SF2568: Parallel Computations for Large-Scale Problems

Lecture 6: Image Reconstruction and Poisson's Equation

January 30, 2024

Acknowledgements

These slides are an extension of slides by Michael Hanke and Niclas Jansson.

Introduction

Question

What does image processing and the solution of partial differential equations have in common?

Digital Images

Definition

A (digital) **image** is a $M \times N$ matrix of pixel values, the *pixmap*

- We will assume that each pixel is represented by its gray level. Thus, we assume the image to be black and white
- A coloured image consists of a collection of pixmaps
- We assume that the type of pixel is `double`. In practice, most often 8-bit values are used (`unsigned char`)

Smoothing, Sharpening, Noise Reduction

- **Smoothing**: suppresses large fluctuations in intensity over the image
- **Sharpening**: accentuates transitions and enhances the details
- **Noise reduction**: suppresses a noise signal present in an image

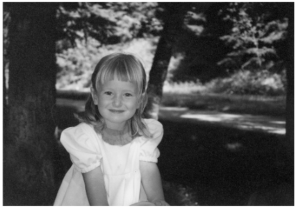
Smoothing By Local Filtering

Idea: Replace each pixel value $\tilde{u}_{mn}$ by the mean of the surrounding pixels:

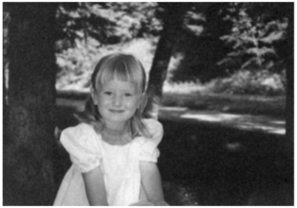
$$\begin{aligned}\tilde{u}_{mn} = \frac{1}{9}\Big(& u_{m-1,n-1} && + u_{m-1,n} && + u_{m-1,n+1} \\ + & u_{m,n-1} && + u_{m,n} && + u_{m,n+1} \\ + & u_{m+1,n-1} && + u_{m+1,n} && + u_{m+1,n+1}\Big)\end{aligned}$$

Smoothing – Example

Original



Result



Weighted Masks

The mean value can conveniently be described by a 3×3 matrix W,

$$W = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Application:

$$\begin{aligned} \tilde{u}_{mn} = \quad & w_{-1,-1}u_{m-1,n-1} + w_{-1,0}u_{m-1,n} + w_{-1,1}u_{m-1,n+1} \\ & + w_{0,-1}u_{m,n-1} + w_{0,0}u_{m,n} + w_{0,1}u_{m,n+1} \\ & + w_{1,-1}u_{m+1,n-1} + w_{1,0}u_{m+1,n} + w_{1,1}u_{m+1,n+1} \end{aligned}$$

Mathematically: Convolution

Noise Reduction

$$W = \frac{1}{16}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

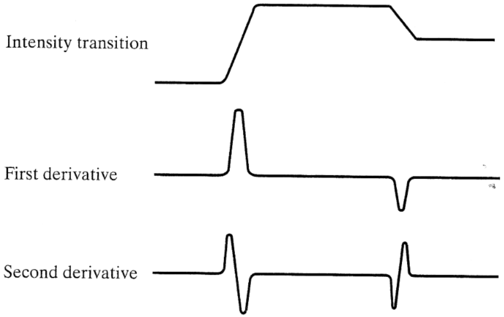
Noisy image

denoised

Edge Detection

- **Edge detection** is the highlighting of the edges of an object, where an edge is a significant change in the grey-level intensity
- **Basic idea**: The rate of change of a quantity can be measured by the **magnitude of its derivative(s)**

Example


Intensity transition
First derivative
Second derivative

The Laplace Operator

Definition

For any function u defined on some two-dimensional domain, the Laplacian Δu of u is defined as

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Approximating The Laplacian

- Approximate the derivatives,

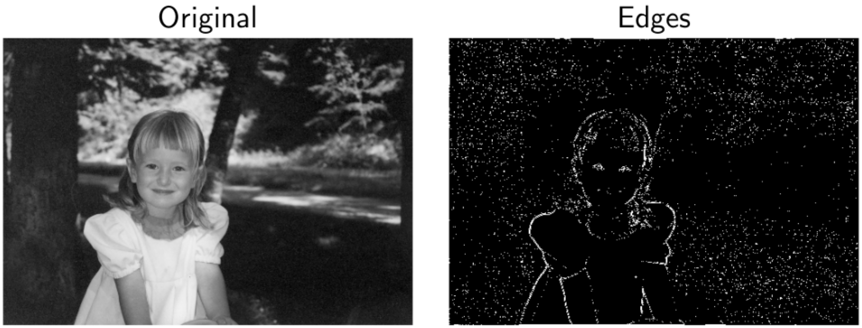
$$\frac{\partial^2 u}{\partial x^2}(x,y) \approx \frac{1}{h^2}\left(u(x-h,y) - 2u(x,y) + u(x+h,y)\right), \qquad h > 0$$

- We obtain the weight matrix,

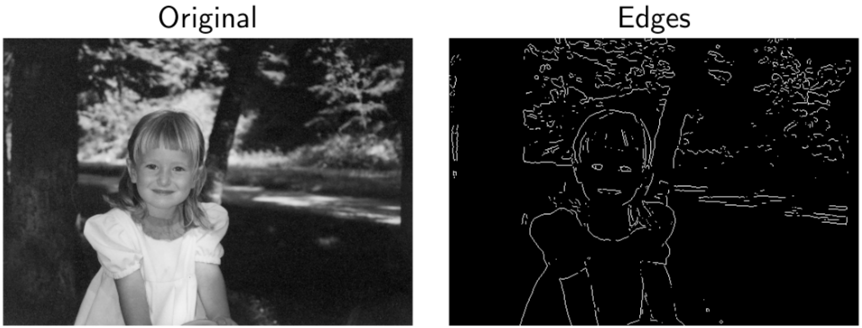
$$W = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- *This fits exactly in our framework!*

Edge Detection By The Discrete Laplacian



Using First Order Derivatives – Sobel Operator



Poisson Equation

- Ubiquitous equation
  - Fluid flow, electromagnetics, gravitational interaction,. . .
- In two dimensions, Poisson's equation reads:
  - Solve $\Delta u = f(x, y)$ for $(x, y \in \Omega)$
  - Subject to the boundary condition $u(x, y) = g(x, y)$ for $(x, y) \in \partial\Omega$
- For simplicity, consider only $\Omega = (0, 1) \times (0, 1)$
- Generalizations to other dimensions are obvious

Discrete Approximation

- Define a *mesh* (or grid): For a given N, let

$$h = 1/(N-1), \qquad x_m = mh, \qquad y_n = nh$$

- Let $u_{mn} \approx u(x_m, y_n)$, $f_{mn} = f(x_m, y_n)$
- Using the Laplace approximation from above, we obtain a system of equations

$$\frac{1}{h^2}\left(u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} - 4u_{m,n}\right) = f_{mn},$$

$$0 < m, n < N-1$$

- In the context of PDE's, the matrix W is usually called a **stencil**

Jacobi Iteration

- **Basic idea**: Rewrite the equation as

$$u_{mn} = \frac{1}{4}\left(u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} - h^2 f_{mn}\right)$$

- For some starting guess (e.g., $u_{mn} = 0$), iterate this equation,

```
while ( not_done )
  for (m,n) in 1:N-2 x 1:N-2
    ut(m,n) =( u(m-1,n) + u(m+1,n)
             +  u(m,n-1) + u(m,n+1)
             -  h^2 f(m,n)          )/4;
  end
  u = ut;
end
```

Accuracy

- How do we know that the answer is "good enough"?
    - When the computed solution has reached a reasonable approximation to the exact solution
    - When we can validate the computed solution in the field
- But often we do not know the exact solution, and must estimate the error, e.g.,
    - Stop when the residual is small enough, $r = Au - f$
    - Stop when the change $u - u'$ in u is small
    - Both approaches must be designed carefully!

