HPSC – Lecture 20

Outline:

- Heat equation and discretization
- Iterative methods

Sample codes:

- /codes/openmp/jacobi1d_omp1.f90
- /codes/openmp/jacobi1d_omp2.f90

Heat Equation / Diffusion Equation

Partial differential equation (PDE) for $u(x, t)$
in one space dimension and time.

u represents temperature in a 1-dimensional metal rod.

Or concentration of a chemical diffusing in a tube of water.

Heat Equation / Diffusion Equation

Partial differential equation (PDE) for $u(x, t)$
in one space dimension and time.

u represents temperature in a 1-dimensional metal rod.

Or concentration of a chemical diffusing in a tube of water.

The PDE is

$$u_t(x, t) = Du_{xx}(x, t) + f(x, t)$$

where subscripts represent partial derivatives,

D = diffusion coefficient (assumed constant in space & time),

$f(x, t)$ = source term (heat or chemical being added/removed).

Heat Equation / Diffusion Equation

Partial differential equation (PDE) for $u(x, t)$
in one space dimension and time.

u represents temperature in a 1-dimensional metal rod.

Or concentration of a chemical diffusing in a tube of water.

The PDE is

$$u_t(x, t) = Du_{xx}(x, t) + f(x, t)$$

where subscripts represent partial derivatives,

D = diffusion coefficient (assumed constant in space & time),

$f(x, t)$ = source term (heat or chemical being added/removed).

Also need initial conditions $u(x, 0)$

and boundary conditions $u(x_1, t)$, $u(x_2, t)$.

Steady state diffusion

If $f(x, t) = f(x)$ does not depend on time and if the boundary conditions don't depend on time, then $u(x, t)$ will converge towards steady state distribution satisfying

$$0 = Du_{xx}(x) + f(x)$$

(by setting $u_t = 0$.)

This is now an ordinary differential equation (ODE) for $u(x)$.

Steady state diffusion

If $f(x, t) = f(x)$ does not depend on time and if the boundary conditions don't depend on time, then $u(x, t)$ will converge towards steady state distribution satisfying

$$0 = Du_{xx}(x) + f(x)$$

(by setting $u_t = 0$.)

This is now an ordinary differential equation (ODE) for $u(x)$.

We can solve this on an interval, say $0 \le x \le 1$ with

Boundary conditions:

$$u(0) = \alpha, \qquad u(1) = \beta.$$

Steady state diffusion

More generally: Take $D = 1$ or absorb in f,

$$u_{xx}(x) = -f(x) \qquad \text{for } 0 \le x \le 1,$$

Boundary conditions:

$$u(0) = \alpha, \qquad u(1) = \beta.$$

Can be solved exactly if we can integrate f twice and use boundary conditions to choose the two constants of integration.

Steady state diffusion

More generally: Take $D = 1$ or absorb in f,

$$u_{xx}(x) = -f(x) \qquad \text{for } 0 \le x \le 1,$$

Boundary conditions:

$$u(0) = \alpha, \qquad u(1) = \beta.$$

Can be solved exactly if we can integrate f twice and use boundary conditions to choose the two constants of integration.

Example: $\alpha = 20$, $\beta = 60$, $f(x) = 0$ (no heat source)

Solution: $u(x) = \alpha + x(\beta - \alpha) \qquad \Longrightarrow u''(x) = 0.$

No heat source $\Longrightarrow$ linear variation in steady state ($u_{xx} = 0$).

Steady state diffusion

More generally: Take $D = 1$ or absorb in f,

$$u_{xx}(x) = -f(x) \qquad \text{for } 0 \le x \le 1,$$

Boundary conditions:

