AMath 483/583 — Lecture 21 — May 13, 2011

Today:

- OpenMP and MPI versions of Jacobi iteration
- Gauss-Seidel and SOR iterative methods

Next week:

- More MPI
- Debugging and totalview
- GPU computing

Read: Class notes and references

$CLASSHG/codes/openmp/jacobi1.f90

$CLASSHG/codes/openmp/jacobi2_omp.f90

$CLASSHG/codes/mpi/jacobi2_mpi.f90

Jacobi iteration

$$(U_{i-1} - 2U_i + U_{i+1}) = -\Delta x^2 f(x_i)$$

Solve for U_i:

$$U_i = \frac{1}{2}\left(U_{i-1} + U_{i+1} + \Delta x^2 f(x_i)\right).$$

Note: With no heat source, $f(x) = 0$,
the temperature at each point is average of neighbors.

Jacobi iteration

$$(U_{i-1} - 2U_i + U_{i+1}) = -\Delta x^2 f(x_i)$$

Solve for U_i:

$$U_i = \frac{1}{2}\left(U_{i-1} + U_{i+1} + \Delta x^2 f(x_i)\right).$$

Note: With no heat source, $f(x) = 0$,
the temperature at each point is average of neighbors.

Suppose $U^{[k]}$ is a approximation to solution. Set

$$U_i^{[k+1]} = \frac{1}{2}\left(U_{i-1}^{[k]} + U_{i+1}^{[k]} + \Delta x^2 f(x_i)\right) \quad \text{for } i = 1,\ 2,\ \ldots,\ n.$$

Repeat for $k = 0,\ 1,\ 2,\ \ldots$ until convergence.

Can be shown to converge (eventually... very slow!)

Jacobi with OpenMP – coarse grain

General Approach:

- Fork threads only once at start of program.
- Each thread is responsible for some portion of the arrays, from `i=istart` to `i=iend`.
- Each iteration, must copy `u` to `uold`, update `u`, check for convergence.
- Convergence check requires coordination between threads to get global `dumax`.
- Print out final result after leaving parallel block

See code in the repository or the notes:
$CLASSHG/codes/openmp/jacobi2_omp.f90

Jacobi with MPI

Each process is responsible for some portion of the arrays,
from `i=istart` to `i=iend`.

No shared memory: each process only has part of array.

Updating formula:

```
u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i))
```

Need to exchange values at boundaries:

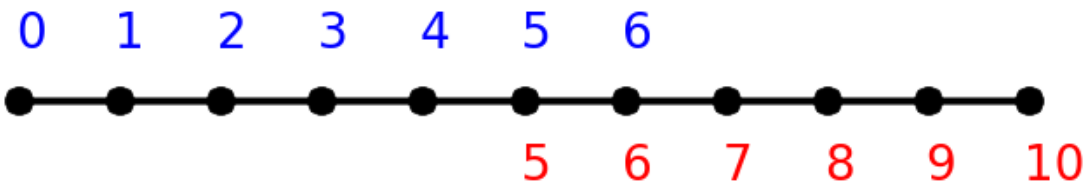
Updating at `i=istart` requires `uold(istart-1)`

Updating at `i=iend` requires `uold(istart+1)`

Example with $n = 9$ interior points (plus boundaries):

Process 0 has `istart = 1, iend = 5`

Process 1 has `istart = 6, iend = 9`



Jacobi with MPI

Other issues:

- Convergence check requires coordination between processes to get global `dumax`.
  Use `MPI_ALLREDUCE` so all process check same value.

- Part of final result must be printed by each process (into common file `heatsoln.txt`), in proper order.

See code in the repository or the notes:
$CLASSHG/codes/mpi/jacobi2_mpi.f90

Jacobi with MPI — splitting up arrays

```
real(kind = 8),dimension(:), allocatable :: f, u, uold
...
points_per_task = (n + ntasks - 1)/ntasks
call mpi_comm_rank(MPI_COMM_WORLD, me, ierr)
istart = me * points_per_task + 1
iend = min((me + 1)*points_per_task, n)

allocate(f(istart-1:iend+1), u(istart-1:iend+1), &
         uold(istart-1:iend+1))
```

Note that each process works on only a part of the array.

Distributed memory model, so no large shared array.

Includes “ghost cells” to store boundary values from neighboring processes.

Jacobi with MPI — Sending to neighbors

```
call mpi_comm_rank(MPI_COMM_WORLD, me, ierr)
...

do iter = 1, maxiter
    uold = u

    if (me > 0) then
        ! Send left endpoint value to "left"
        call mpi_isend(uold(istart), 1, MPI_DOUBLE_PRECI
            me - 1, 1, MPI_COMM_WORLD, req1, ierr)
      end if

    if (me < ntasks-1) then
        ! Send right endpoint value to "right"
        call mpi_isend(uold(iend), 1, MPI_DOUBLE_PRECISI
            me + 1, 2, MPI_COMM_WORLD, req2, ierr)
      end if
    end do
```

Note: Non-blocking `mpi_isend` is used,

Different tags (1 and 2) for left-going, right-going messages.