Boundary Condition

- Evaluating the stencil is not possible near the boundary
- For Poisson's equation $\rightarrow$ invoke the boundary condition
- In image processing, there are two possibilities:
    1. Discard the boundary (the new image is 2 pixels smaller in both dimensions)
    2. Modify the weight matrices such that only existing neighbours are used

The Common Denominator

Conclusion

The methods considered use a **uniform mesh** for their data

- Such methods are very common in applications
- They can easily be adapted to problems of any (spatial) dimension

Observations

Observations

- The computations for each point u_{ij} are completely decoupled
- The number of operations per data point is constant
- The new value at each data point depends only on its nearest neighbours

Conclusion

- A good parallelisation strategy is data partitioning

To keep things as simple as possible, consider only a one-dimensional array (a vector)

$$u = (u_0, \ldots, u_{M-1})^T$$

Data Distribution

Definition

Assume that we have P processes (enumerated $0, \dots, P-1$). A **P-fold data distribution** of the index set $\mathcal{M} = \{0, \dots, \mathcal{M}-1\}$ is a bijective mapping μ which maps each **global index** $m \in \mathcal{M}$ to a pair of indices (p, i) where p is the process identifier and i the **local index**.

Notes

- This definition allows for the fact that the number of elements on each process varies with p. Of course, this is necessary if P does not divide M evenly
- Technically, we assume that the local index set on each process is a set of consecutive integers, often (**but not always!**) $0 \leq i < I_p$

Example – Linear Data Distribution

Idea

Split the vector into equal chunks and allocate the p-th chunk to process p

- At $p = 0:\ u_0, u_1, \ldots, u_{l_0 - 1}$
- At $p = 1:\ u_{l_0}, \ldots, u_{l_0 + l_1 - 1}$
- In general, $p:\ u_{l_{p-1}}, \ldots, u_{l_{p-1} + l_p - 1}$
- If P does not divide M evenly, distribute the remaining R elements to the first few processes

Example – Linear Data Distribution

- The load-balanced linear data distribution is:

$$
\begin{aligned}
L &= \left\lfloor \frac{M}{P} \right\rfloor \\
R &= M \text{ mod } P \\
\mu(m) &= (p, i) \text{ where } \begin{cases} p = \max\left(\left\lfloor \frac{m}{L+1} \right\rfloor, \left\lfloor \frac{m-R}{L} \right\rfloor\right) \\ i = m - pL - \min(p, R) \end{cases} \\
I_p &= \left\lfloor \frac{M+P-p-1}{P} \right\rfloor \\
\mu^{-1}(p, i) &= pL + \min(p, R) + i
\end{aligned}
$$

Example – Scatter Distribution

Idea

Allocate consecutive vector components to consecutive processes

- At $p = 0:\ u_0, u_P, u_{2P}, \dots$
- At $p = 1:\ u_1, u_{P+1}, \dots$
- In general, $p:\ u_p, u_{P+p}, \dots$

Example – Scatter Distribution

- The load-balanced scatter distribution is:

$$\mu(m) = (p, i) \text{ where } \begin{cases} p = m \text{ mod } P \\ i = \left\lfloor \dfrac{m}{P} \right\rfloor \end{cases}$$

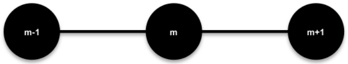
$$I_p = \left\lfloor \frac{M + P - p - 1}{P} \right\rfloor$$

$$\mu^{-1}(p, i) = iP + p$$

A Distributed Vector

- The one-dimensional version of the convolution formula reads

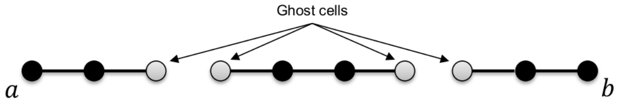
$$\tilde{u}_m = w_{-1}u_{m-1} + w_0u_m + w_1u_{m+1}$$



- Each evaluation needs its neighbours. Consequently, the **linear data distribution** is most appropriate
- Each process needs one element stored on the processes to the “left” and “right”
- This additional data is called ghost cells (or ghost points)