$$u(0) = \alpha, \qquad u(1) = \beta.$$

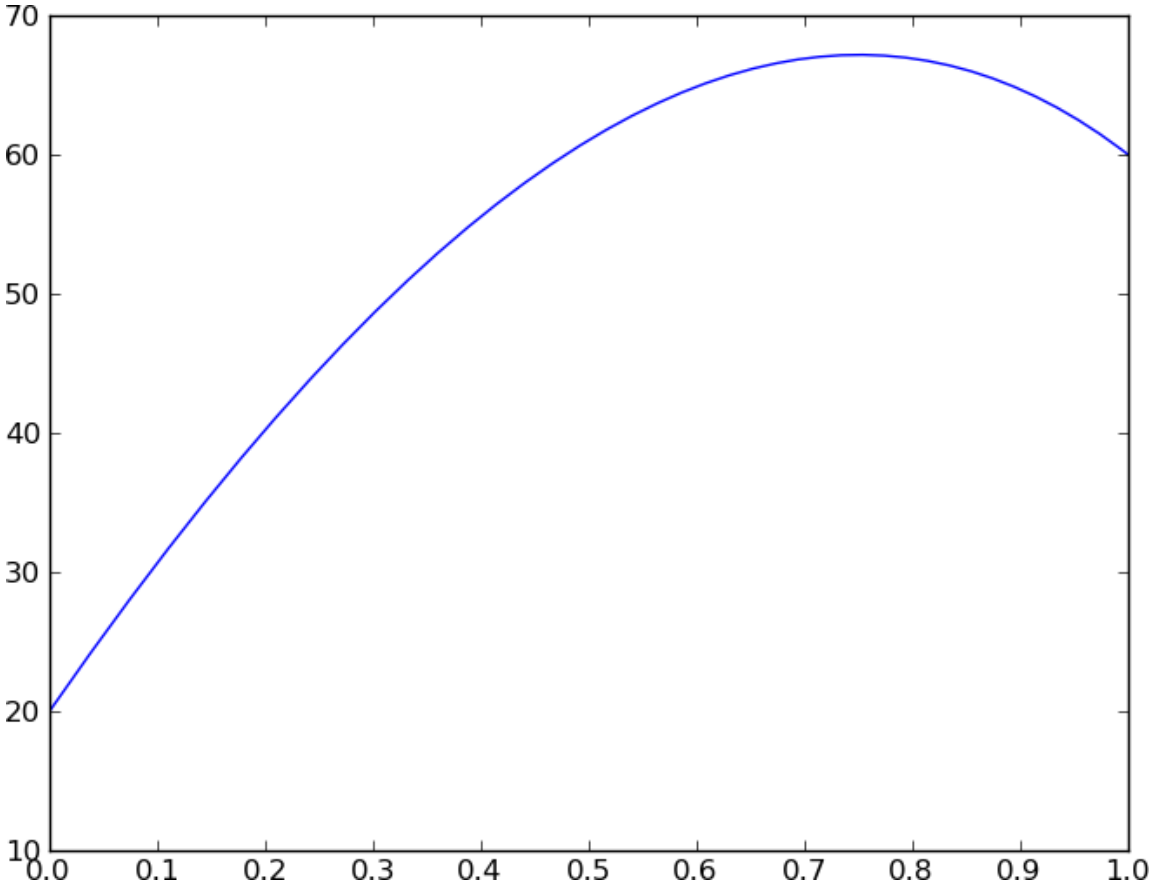
Can be solved exactly if we can integrate f twice and use boundary conditions to choose the two constants of integration.

More interesting example:

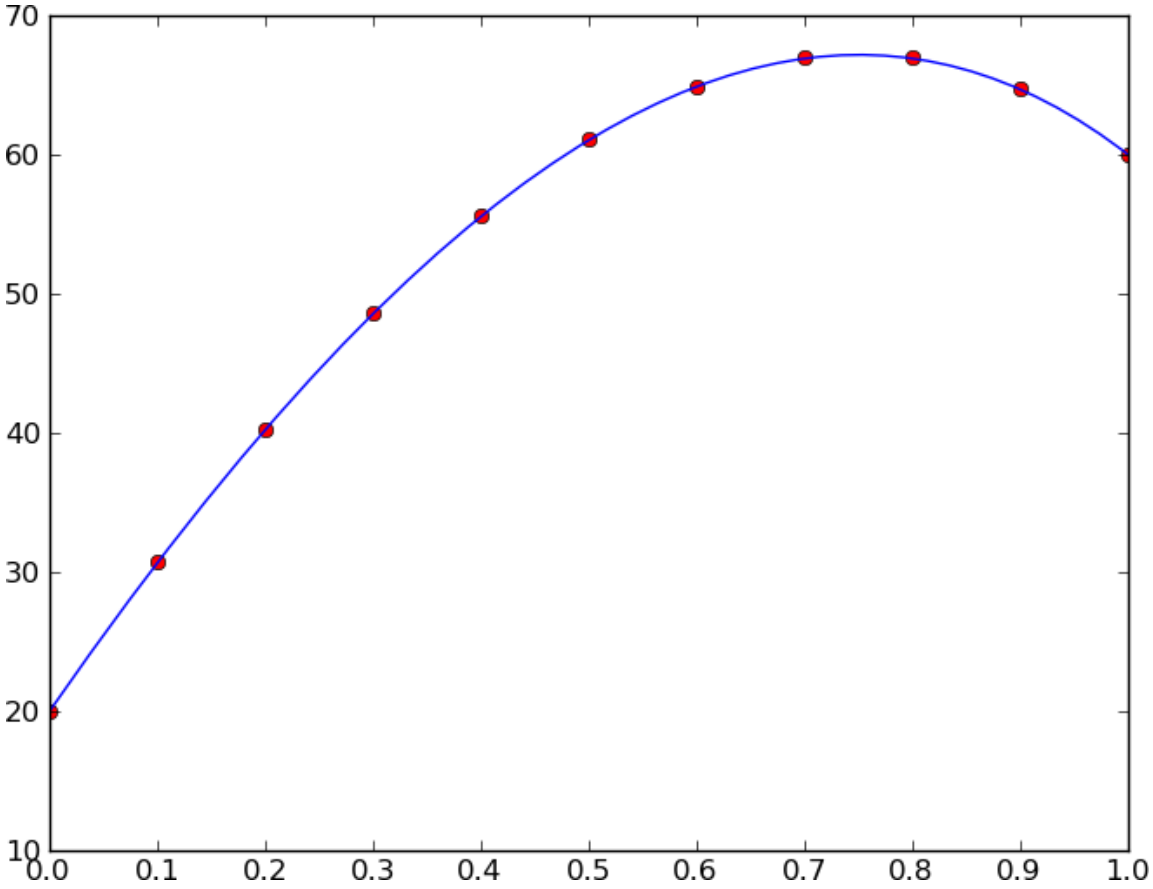
Example: $\alpha = 20, \quad \beta = 60, \quad f(x) = 100e^x,$

Solution: $u(x) = (100e - 60)x + 120 - 100e^x.$

Steady state diffusion



Steady state diffusion



For more complicated equations, numerical methods must generally be used, giving approximations at discrete points.