Jacobi with MPI — Receiving from neighbors

Note: `uold(istart)` from `me+1` goes into `uold(iend+1)`:
`uold(iend)` from `me-1` goes into `uold(istart-1)`:

```
do iter = 1, maxiter

    ! mpi_send's from previous slide

    if (me < ntasks-1) then
        ! Receive right endpoint value
        call mpi_recv(uold(iend+1), 1, MPI_DOUBLE_PRECIS
            me + 1, 1, MPI_COMM_WORLD, mpistatus, ierr)
      end if

    if (me > 0) then
        ! Receive left endpoint value
        call mpi_recv(uold(istart-1), 1, MPI_DOUBLE_PREC
            me - 1, 2, MPI_COMM_WORLD, mpistatus, ierr)
      end if


    ! Apply Jacobi iteration on my section of array
    do i = istart, iend
        u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i)
        dumax_task = max(dumax_task, abs(u(i) - uold(i))
        end do
    end do
```

Jacobi with MPI — Convergence test

```
do iter = 1, maxiter

    ! Send and receive boundary data (previous slides)

    dumax_task = 0.d0

    ! Jacobi update:
    do i = istart, iend
      u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i))
      dumax_task = max(dumax_task, abs(u(i) - uold(i)))
    end do

    ! Take global maximum of dumax values
    call mpi_allreduce(dumax_task, dumax_global, 1, &
         MPI_DOUBLE_PRECISION, &
         MPI_MAX, MPI_COMM_WORLD, ierr)

    if (dumax_global < tol) exit

enddo
```

Jacobi with MPI — Writing solution in order

Want to write table of values `x(i),u(i)` in `heatsoln.txt`.

Need them to be in proper order, so Process 0 must write to this file first, then Process 1, etc.

Jacobi with MPI — Writing solution in order

Want to write table of values `x(i),u(i)` in `heatsoln.txt`.

Need them to be in proper order, so Process 0 must write to this file first, then Process 1, etc.

Approach:

Each process `me` waits for a message from `me-1` indicating that it has finished writing its part. (Contents not important.)

Each process must open the file (without clobbering values already there), write to it, then close the file.

Jacobi with MPI — Writing solution in order

Want to write table of values `x(i),u(i)` in `heatsoln.txt`.

Need them to be in proper order, so Process 0 must write to this file first, then Process 1, etc.

Approach:

Each process `me` waits for a message from `me-1` indicating that it has finished writing its part. (Contents not important.)

Each process must open the file (without clobbering values already there), write to it, then close the file.

Assumes all processes share a file system!

On cluster or supercomputer, need to either:

send all results to single process for writing, or

write distributed files that may need to be combined later

(some visualization tools handle distributed data!)

Heat equation in 2 dimensions

One-dimensional equation generalizes to

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

on some domain in the x-y plane, with initial and boundary conditions.

We will only consider rectangle $0 \le x \le 1, \;\; 0 \le y \le 1$.

Heat equation in 2 dimensions

One-dimensional equation generalizes to

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

on some domain in the x-y plane, with initial and boundary conditions.

We will only consider rectangle $0 \le x \le 1, \;\; 0 \le y \le 1$.

Steady state problem (with $D = 1$):

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

Finite difference equations in 2D

$$\frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right) = -f(x_i, y_j).$$

On $n \times n$ grid ($\Delta x = \Delta y = 1/(n+1)$) this gives a linear system of n^2 equations in n^2 unknowns.

The above equation must be satisfied for $i = 1,\ 2,\ \ldots,\ n$ and $j = 1,\ 2,\ \ldots,\ n$.

Matrix is $n^2 \times n^2$,

e.g. on 100 by 100 grid, matrix is $10,000 \times 10,000$.

Contains $(10,000)^2 = 100,000,000$ elements.

Finite difference equations in 2D

$$\frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right) = -f(x_i, y_j).$$

On $n \times n$ grid ($\Delta x = \Delta y = 1/(n+1)$) this gives a linear system of n^2 equations in n^2 unknowns.

The above equation must be satisfied for $i = 1,\ 2,\ \ldots,\ n$ and $j = 1,\ 2,\ \ldots,\ n$.