Ghost Cells

- Two adjacent processes p and $p+1$ need to share two data points. This is called the **overlap** between two processes, and is dependent on the **width** of the stencil
- *For Example:* Assume our convolution formula and the domain between a and b
- On a distributed memory machine we need to divide it into chunks
- The local array $u_i,\ 0 \leq i < I_p$ will be surrounded by two cells (with the exception of the first and the last processes)

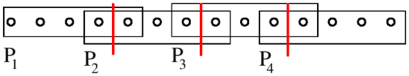


- This is conveniently done by **enlarging** the local vector

Fill The Ghost Cells – Communication

- Before we can start applying the stencil, the ghost cells must be filled
- Attempted erroneous solution (assume an overlap of two for simplicity)

```
receive(u[0],p-1);
send(u[1],p-1);
receive(u[Ip-1],p+1);
send(u[Ip-2],p+1);
```



Deadlock!

- Mismatch in communication. All processes waiting to receive
- Possible solutions:
    - Rewrite program so that calls to send and receive are matched
    - Use non-blocking communication

Communication – A New Attempt

- Exchange send and receive

```
if p > 0
  send(u[1],p-1);
  receive(u[0],p-1);
end
if p < P-1
  receive(u[Ip-1],p+1);
  send(u[Ip-2],p+1);
end
```

- Code works! But very inefficient!
  - Most processes are idle during communication
  - Possible solution: Use different communication pattern

An Efficient But Unreliable Solution

```
send(u[lp-2],p+1);
receive(u[0],p-1);
send(u[1],p-1);
receive(u[lp-1],p+1);
```

Properties:

+ communication time is optimal: $2(t_{startup} + 8t_{data})$
- Relies on the network to buffer the messages.
  This is not guaranteed by MPI!

The Safe Solution

- The idea is a red-black (checker-board) colouring:
  - Even p: assign red
  - Odd p: assign black

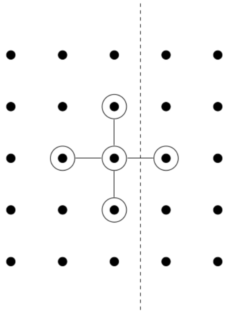
Communication appears in two steps: red/black and black/red

```
if mycolor == red
  send(u[Ip-2],p+1);
  receive(u[Ip-1],p+1);
  send(u[1],p-1);
  receive(u[0],p-1);
else
  receive(u[0],p-1);
  send(u[1],p-1);
  receive(u[Ip-1],p+1);
  send(u[Ip-2],p+1);
end
```

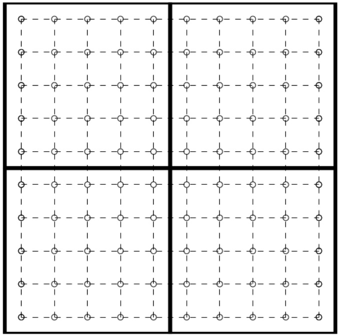
Communication time is only doubled compared to the previous

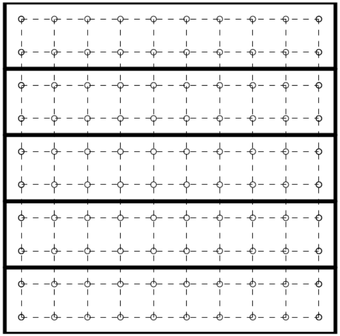
Generalizations To Two Dimensions

- Sample stencil (Poisson)
- Use an array of $R = P \times Q$ processes
- Distributed equal chunks of the pixmap/solution onto these processes
- Different partitions are called **process geometry** or **process topology**



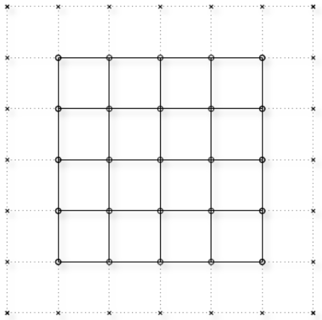
Process Topology


P=Q=2


P=1, Q=5

Ghost Cells

- Each process needs values found on neighbouring processes
- Use **ghost cells**



  - Circles: local grid points
  - Crosses: ghost points
- The data distribution is constructed individually for the x and y directions along the lines of the 1D example

Communication of Ghost Points

Question

How should the exchange of the ghost points corresponding to the inter-process boundaries be implemented?