Finite difference method

Define grid points $x_i = i\Delta x$ in interval $0 \leq x \leq 1$, where

$$\Delta x = \frac{1}{n+1}$$

So $x_0 = 0$, $x_{n+1} = 1$, and the n grid points $x_1, x_2, \ldots, x_n$ are equally spaced inside the interval.

Finite difference method

Define grid points $x_i = i\Delta x$ in interval $0 \leq x \leq 1$, where

$$\Delta x = \frac{1}{n+1}$$

So $x_0 = 0$, $x_{n+1} = 1$, and the n grid points $x_1, x_2, \ldots, x_n$ are equally spaced inside the interval.

Let $U_i \approx u(x_i)$ denote approximate solution.

We know $U_0 = \alpha$ and $U_{n+1} = \beta$ from boundary conditions.

Finite difference method

Define grid points $x_i = i\Delta x$ in interval $0 \le x \le 1$, where

$$\Delta x = \frac{1}{n+1}$$

So $x_0 = 0$, $x_{n+1} = 1$, and the n grid points $x_1, x_2, \ldots, x_n$ are equally spaced inside the interval.

Let $U_i \approx u(x_i)$ denote approximate solution.

We know $U_0 = \alpha$ and $U_{n+1} = \beta$ from boundary conditions.

Idea: Replace differential equation for $u(x)$ by system of n algebraic equations for U_i values ($i = 1, 2, \ldots, n$).

Finite difference method

$$U_i \approx u(x_i)$$

$$u_x(x_{i+1/2}) \approx \frac{U_{i+1} - U_i}{\Delta x}$$

$$u_x(x_{i-1/2}) \approx \frac{U_i - U_{i-1}}{\Delta x}$$

Finite difference method

$$U_i \approx u(x_i)$$

$$u_x(x_{i+1/2}) \approx \frac{U_{i+1} - U_i}{\Delta x}$$

$$u_x(x_{i-1/2}) \approx \frac{U_i - U_{i-1}}{\Delta x}$$

So we can approximate second derivative at x_i by:

$$u_{xx}(x_i) \approx \frac{1}{\Delta x} \left(\frac{U_{i+1} - U_i}{\Delta x} - \frac{U_i - U_{i-1}}{\Delta x} \right)$$

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

Finite difference method

$$U_i \approx u(x_i)$$

$$u_x(x_{i+1/2}) \approx \frac{U_{i+1} - U_i}{\Delta x}$$

$$u_x(x_{i-1/2}) \approx \frac{U_i - U_{i-1}}{\Delta x}$$

So we can approximate second derivative at x_i by:

$$u_{xx}(x_i) \approx \frac{1}{\Delta x}\left(\frac{U_{i+1} - U_i}{\Delta x} - \frac{U_i - U_{i-1}}{\Delta x}\right)$$

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

This gives coupled system of n linear equations:

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

for $i = 1, 2, \ldots, n$. With $U_0 = \alpha$ and $U_{n+1} = \beta$.

Tridiagonal linear system

$$\alpha - 2U_1 + U_2 = -\Delta x^2 f(x_1) \qquad (i = 1)$$
$$U_1 - 2U_2 + U_3 = -\Delta x^2 f(x_2) \qquad (i = 2)$$
Etc.

For $n = 5$:

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix} = -\Delta x^2 \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{bmatrix} - \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ \beta \end{bmatrix}$$

Tridiagonal linear system

$$\alpha - 2U_1 + U_2 = -\Delta x^2 f(x_1) \qquad (i = 1)$$
$$U_1 - 2U_2 + U_3 = -\Delta x^2 f(x_2) \qquad (i = 2)$$
Etc.

For $n = 5$:

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \end{bmatrix} = -\Delta x^2 \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \end{bmatrix} - \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ \beta \end{bmatrix}$$

General $n \times n$ system requires $O(n^3)$ flops to solve.

Tridiagonal $n \times n$ system requires $O(n)$ flops to solve.

Could use LAPACK routine dgtsv.

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 $0 \leq x \leq 1,\ 0 \leq y \leq 1$.

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 $D = 1$):

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

This is a PDE in two spatial variables. (Poisson Problem)

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 $D = 1$):

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

This is a PDE in two spatial variables. (Poisson Problem)

Laplace's equation if $f(x, y) \equiv 0$.

$\nabla^2 = (\partial_x^2 + \partial_y^2)$ is the Laplacian operator.

Finite difference equations for 2D Poisson problem

Let $U_{ij} \approx u(x_i, y_j)$.

Replace differential equation

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

by algebraic equations

$$\frac{1}{\Delta x^2}\left(U_{i-1,j} - 2U_{i,j} + U_{i+1,j}\right) + \frac{1}{\Delta y^2}\left(U_{i,j-1} - 2U_{i,j} + U_{i,j+1}\right) = -f(x_i, y_j)$$

Finite difference equations for 2D Poisson problem

Let $U_{ij} \approx u(x_i, y_j)$.

Replace differential equation

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

by algebraic equations

$$\frac{1}{\Delta x^2}(U_{i-1,j} - 2U_{i,j} + U_{i+1,j}) + \frac{1}{\Delta y^2}(U_{i,j-1} - 2U_{i,j} + U_{i,j+1}) = -f(x_i, y_j)$$

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

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

Finite difference equations for 2D Poisson problem

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

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 for 2D Poisson problem

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

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 $j = 1, 2, \ldots, n$.

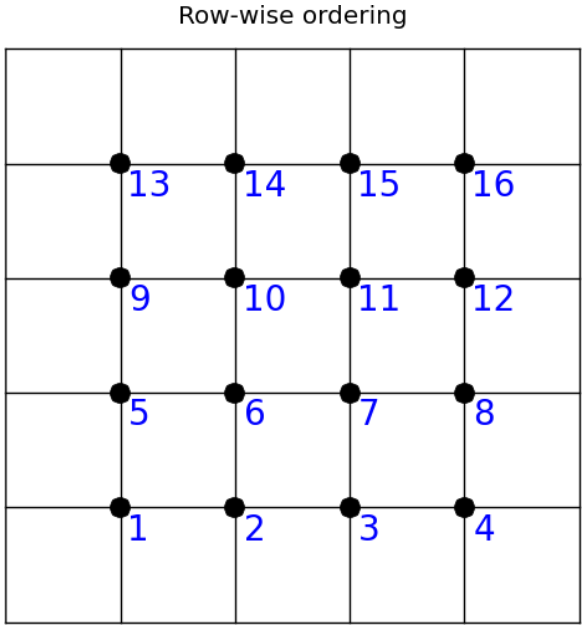
Matrix is $n^2 \times n^2$,

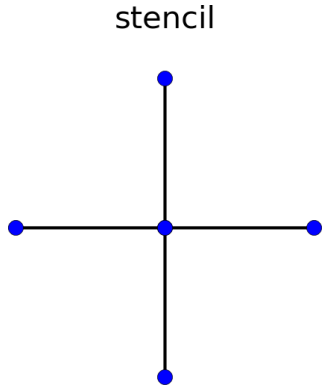
e.g. on 100 by 100 grid, matrix is $10,000 \times 10,000$.

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 for 2D Poisson problem





Matrix has block tridiagonal structure:

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

Iterative methods

Back to one space dimension first...

Coupled system of n linear equations:

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

for $i = 1, 2, \ldots, n$. With $U_0 = \alpha$ and $U_{n+1} = \beta$.

Iterative method starts with initial guess $U^{[0]}$ to solution and then improves $U^{[k]}$ to get $U^{[k+1]}$ for $k = 0, 1, \ldots$.

Note: Generally does not involve modifying matrix A.

Do not have to store matrix A at all, only know about stencil.

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.

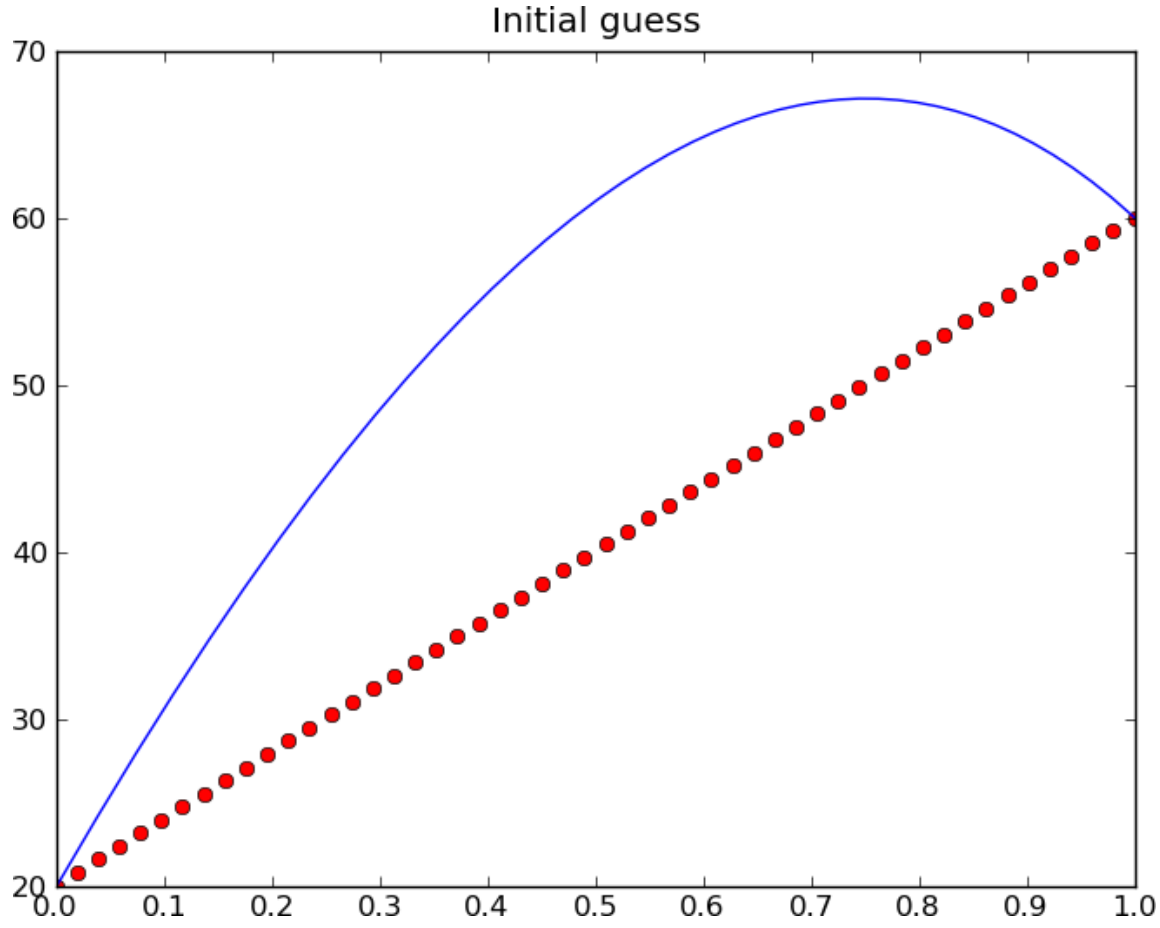
Suppose $U^{[k]}$ is a approximation to solution. Set

