AMath 483/583 — Lecture 24

Outline:

- Heat equation and discretization
- OpenMP and MPI for iterative methods
- Jacobi, Gauss-Seidel, SOR

Notes and Sample codes:

- Class notes: Linear algebra software
- $UWHPSC/codes/openmp/jacobi1d_omp1.f90
- $UWHPSC/codes/openmp/jacobi1d_omp2.f90
- $UWHPSC/codes/mpi/jacobi1d_mpi.f90

Notes:

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 $0 \leq x \leq 1$ with

Boundary conditions:

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

Notes:

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 $U_{n+1} = \beta$.

Notes:

Iterative methods

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.

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

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

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

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

Jacobi 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 – 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:
    $UWHPSC/codes/openmp/jacobi1d_omp2.f90

Notes:

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))`
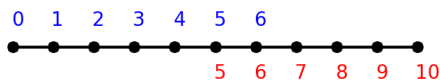Need to exchange values at boundaries:
    Updating at `i=istart` requires `uold(istart-1)`
    Updating at `i=iend` requires `uold(istart+1)`

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

Process 0 has `istart = 1, iend = 5`
Process 1 has `istart = 6, iend = 9`

```
        0   1   2   3   4   5   6
        •—•—•—•—•—•—•—•—•—•—•
                    5   6   7   8   9   10
```

Notes:

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.

Notes:

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)`:

```fortran
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
```

Notes:

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:
$UWHPSC/codes/mpi/jacobi1d_mpi.f90

Notes:

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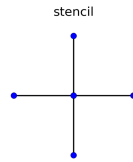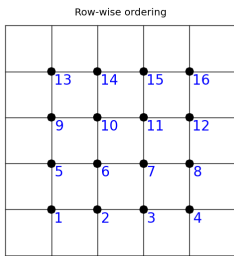
Each process must open the file (without clobbering values already there), write to it, then close the file.

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!)
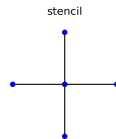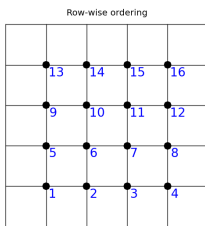
Notes:

Jacobi in 2D

Row-wise ordering

stencil

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

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

Notes:

Jacobi in 2D using MPI

Row-wise ordering

stencil

With two processes: Could partition unknown into
   Process 0 takes grid points 1–8
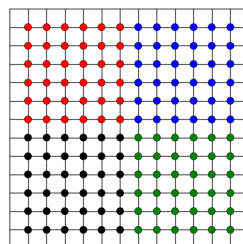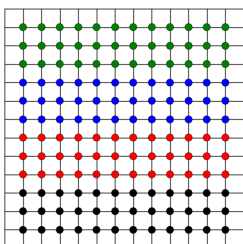   Process 1 takes grid points 9–16

Each time step:
   Process 0 sends top boundary (5–8) to Process 1,
   Process 1 sends bottom boundary (9–12) to Process 0.

Notes:

Jacobi in 2D using MPI

With more grid points and processes...
   Could partition several different ways, e.g. with 4 processes:
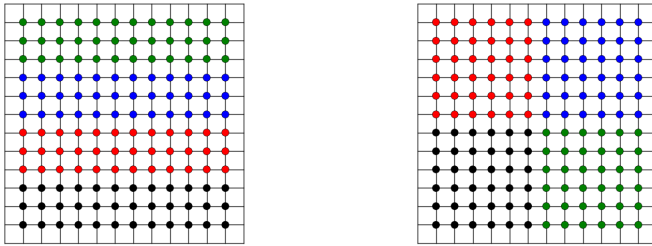
The partition on the right requires less communication.

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.

Notes:

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?

Notes:

Creating a communicator for Cartesian blocks

```fortran
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.

Notes:

Gauss-Seidel iteration in Fortran

```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.

Notes:

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.

Notes:

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
```

Notes:

Gauss-Seidel method in 2D

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

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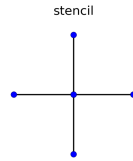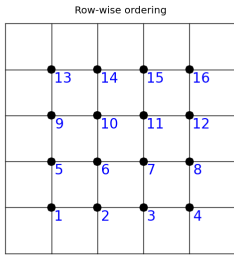
Solve for $U_{i,j}$ and iterate:

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

Again no need for matrix A.

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

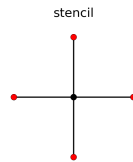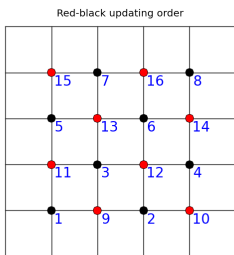## Notes:

Gauss-Seidel in 2D

Row-wise ordering

stencil

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

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

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

Notes:

Red-black ordering in 2D

Red-black updating order

stencil

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.

Notes:

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:

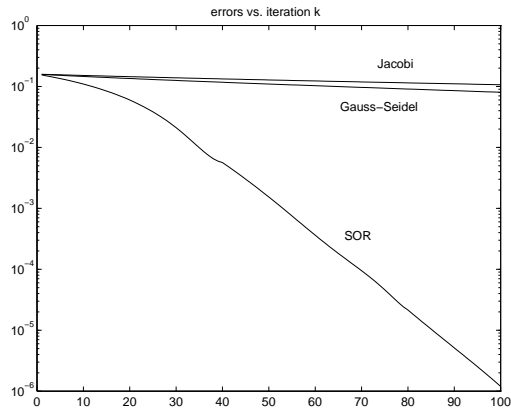$$U_i^* = \frac{1}{2}(U_{i-1}^{[k+1]} + U_{i+1}^{[k]} + \Delta x^2 f(x_i))$$
$$U_i^{[k+1]} = U_i^{[k]} + \omega(U_i^* - U_i^{[k]})$$

where $1 < \omega < 2$.

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

Notes:

Convergence rates

errors vs. iteration k

Notes:

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

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

   ! check for convergence...
```

Note that `uold, ustar` must be private!

Notes:

Announcements

Homework 6 is in the notes and due next **Friday**.

Quizzes for this week's lectures due next **Wednesday**.

Office hours today 9:30 – 10:20.

Next week:

Monday: no class

Wednesday: Guest lecture —

Brad Chamberlain, Cray

Chapel: A Next-Generation Partitioned
Global Address Space (PGAS) Language

Notes: