect addtitional error terms when comparing against an analytic solution (assume is exists) of the problem?