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

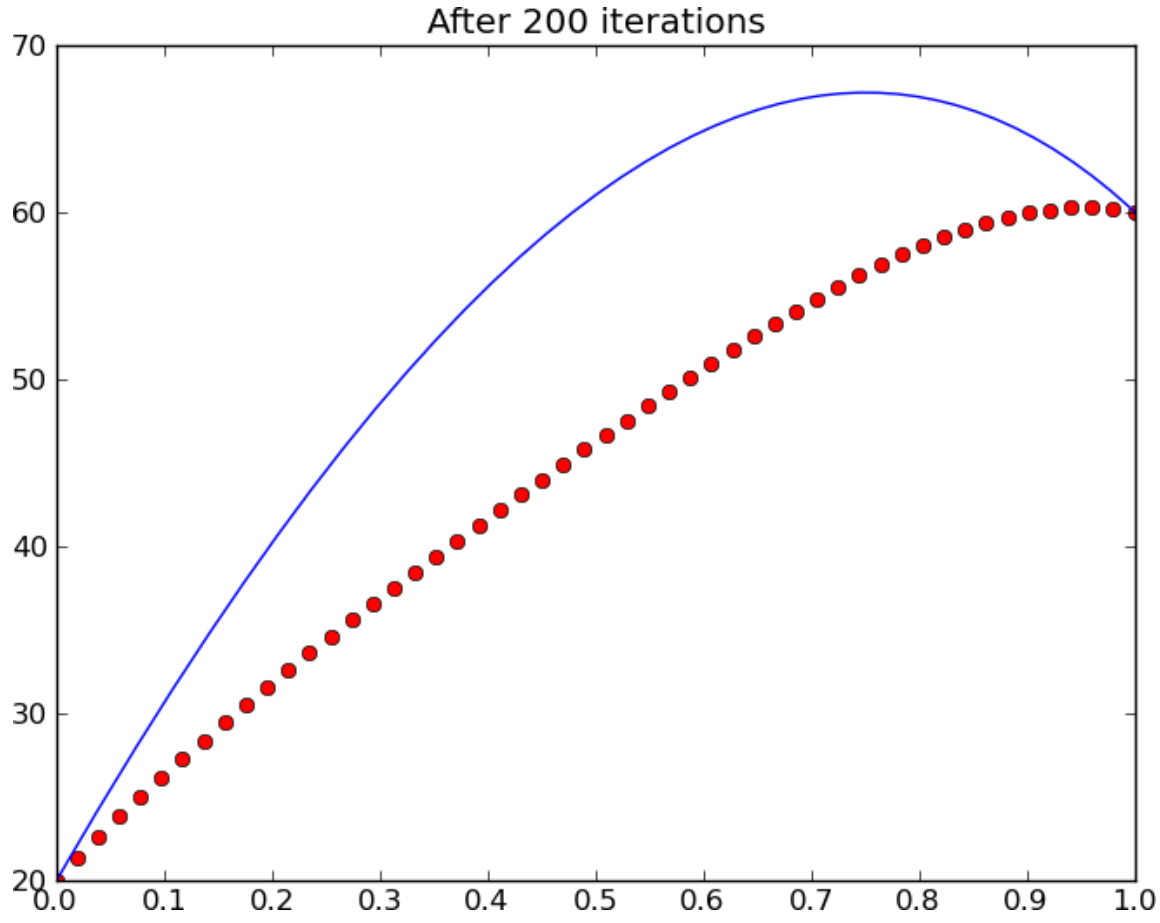
Repeat for $k = 0, 1, 2, \ldots$ until convergence.

