Today’s lecture

• Application study: mesh based methods
• We’ll start with the 1 dimensional case
• Then generalize to higher dimensions
• Models of performance
Mesh based methods

- Many physical problems are simulated on a uniform mesh in 1, 2 or 3 dimensions
- Values are defined on a discrete set of points
- A mapping from ordered pairs to physical observables like temperature and pressure
- One application: differential equations

Differential equations

- A differential equation is a set of equations involving derivatives of a function (or functions), and specifies a solution to be determined under certain constraints
- Constraints often specify boundary conditions or initial values that the solution must satisfy
- When there are functions of a single variable we have an Ordinary Differential Equation (ODE)
  \[ u'(x) = \cos(x), u(0) = 0, u(\pi) = 0, \text{ i.e. } u(x) = -\sin(x) \]
- When there are functions of a multiple variables we have a Partial Differential Equation (PDE)
  \[ u_{xx} + u_{yy} = 0 \text{ within a square box} \]
  \[ u = 1 \text{ on perimeter of the box} \]
Poisson’s Equation

- Ubiquitous equation
  - Fluid flow, gravitational interaction, ...
- In two dimensions: a PDE
  solve \( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \), \( x,y \in [0,1] \)
  subject to boundary conditions
  \( u(x,y) = \sin(x)\sin(y) \) on \( \partial \Omega \)
  We refer to \([0,1] \times [0,1]\) as the domain
- In one dimension: ordinary differential eqn (ODE)
  solve \(-u''(x) = f(x), x \in [0,1]\)
  subject to the boundary condition
  \( u(0) = a, u(1) = b \)

Solving an ODE

- Consider the 1D case:
  \(-u''(x) = f(x), x \in [0,1]\)
- Make a discrete approximation
  Define \( u(x) \) at points \( i^*h, h=1/(N-1) \)
  Let \( u_i = u(i^*h) \)
System of equations

\[-u''(x) = f(x), \ x \in [0, 1]\]

- Approximation to \(u''(x)\)
  \[u''(x) \approx (u(x+h) - 2u(x) + u(x-h))/h^2\]
  We derive this approximation using Taylor series
- Obtain a system of equations
  \[(-u_{i-1} + 2u_i - u_{i+1})/h^2 = f_i, \ i \in 1..n-1\]

Iterative method for solving systems of equations

- Rewrite the system of equations
  \[(-u_{i-1} + 2u_i - u_{i+1})/h^2 = f_i, \ i \in 1..n-1\]
- It can be shown that the following Gauss-Seidel algorithm will arrive at the solution …
- assuming an initial guess for the \(u_i\)
  Repeat until the result is satisfactory
    for \(i = 1 : N-1\)
      \[u_i = (u_{i+1} + u_{i-1} + h^2 f_i)/2\]
    end for
  end Repeat
Accuracy

- When is the answer is “satisfactory?”
  - The computed solution has reached a reasonable approximation to the exact solution
- We can validate the computed solution in the field, i.e. wet lab experimentation
- We often don’t know the exact solution, and must estimate the error

Using the residual to estimate the error

- Recall the equations
  \[\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, \ i \in 1..n-1\]
- Define the residual \(r_i\):
  \[r_i = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} - f_i, \ i \in 1..n-1\]
- Thus, our computed solution is correct when \(r_i = 0\)
- We can obtain a good estimate of the error by finding the maximum \(r_i \ \forall i\)
- Another possibility is to take the root mean square (L2 norm)
  \[\sqrt{\sum r_i^2}\]
Parallel implementation

• An attempt to parallelize the Gauss-Seidel algorithm in a simple way will fail, since there are loop carried dependences
• The value of \( u[i] \) computed in iteration \( i \) depends on \( u[i] \) computed in iteration \( i-1 \)

\[
\text{for } i = 1 : N-1 \\
u[i] = (u[i-1]+u[i+1] + h*h*f[i])/2 \\
\text{end for}
\]

Parallel implementation

• To avoid making in-place updates that inhibit parallelization we may use
  – the red-black strategy
  – Jacobi’s method
Jacobi’s method

- We don’t update the solution array in place
- There are two arrays \( u \) and \( u_{\text{new}} \), and we store updates in \( u_{\text{new}} \)
  
  \[
  \text{for } i = 1 : N-1 \\
  \quad u_{\text{new}}[i] = \frac{u[i-1]+u[i+1] + h^2 f[i]}{2}
  \]

- After each sweep, we swap \( u \) and \( u_{\text{new}} \)

---

Convergence of Jacobi’s method

![Convergence of Jacobi’s method](image-url)
Tradeoffs

- We can now parallelize the algorithm, since there are no longer any loop carried dependencies within the $i$ loop.
- However, this change reduces the convergence rate by about a factor of two.
- We have doubled the amount of work we need to perform in exchange for being able to realize parallel speedups.

Convergence check

- Each process computes the error for its assigned part of the problem.
- We need a global error so that we compute a result that is consistent with the single processor implementation.
- This requires collective communication: an all-reduction.
Code for the curious: the rb program

- Implements Gauss-Seidel Red Black algorithm
- Observe speedups and scaled speedups
  - Run on at least 4 processors, more if possible
- Vary the convergence check frequency (f) and note the effect on performance
  - Run on at least 8 processors
  - You may need to vary N, and f to observe the effect

Stencils and molecules

- We call the numerical operator that sweeps over the solution array a **stencil operator**
- In 1D we have the **finite difference molecule**: \(1 \ -2 \ 1\)
Poisson’s Equation in higher dimensions

- In 2D we seek a function of 2 variables and we therefore have partial derivatives.
- One of the simplest molecules is the so called “5 point stencil”.
- We may use Jacobi’s method or red/black ordering.

\[
\begin{pmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

for \((i,j)\) in \(1:N \times 1:N\)
where \((i+j)\) is odd/even

\[
u[i][j] = \frac{(u[i-1][j] + u[i+1][j] + u[i][j-1] + u[i][j+1] + f[i][j]*h^2)}{4} ;
\]
Processor geometries

- In two dimensions we can have 1-dimensional partitions (strips) or 2-dimensional partitions (blocks)
- We express the different partitions in terms of processor geometry
- For P processors the geometries are of the form $p_0 \times p_1$, where $P = p_0 p_1$
- For P=4, are 3 possible geometries

Ghost cells

- Non-contiguous data transfers in multidimensional processor geometries
Communication

• We need to manage communication carefully
• Why don’t we simply send a message to receive each individual point on the interior boundary?