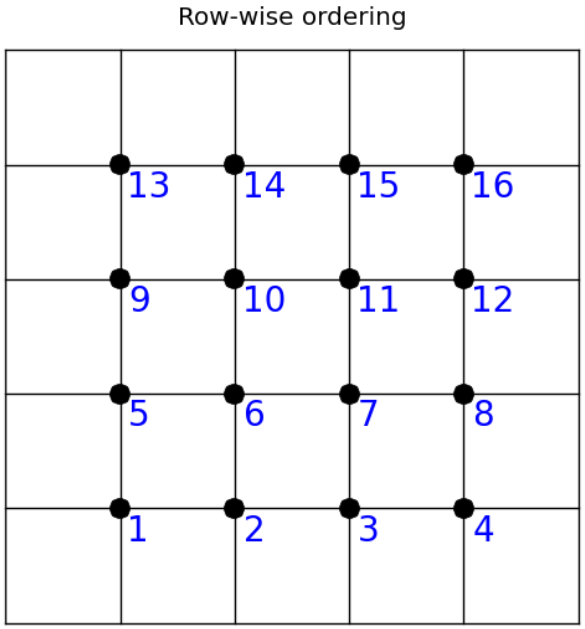
Matrix is $n^2 \times n^2$,

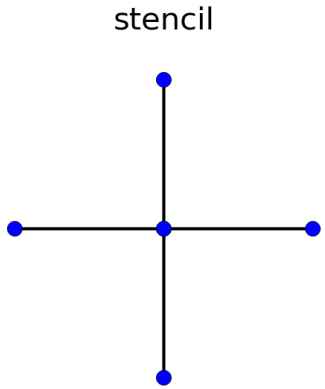
e.g. on 100 by 100 grid, matrix is $10,000 \times 10,000$.

Contains $(10,000)^2 = 100,000,000$ elements.

Matrix is sparse: each row has at most 5 nonzeros out of n^2 elements! But structure is no longer tridiagonal.

Finite difference equations in 2D

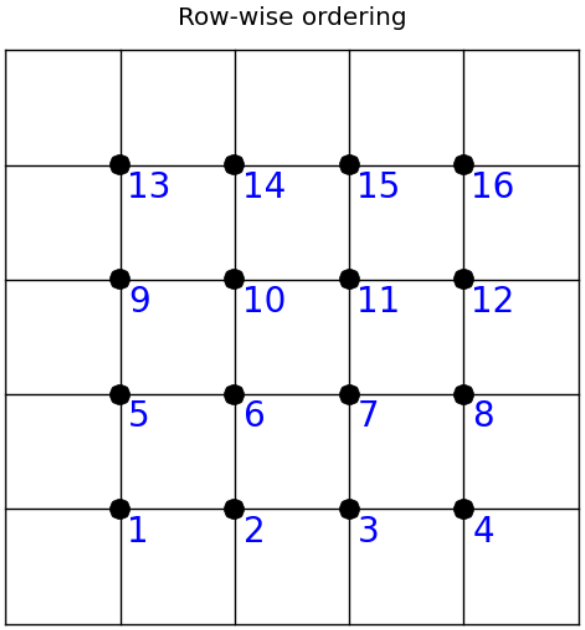


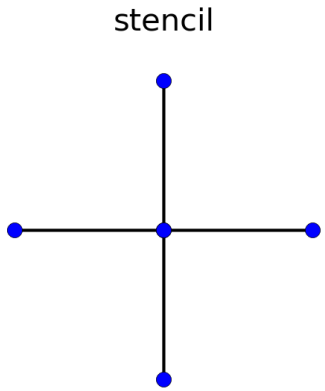


Matrix has block tridiagonal structure:

$$A = \frac{1}{h^2}\begin{bmatrix} T & I & & \\ I & T & I & \\ & I & T & I \\ & & I & T \end{bmatrix} \qquad T = \begin{bmatrix} -4 & 1 & & \\ 1 & -4 & 1 & \\ & 1 & -4 & 1 \\ & & 1 & -4 \end{bmatrix}$$

Jacobi in 2D

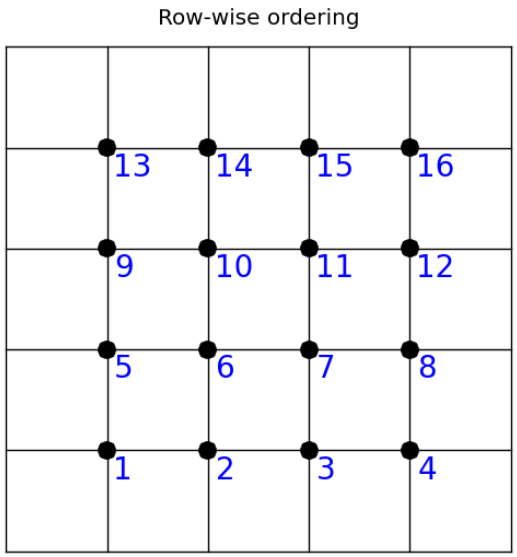


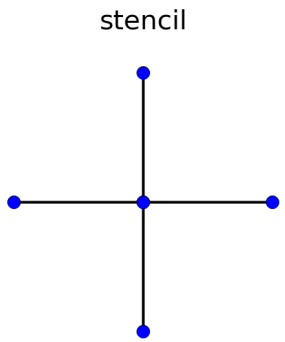


Updating point 7 for example (u_{32}):