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

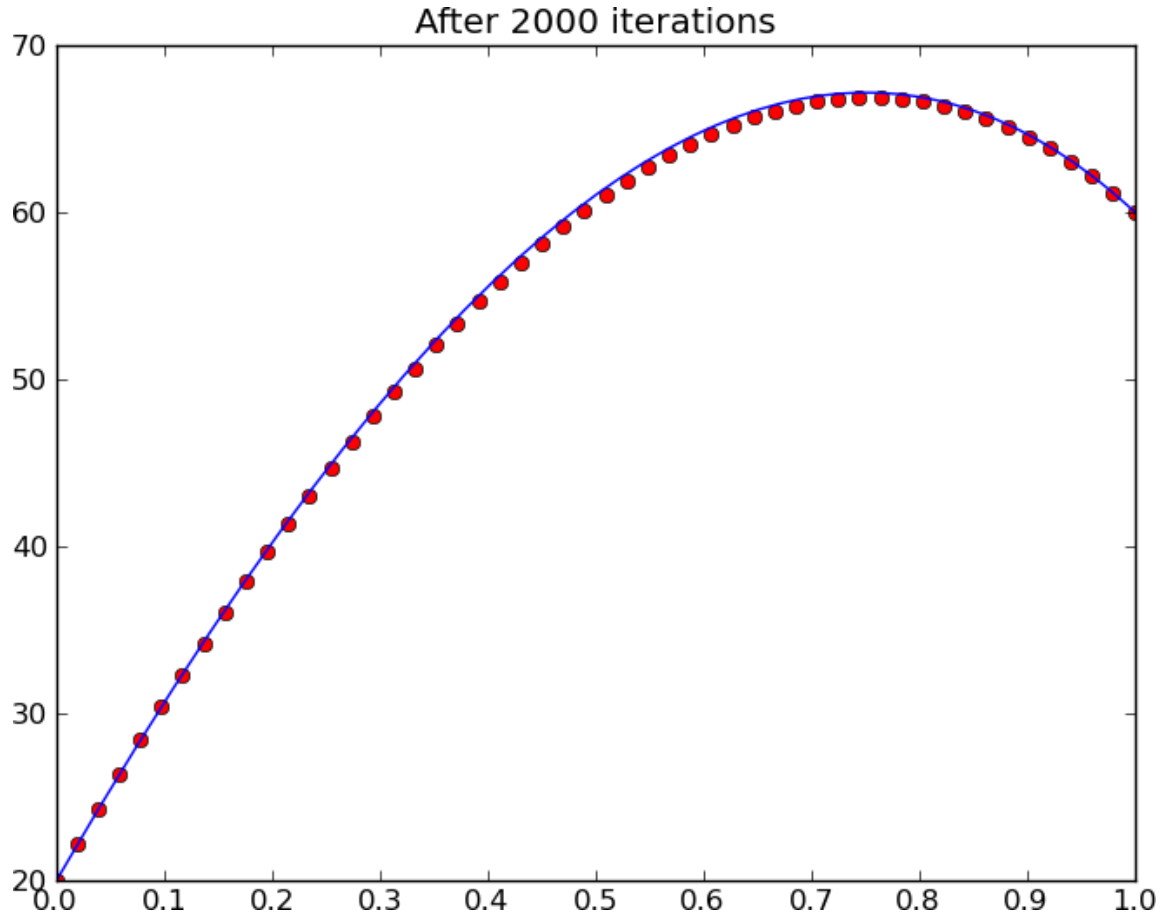
Slow convergence of Jacobi



Slow convergence of Jacobi



Slow convergence of Jacobi



Iterative methods

Jacobi iteration is about the worst possible iterative method.

But it's very simple, and useful as a test for parallelization.

Better iterative methods:

- Gauss-Seidel
- Successive Over-Relaxation (SOR)
- Conjugate gradients
- Preconditioned conjugate gradients
- Multigrid

Iterative methods – initialization

```
! allocate storage for boundary points too:
allocate(x(0:n+1), u(0:n+1), f(0:n+1))

dx = 1.d0 / (n+1.d0)

!$omp parallel do
do i=0,n+1
    ! grid points:
    x(i) = i*dx
    ! source term:
    f(i) = 100.*exp(x(i))
    ! initial guess (linear function):
    u(i) = alpha + x(i)*(beta-alpha)
    enddo
```

Jacobi iteration in Fortran

```
uold = u  ! starting values before updating

do iter=1,maxiter

    dumax = 0.d0

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

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

    uold = u  ! for next iteration
    enddo
```

Note: we must use old value at $i-1$ for Jacobi.

Otherwise we get the Gauss-Seidel method.

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

Jacobi iteration in Fortran

```
uold = u  ! starting values before updating

do iter=1,maxiter

    dumax = 0.d0

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

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

    uold = u  ! for next iteration
    enddo
```

Note: we must use old value at $i-1$ for Jacobi.

Otherwise we get the Gauss-Seidel method.

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

This actually converges faster!

Jacobi with OpenMP `parallel do` (fine grain)

See: /codes/openmp/jacobi1d_omp1.f90

```
uold = u  ! starting values before updating

do iter=1,maxiter

     dumax = 0.d0

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

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

     !$omp parallel do
     do i=1,n
         uold(i) = u(i) ! for next iteration
         enddo
     enddo
```

Note: Forking threads twice each iteration.

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:
codes/openmp/jacobi1d_omp2.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:

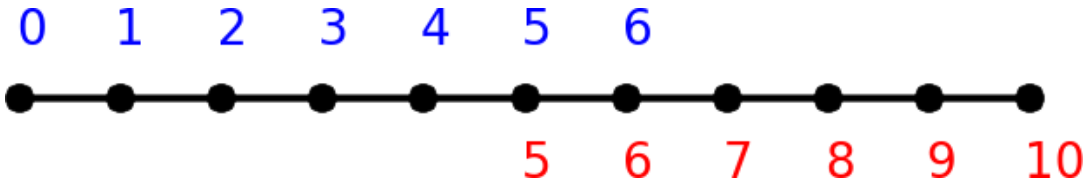
Updating at `i=istart` requires `uold(istart-1)`

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 — 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

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:
/codes/mpi/jacobi1d_mpi.f90

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.)

