

Phys 7411:
The 2-D Poisson Problem
Reading (Dongarra: Chapter 16)

Juana Moreno - moreno@lsu.edu
and
Karen Tomko – ktomko@osc.edu
Spring 2009

The 2-D Poisson Problem

- Elliptic partial differential equation:

$$\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} = f(x, y) \quad \text{in the interior}$$

$$u(x, y) = g(x, y) \quad \text{on the boundary}$$

- It describes fluid flow, electrostatics, equilibrium heat flow.
- To compute solution:

- discrete mesh: (x_i, y_j) ; $x_{i+1} - x_i = h, y_{j+1} - y_j = h$

- centered-difference approximation:

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j))}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1}))}{h^2} = f(x_i, y_j)$$

- Jacobi method

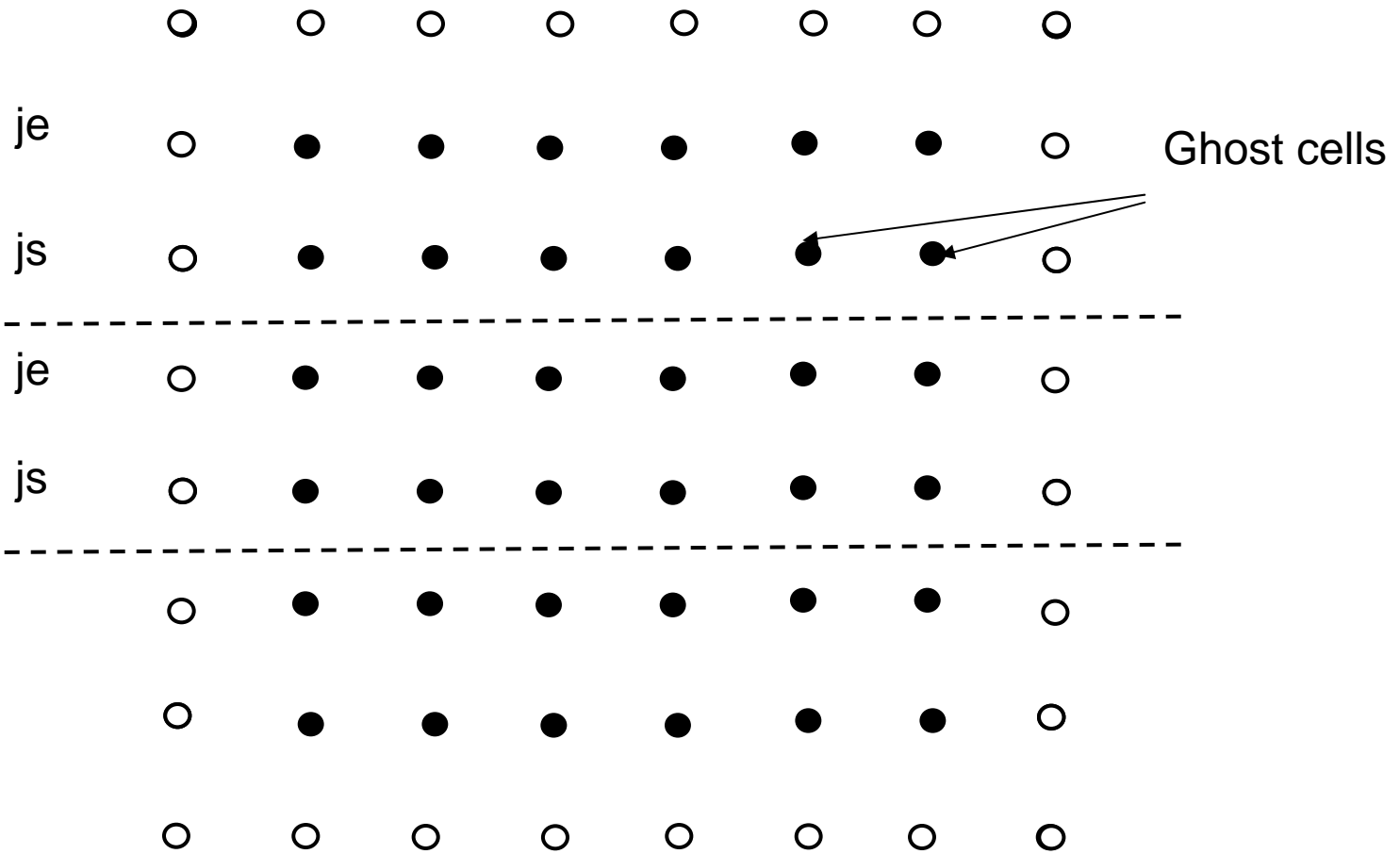
$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - h^2 f_{i,j})$$

The 2-D Poisson Problem

Jacobi algorithm:

```
real u(0:n,0:n), unew(0:n,0:n), f(1:n,1:n), h
!Initialize f, u(0,*), u(n,*), u(*,0), u(*,n) with g
h=1.0/n
do k=1,maxiter
  do j=1,n-1
    do i=1,n-1
      unew(i,j)=0.25*(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1)- h*h*f(i,j))
    enddo
  enddo
! Check for convergence
! Update u for the next iteration
  u=unew
enddo
```

Parallel implementation: MPI



```

use mpi
real u(0:n,js-1:je+1), unew(0:n,js-1:je+1), f(1:n-1,js:je), h
Integer nbr_down, nbr_up, status(MPI_STATUS_SIZE), ierr
!Initialize f, u(0,*), u(n,*), u(*,0), u(*,n) with g
h=1.0/n
do k=1,maxiter
! Send down
Call MPI_Sendrecv (u(1,js),n-1,mpi_real, nbr_down,k,&
                    u(1,je+1),n-1, mpi_real, nbr_up,k,&
                    MPI_COMM_WORLD, status, ierr)
! Send up
Call MPI_Sendrecv (u(1,je),n-1,mpi_real, nbr_up,k+1,&
                    u(1,js-1),n-1, mpi_real, nbr_down,k+1,&
                    MPI_COMM_WORLD, status, ierr)
do j=js,je
  do i=1,n-1
    unew(i,j)=0.25*(u(i+1,j)+ u(i-1,j)+ u(i,j+1)+ u(i,j-1)- h*h*f(i,j))
  enddo
enddo
! Check for convergence
! Update u for the next iteration
  u=unew
enddo

```

Parallel implementation: OpenMP

```
real u(0:n,0:n), unew(0:n,0:n), f(1:n-1,1:n-1), h
!Initialize f, u(0,*), u(n,*), u(*,0), u(*,n) with g
h=1.0/n
do k=1,maxiter
! $omp parallel
! $omp do
do j=1,n-1
    do i=1,n-1
        unew(i,j)=0.25*(u(i+1,j)+ u(i-1,j)+ u(i,j+1)+ u(i,j-1)- h*h*f(i,j))
    enddo
enddo
! $omp enddo
! Check for convergence
! Update u for the next iteration
    u=unew
! $omp end parallel
enddo
```

Global operations: Reduction

Single processor

```
real u(0:n,0:n), unew(0:n,0:n), twonorm
.....
twonorm=0.0
do j=1,n-1
  do i=1,n-1
    twonorm=twonorm+ (unew(i,j)-u(i,j))**2
  enddo
enddo
twonorm=sqrt(twonorm)
If (twonorm.le.tol)
!... Declare convergence
```

MPI

```
use mpi
real u(0:n,js-1:je+1),unew(0:n, js-1:e+1), &
      twonorm

integer ierr
.....
twonorm_local=0.0
do j=js,je
  do i=1,n-1
    twonorm_local=twonorm_local+&
      (unew(i,j)-u(i,j))**2
  enddo
dnddo
Call MPI_Allreduce(twonorm_local,twonorm,&
1,MPI_REAL,MPI_SUM,MPI_COMM_WORLD,
& ierr)
twonorm=sqrt(twonorm)
If (twonorm.le.tol)
!... Declare convergence
```

Global operations: Reduction

Single processor

```
real u(0:n,0:n), unew(0:n,0:n), twonorm
.....
twonorm=0.0
do j=1,n-1
  do i=1,n-1
    twonorm=twonorm+ (unew(i,j)-u(i,j))**2
  enddo
enddo
twonorm=sqrt(twonorm)
If (twonorm.le.tol)
!... Declare convergence
```

OpenMP

```
real u(0:n,0:n), unew(0:n,0:n), twonorm
.....
twonorm=0.0
!$omp parallel
!$omp do private(lidiff) reduction (+:twonorm)
do j=1,n-1
  do i=1,n-1
    lidiff = (unew(i,j)-u(i,j))**2
    twonorm=twonorm+lidiff
  enddo
enddo
!$om enddo
!$omp end parallel
  twonorm=sqrt(twonorm)
If (twonorm.le.tol)
!... Declare convergence
```