$$U_{32}^{[k+1]} = \frac{1}{4}(U_{22}^{[k]} + U_{42}^{[k]} + U_{21}^{[k]} + U_{41}^{[k]} + h^2 f_{32})$$

Jacobi in 2D using MPI





With two processes: Could partition unknown into

Process 0 takes grid points 1–8

Process 1 takes grid points 9–16

Each time step:

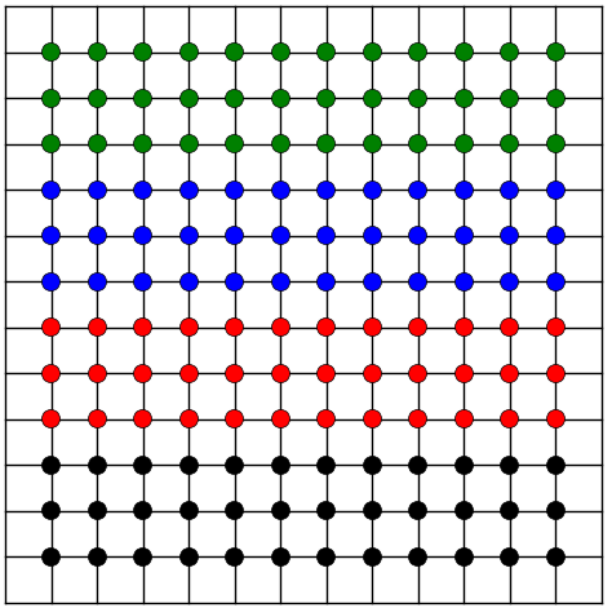
Process 0 sends top boundary (5–8) to Process 1,

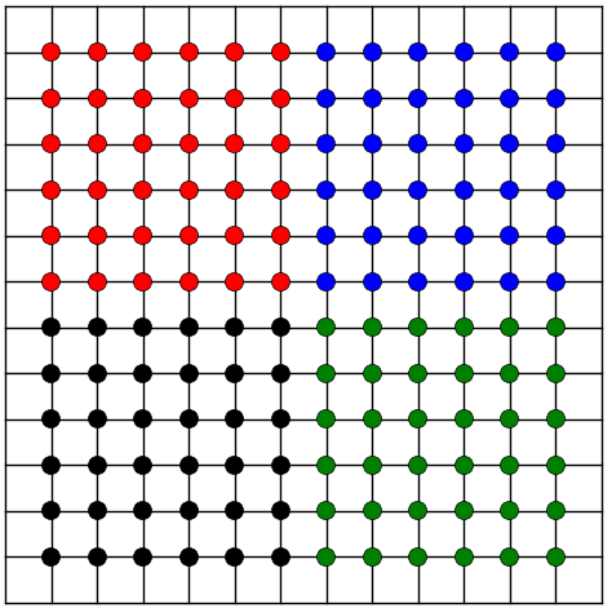
Process 1 sends bottom boundary (9–12) to Process 0.

Jacobi in 2D using MPI

With more grid points and processes...

Could partition several different ways, e.g. with 4 processes:

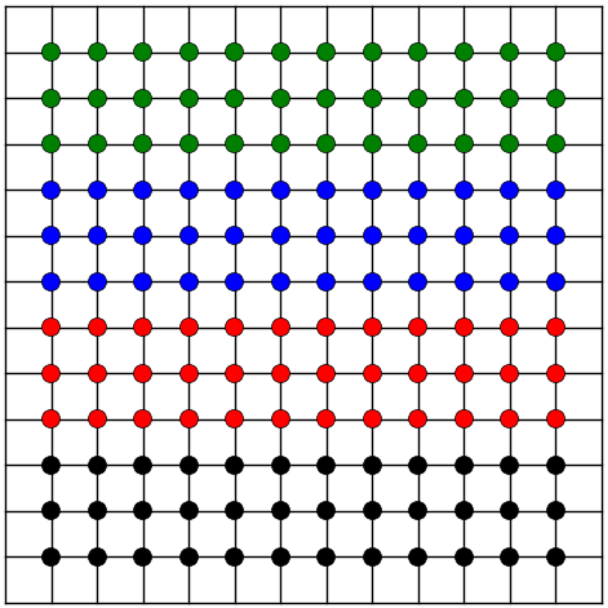


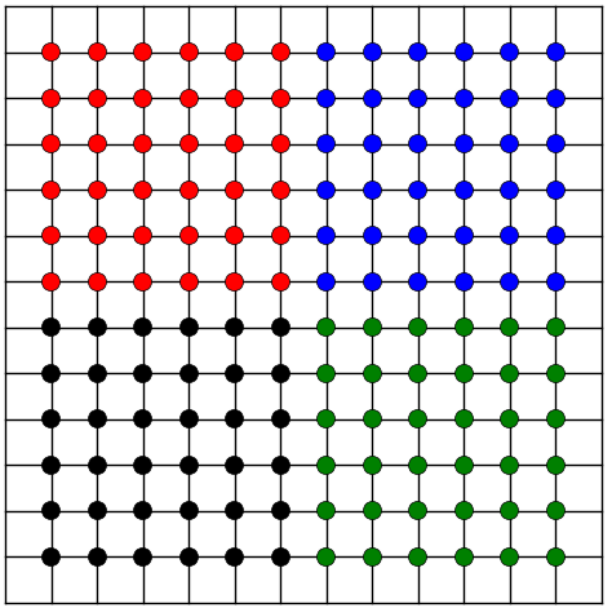


Jacobi in 2D using MPI

With more grid points and processes...

Could partition several different ways, e.g. with 4 processes:



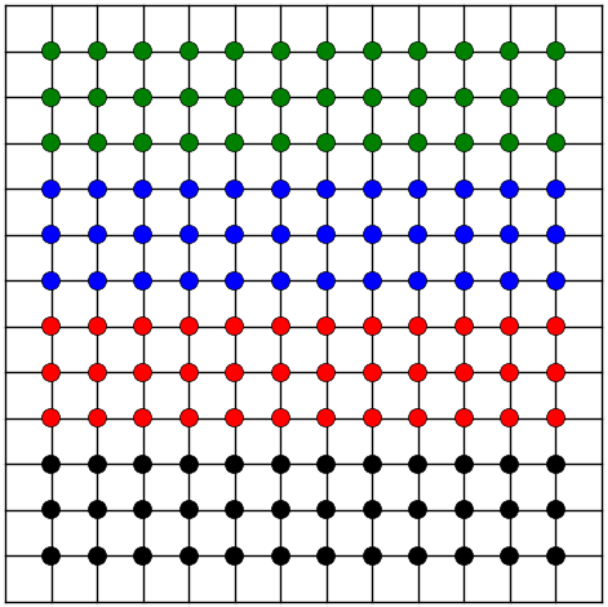


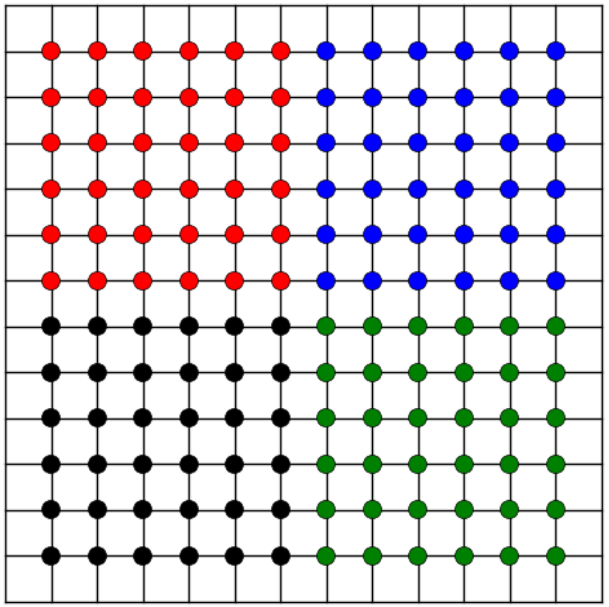
The partition on the right requires less communication.

With m^2 processes on grid with n^2 points,

$2m^2n$ boundary points on left, $2mn$ boundary points on right.

Jacobi in 2D using MPI





For partition on left: Natural to number processes 0,1,2,3 and pass boundary data from Process k to $k \pm 1$.

For $m \times m$ array of processors as on right: How do we figure out the neighboring process numbers?

Creating a communicator for Cartesian blocks

```
integer dims(2)
logical isperiodic(2), reorder

ndim = 2          ! 2d grid of processes
dims(1) = 4       ! for 4x6 grid of processes
dims(2) = 6
isperiodic(1) = .false.    ! periodic in x?
isperiodic(2) = .false.    ! periodic in y?
reorder = .true.           ! optimize ordering

call MPI_CART_CREATE(MPI_COMM_WORLD, ndim, &
     dims, isperiodic, reorder, comm2d, ierr)
```

Can find neighboring processes within comm2d using `MPI_CART_SHIFT`

Gauss-Seidel iteration in Fortran

```
do iter=1,maxiter

    dumax = 0.d0

    do i=1,n
      uold = u(i)
      u(i) = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
      dumax = max(dumax, abs(u(i)-uold))
      enddo

    ! check for convergence:
    if (dumax .lt. tol) exit

    enddo
```

Note: Now `u(i)` depends on value of `u(i-1)` that has already been updated for previous `i`.

Gauss-Seidel iteration in Fortran

```
do iter=1,maxiter

    dumax = 0.d0

    do i=1,n
      uold = u(i)
      u(i) = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
      dumax = max(dumax, abs(u(i)-uold))
      enddo

    ! check for convergence:
    if (dumax .lt. tol) exit

    enddo
```

Note: Now `u(i)` depends on value of `u(i-1)` that has already been updated for previous `i`.

Good news: This converges about twice as fast as Jacobi!

Gauss-Seidel iteration in Fortran

```
do iter=1,maxiter

    dumax = 0.d0

    do i=1,n
      uold = u(i)
      u(i) = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
      dumax = max(dumax, abs(u(i)-uold))
      enddo

    ! check for convergence:
    if (dumax .lt. tol) exit

    enddo
```

Note: Now `u(i)` depends on value of `u(i-1)` that has already been updated for previous `i`.

Good news: This converges about twice as fast as Jacobi!

But... loop carried dependence! Cannot parallelize so easily.

Red-black ordering

We are free to write equations of linear system in any order...

reordering rows of coefficient matrix, right hand side.

Can also number unknowns of linear system in any order...

reordering elements of solution vector.

Red-black ordering

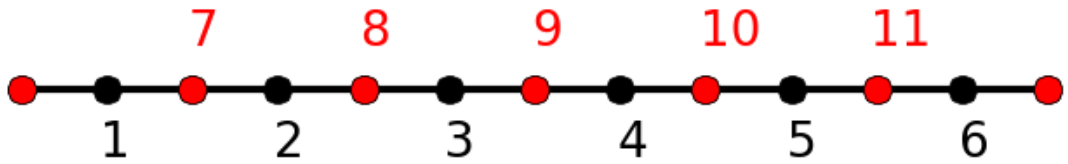
We are free to write equations of linear system in any order...
reordering rows of coefficient matrix, right hand side.

Can also number unknowns of linear system in any order...
reordering elements of solution vector.

Red-black ordering: Iterate through points with odd index first ($i = 1,\ 3,\ 5,\ \ldots$) and then even index points ($i = 2,\ 4,\ 6,\ \ldots$).

Then all black points can be updated in any order,
all red points can then be updated in any order.

Same asymptotic convergence rate as natural ordering.



Red-Black Gauss-Seidel

```
do iter=1,maxiter
    dumax = 0.d0

    ! UPDATE ODD INDEX POINTS:
    !$omp parallel do reduction(max : dumax) &
    !$omp private(uold)
    do i=1,n,2
        uold = u(i)
        u(i) = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
        dumax = max(dumax, abs(u(i)-uold))
        enddo

    ! UPDATE EVEN INDEX POINTS:
    !$omp parallel do reduction(max : dumax) &
    !$omp private(uold)
    do i=2,n,2
        uold = u(i)
        u(i) = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
        dumax = max(dumax, abs(u(i)-uold))
        enddo

    ! check for convergence:
    if (dumax .lt. tol) exit
    enddo
```

Gauss-Seidel method in 2D

If $\Delta x = \Delta y = h$:

$$\frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right) = -f(x_i, y_j).$$

Solve for $U_{i,j}$ and iterate:

$$u_{i,j}^{[k+1]} = \frac{1}{4}(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k]} - h^2 f_{i,j})$$

Again no need for matrix A.

Gauss-Seidel method in 2D

If $\Delta x = \Delta y = h$:

$$\frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right) = -f(x_i, y_j).$$

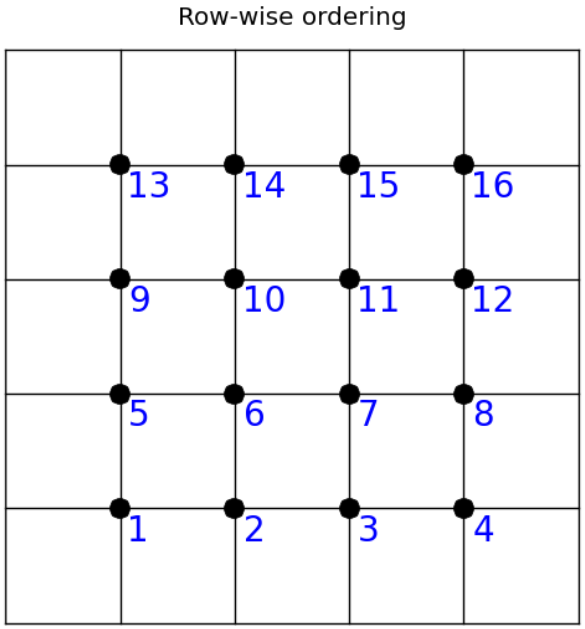
Solve for $U_{i,j}$ and iterate:

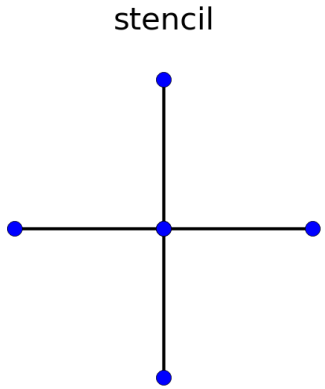
$$u_{i,j}^{[k+1]} = \frac{1}{4}(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k]} - h^2 f_{i,j})$$

Again no need for matrix A.

Note: Above indices for old and new values assumes we iterate in the natural row-wise order.

Gauss-Seidel in 2D



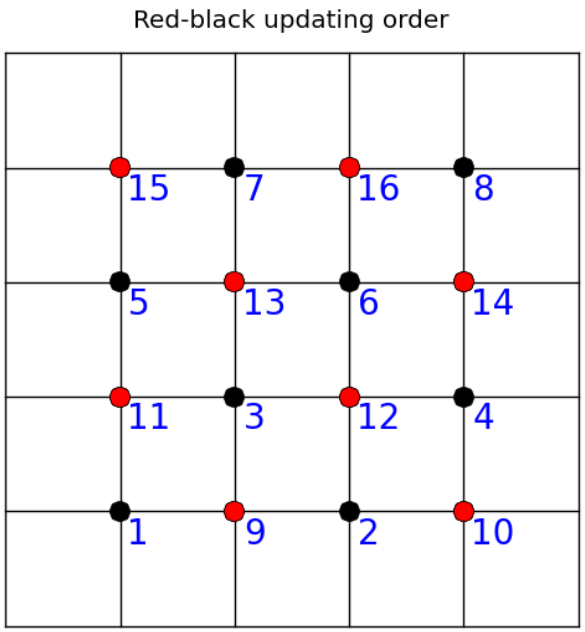


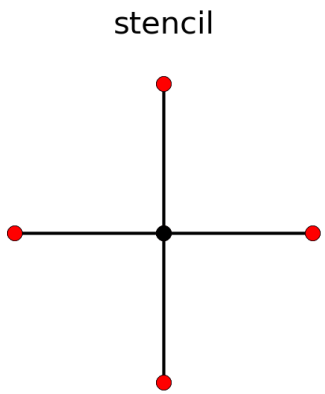
Updating point 7 for example (u_{32}):

Depends on new values at points 6 and 3, old values at points 7 and 10.

$$U_{32}^{[k+1]} = \frac{1}{4}(U_{22}^{[k+1]} + U_{42}^{[k]} + U_{21}^{[k+1]} + U_{41}^{[k]} + h^2 f_{32})$$

Red-black ordering in 2D





Again all black points can be updated in any order:

New value depends only on red neighbors.

Then all red points can be updated in any order:

New value depends only on black neighbors.

SOR method

Gauss-Seidel move solution in right direction but not far enough in general.

Iterates “relax” towards solution.

SOR method

Gauss-Seidel move solution in right direction but not far enough in general.

Iterates "relax" towards solution.

Successive Over-Relaxation (SOR):

Compute Gauss-Seidel approximation and then go further:

$$U_i^{GS} = \frac{1}{2}(U_{i-1}^{[k+1]} + U_{i+1}^{[k]} + \Delta x^2 f(x_i))$$
$$U_i^{[k+1]} = U_i^{[k]} + \omega(U_i^{GS} - U_i^{[k]})$$

where $1 < \omega < 2$.

SOR method

Gauss-Seidel move solution in right direction but not far enough in general.

Iterates "relax" towards solution.