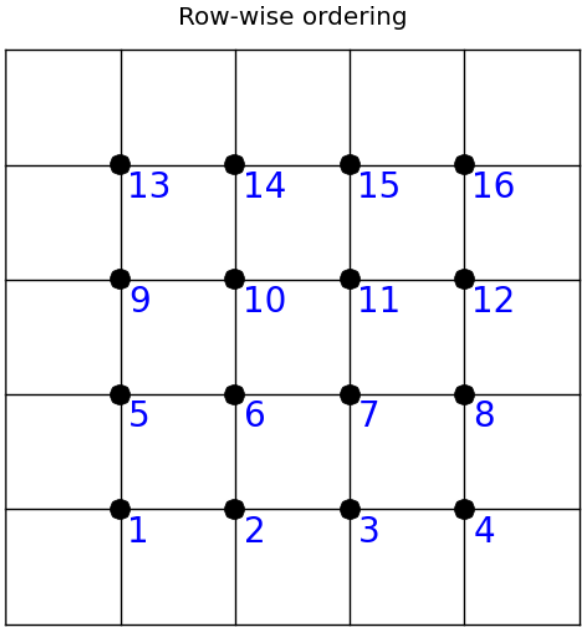
Each process must open the file (without clobbering values already there), write to it, then close the file.

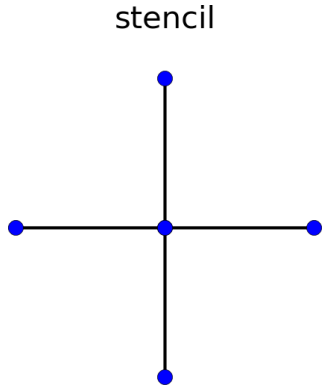
Assumes all processes share a file system!

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!)

Jacobi in 2D

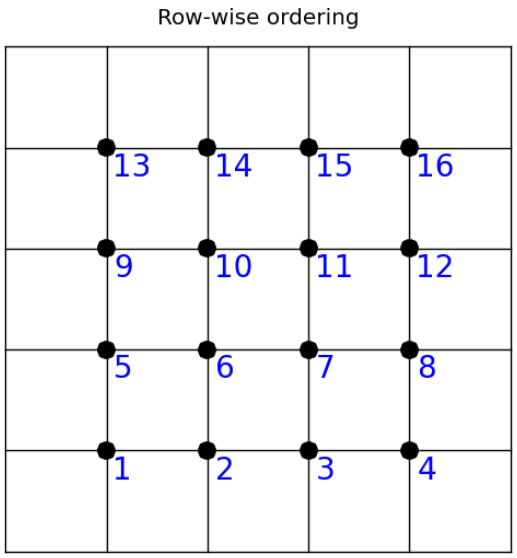


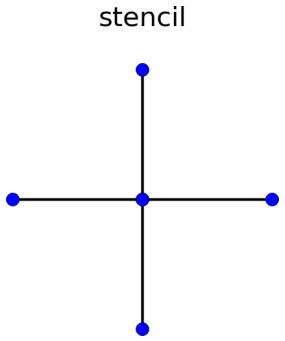


Updating point 7 for example (u_{32}):

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

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:

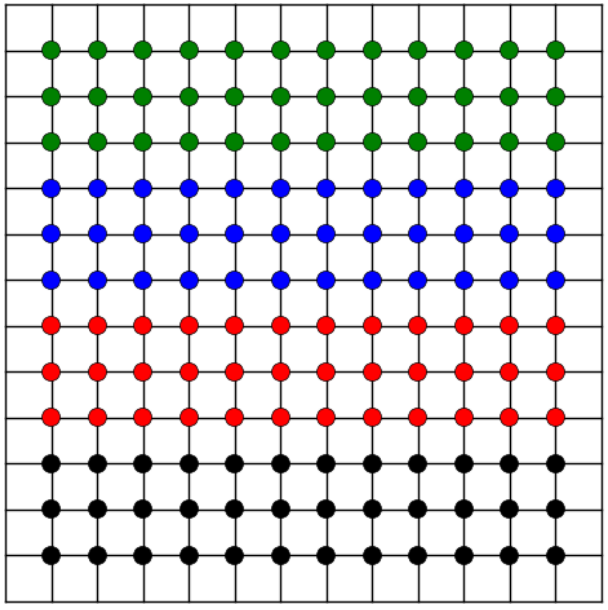
Process 0 sends top boundary (5–8) to Process 1,

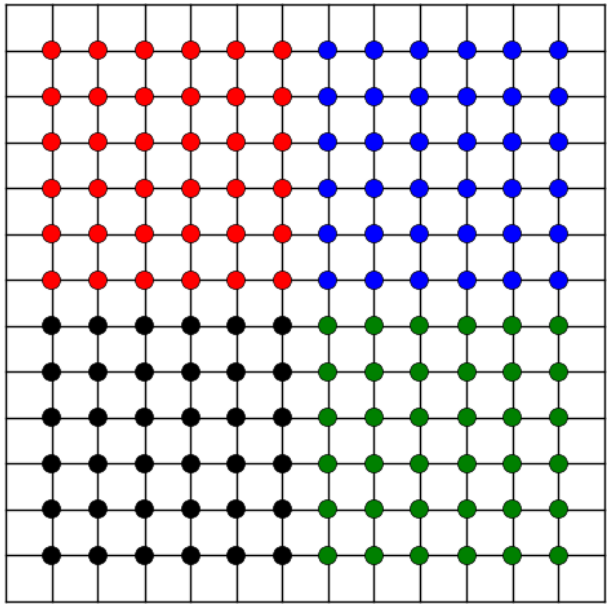
Process 1 sends bottom boundary (9–12) to Process 0.