The handling of the outer boundaries depends on the problem at hand (either ignore them or apply physical boundary conditions)

Some Notation

For a process with "coordinates" (p, q), the neighbours are defined as follows (if they exist)

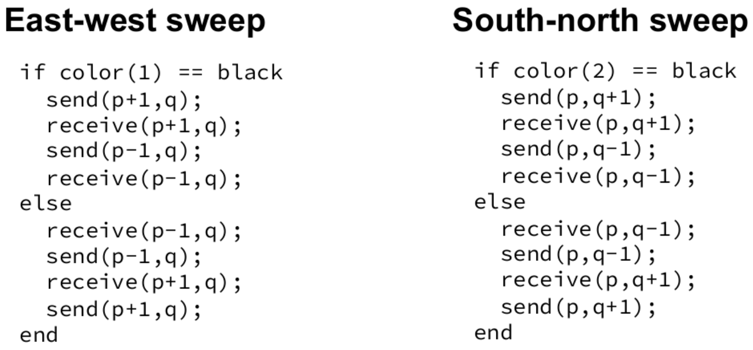
Neighbour	**Coordinates**
east	$(p+1, q)$
west	$(p-1, q)$
north	$(p, q+1)$
south	$(p, q-1)$

Non-Blocking Implementation

1. Initiate send (`MPI_Isend`) to east, west, north, and south neighbours (if present)
2. Initiate receive (`MPI_Irecv`) from west, east, south, and north neighbours
3. Evaluate the stencil away from the boundaries
4. Wait for communication to complete
5. Evaluate stencil near boundaries

Red-Black Communication

- Similar to red-black communication in 1D
- Associate each process with a color (red or black) in the p and q directions such that no neighbour has the same color



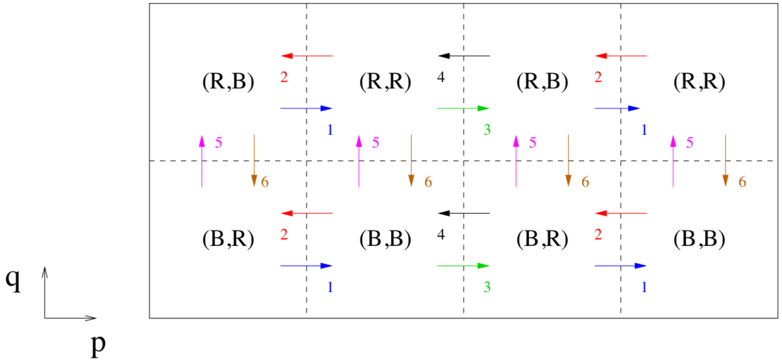
East-west sweep

```
if color(1) == black
  send(p+1,q);
  receive(p+1,q);
  send(p-1,q);
  receive(p-1,q);
else
  receive(p-1,q);
  send(p-1,q);
  receive(p+1,q);
  send(p+1,q);
end
```

South-north sweep

```
if color(2) == black
  send(p,q+1);
  receive(p,q+1);
  send(p,q-1);
  receive(p,q-1);
else
  receive(p,q-1);
  send(p,q-1);
  receive(p,q+1);
  send(p,q+1);
end
```

Red-Black Communication



Number and colours show the communication pattern process colour indicated by (q, p) (note the order)

Red-Black Communication Time

- We assume a perfectly load balanced (linear) distribution,

$$I_p \approx \frac{M}{P} \qquad J_q \approx \frac{N}{Q}$$

- East-west sweep:

$$t_{comm,1} = C(P)(t_{startup} + I_p t_{data})$$

where

$$C(P) = \begin{cases} 0, & \text{if } P = 1 \\ 2, & \text{if } P = 2 \\ 4, & \text{if } P \geq 3 \end{cases}$$