Successive Over-Relaxation (SOR):

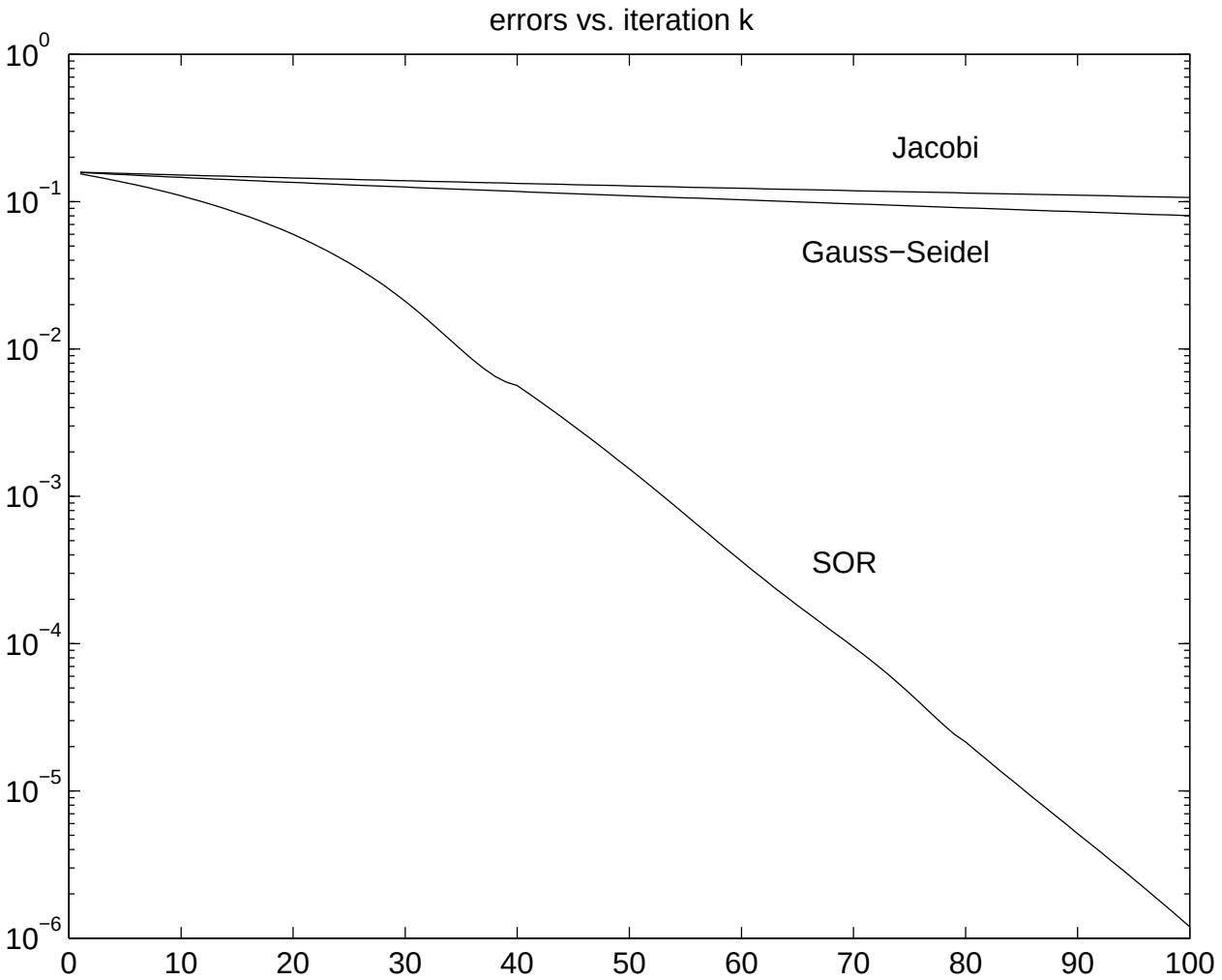
Compute Gauss-Seidel approximation and then go further:

$$U_i^{GS} = \frac{1}{2}(U_{i-1}^{[k+1]} + U_{i+1}^{[k]} + \Delta x^2 f(x_i))$$
$$U_i^{[k+1]} = U_i^{[k]} + \omega(U_i^{GS} - U_i^{[k]})$$

where $1 < \omega < 2$.

Optimal omega (For this problem): $\omega = 2 - 2\pi\Delta x$.

Convergence rates



Red-Black SOR in 1D

```
do iter=1,maxiter
    dumax = 0.d0

    ! UPDATE ODD INDEX POINTS:
    !$omp parallel do reduction(max : dumax) &
    !$omp private(uold, ugs)
    do i=1,n,2
        uold = u(i)
        ugs = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
        u(i) = uold + omega*(ugs-uold)
        dumax = max(dumax, abs(u(i)-uold))
        enddo

    ! UPDATE EVEN INDEX POINTS:
    !$omp parallel do reduction(max : dumax) &
    !$omp private(uold, ugs)
    do i=2,n,2
        uold = u(i)
        ugs = 0.5d0*(u(i-1) + u(i+1) + dx**2*f(i))
        u(i) = uold + omega*(ugs-uold)
        dumax = max(dumax, abs(u(i)-uold))
        enddo

    ! check for convergence...
```

Note that `uold, ugs` must be private!