Jacobi in 2D using MPI

With more grid points and processes...

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

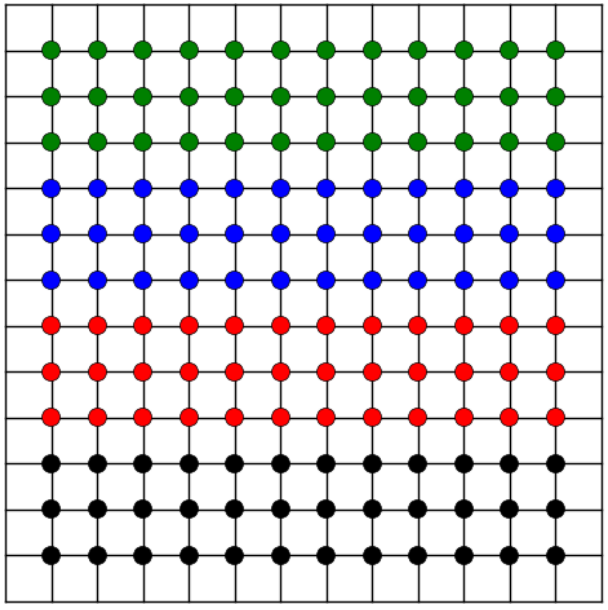


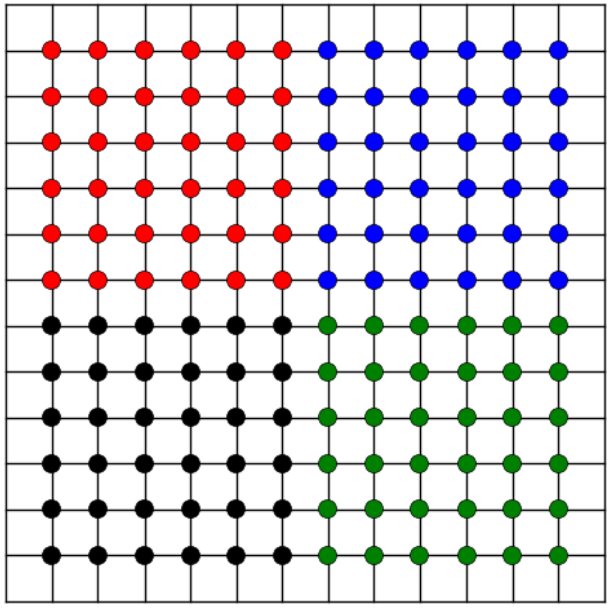


Jacobi in 2D using MPI

With more grid points and processes...

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



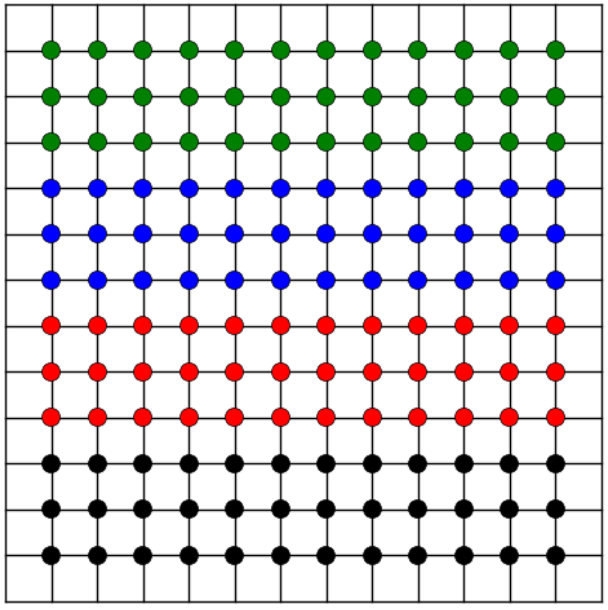


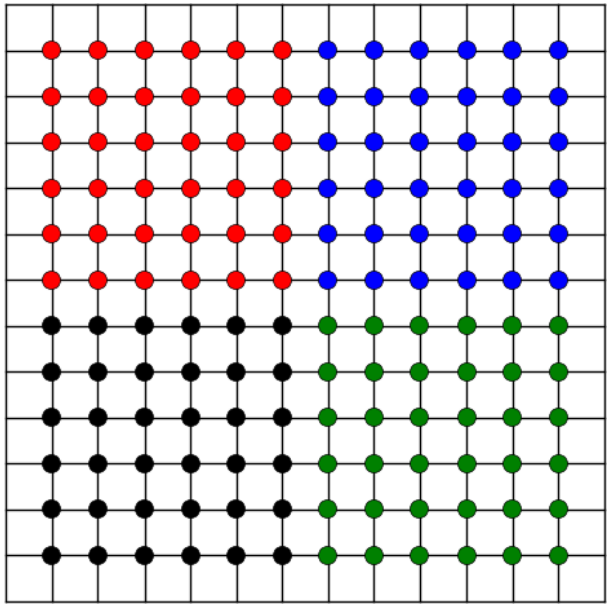
The partition on the right requires less communication.

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

$2(m^2 - 1)n$ boundary points on left,
$4(m - 1)n$ 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)
```

Create communicator comm2d. See also:

MPI_CART_CREATE, MPI_CART_SHIFT, MPI_CART_COORDS.