- Similarly, for the South-north sweep:

$$t_{comm,2} = C(Q)(t_{startup} + J_q t_{data})$$

- Total communication time

$$t_{comm} = (C(P) + C(Q))t_{startup} + \frac{t_{data}}{PQ}(C(P)QM + C(Q)PN)$$

Computation Time

- Assume a (compact) stencil

$$W = \begin{pmatrix} w_{-1,-1} & w_{0,-1} & w_{1,-1} \\ w_{-1,0} & w_{0,0} & w_{1,0} \\ w_{-1,1} & w_{0,1} & w_{1,1} \end{pmatrix}$$

- Let w be the number of nonzero entries in W. Then

$$t_{comm,pq} = \alpha w I_p J_q t_a \approx \alpha w \frac{MN}{PQ} t_a$$

  ($0 < \alpha$ is a small constant)

- Best sequential time

$$T_S^* = \alpha w M N t_a$$

Speedup

$$\begin{aligned} S_R = S_{PQ} &= \frac{T_S^*}{T_R} \\ &\geq R\frac{\alpha w M N t_a}{\alpha w M N t_a + 8Rt_{startup} + 4(QM + PN)t_{data}} \\ &\geq R\frac{1}{1 + \frac{8Rt_{startup}}{\alpha w M N t_a} + \frac{4}{\alpha w}\left(\frac{P}{M} + \frac{Q}{N}\right)\frac{t_{data}}{t_a}} \end{aligned}$$

Conclusion

- For constant R, the speedup reaches an optimal value if MN becomes large
- If MN is fixed, the speedup will eventually degrade if R gets larger
- The speedup becomes better if $(P/M + Q/N)$ attains a minimum for a given problem size and a given number of processes

Optimal Process Topology

- For a given problem size MN and a given number of processes R, find P and $Q = R/P$ such that

$$\phi(P) = \left(\frac{P}{M} + \frac{Q}{N}\right)$$

  becomes minimal

- A simple calculation gives

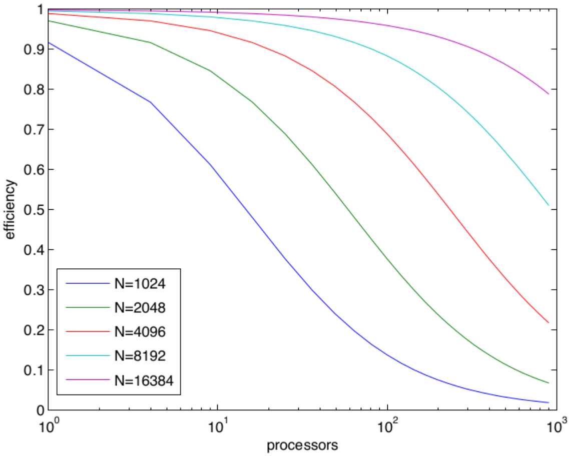
$$P = \sqrt{\frac{M}{N}R}$$

  (provided that these are integers)

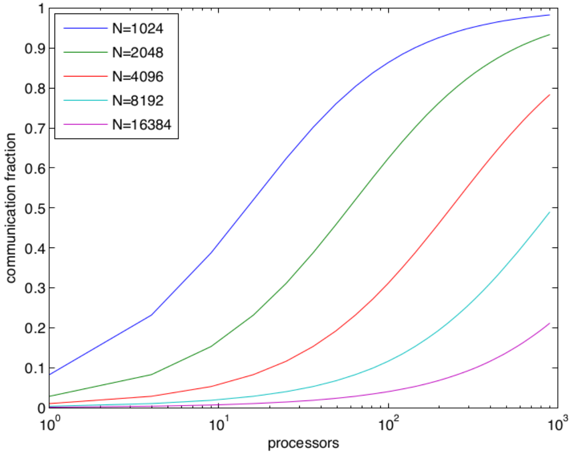
- In the case $M = N$ and R being a square, $P = \sqrt{R}$

Efficiency

For typical data on Lucidor (at PDC), this is the efficiency $E_R = S_R/R$



Communication Fraction



Surface to Volume Ratio

Observation:

- The computation time t_{comp} is proportional to the area $I_p \times J_q$ of the data
- The *communication time* t_{comm} is proportional to the **perimeter** $2(I_p + J_q)$

"Area-perimeter law"

The communication time is negligible if the number of data $M \times N$ is large compared to the number of processes

The Curse of Dimensionality

As we move to higher dimensional spaces, communication becomes relatively more costly,

- In 1D: $2/N$
- In 2D: $4N/N^2 = 4/N$
- In 3D: $6N^2/N^3 = 6/N$

Very important to find overlapping possibilities in 3D!

Virtual Topologies

Virtual Topologies

MPI includes a number of standard routines for defining and handling different process topologies. They are called **virtual topologies**. These routines lead to great simplification of the programming efforts needed

Jacobi Iteration

- **Basic idea**: Rewrite the equation as

$$u_{mn} = \frac{1}{4}\left(u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} - h^2 f_{mn}\right)$$

- For some starting guess (e.g., $u_{mn} = 0$), iterate this equation,

```
while ( not_done )
  for (m,n) in 1:N-2 x 1:N-2
    ut(m,n) =( u(m-1,n) + u(m+1,n)
             +  u(m,n-1) + u(m,n+1)
             -  h^2 f(m,n)          )/4;
  end
  u = ut;
end
```

Gauss-Seidel Iteration

Observation:

- The Jacobi iteration converges very slowly

$$u_{mn} = \frac{1}{4}\left(u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} - h^2 f_{mn}\right)$$

Idea:

- Use the new (better?) values as soon as they are available

Gauss-Seidel Iteration

```
while ( not_done )
  for (m,n) in 1:N-2 x 1:N-2
    u(m,n) =( u(m-1,n) + u(m+1,n)
            +  u(m,n-1) + u(m,n+1)
            -  h^2 f(m,n)          )/4;
  end
  % u = ut;
end
```

Observation: This iteration depends on the order of the unknown!

Lexicographic

Definition

The lexicographic order of the array u_{mn} is given by

$$u_{11}, u_{21}, u_{31}, \ldots, u_{M1}, u_{12}, u_{22}, \ldots, u_{MN}$$

Lexicographic order from the iteration is not safe

```
for n = 1:N
  for m = 1:M
    u(m,n) = ...
  end
end
```

Pipelined Computations

- Gauss-Seidel iterations are purely sequential
- Assume a $P \times Q$ process grid as before
- Process (p, q) cannot start computing before the values on processes $(p-1, q)$ and $(p, q-1)$ are available
- This leads to **pipelined computations**
- At every moment in time, only the processes along diagonals are active . . .

How to Parallelize?

Idea: Use red-black ordering!

- Black points: $m + n$ is even
- Red points : $m + n$ is odd
- Gauss-Seidel iteration

$$u_{mn} = \frac{1}{4}\left(u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} - h^2 f_{mn}\right)$$

- If u_{mn} is black, the values on the right hand side are all red and vice versa
- The "black sweep" and the "red sweep" can be parallelized independently
- Note: This is a different kind of iteration!

Final Remarks

- More efficient methods for solving Poisson's equation include multigrid methods (asymptotical optimal!)
- For a full 9-point stencil, four colours are needed
- Today, the most complex parallel circuit in a PC is the GPU (graphic processing unit)
- Not surprisingly, the GPU is used as a parallel solver unit even for PDEs
- MPI includes the possibility to define virtual topologies thus simplifying the design of the communication a lot

What did we learn?

- Evaluation of stencils for different purposes (image processing, solutions of partial differential equations)
- Data distributions, ghost points, practical aspects
- Efficient communication strategies
- Performance evaluation of the corresponding algorithms
- Pipelined computations (Gauss-Seidel iterations)
- Reformulation of recursive algorithms