COMP 705: Advanced Parallel Computing HW 3: Jacobian Iterative Solver for 2D Heat Equation

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Due: 09/27/17
Updated: 09/19/17

Table of Contents

General Code Requirements

- Modify code to process command line arguments for:
  - 2D processor distribution: $P(p_i, p_j)$
    - where $1 <= p_i <= NP$;
    - $NP = p_i \star p_j$
    - set default to be $p_i = p_j$
  - 2D data distribution: $N(n_i, n_j)$
    - where $1 <= n_i <= M$;
    - $M = n_i \star n_j$
    - set default to be $n_i = n_j$
    - what is M_{max}? Is it a function of the number of processors?
- Create routines for saving time slice of data (temperature).
- Replace code in function *neighbors* to use MPI_Cart_Shift
- Modify parallel code to use row or col communicators created using MPI_Cart functions
- You can choose to work with Kadin or Gropp code
- Writeup results in lab report format.

P1: Analysis

- Compare serial to parallel.
- Measure speedup/efficiency (compare to fig in below)
- Capture time slice data to generate video (or show multiple images on one page)
- Visualize heat propagation using Matlab, gnuplot, or other plotting routines.
- Create video using images:

```
files=$(/bin/ls Iter*png | sort -n ); echo $files; convert -delay 50 -loop 0 $files iter2.gif
```

Parallel Jacobian SOR solver Speedup

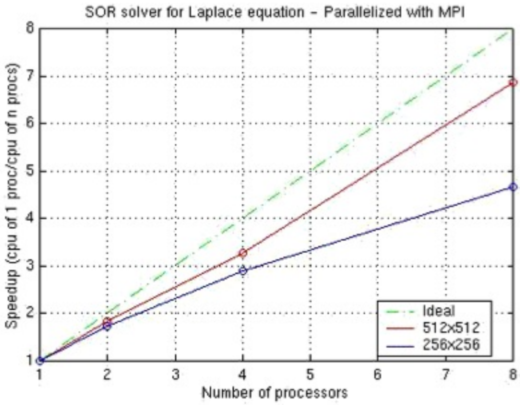
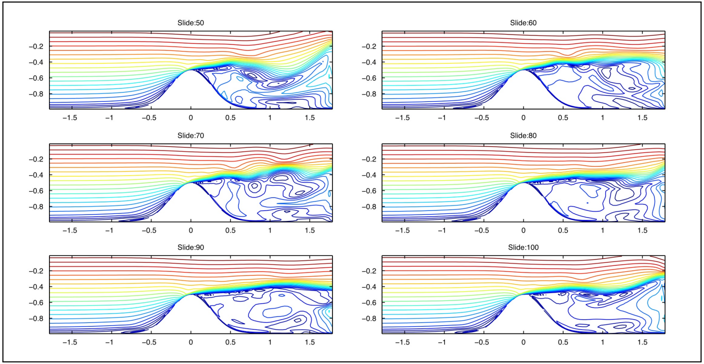


Figure shows the scalability of the MPI implementation of the Laplace equation using SOR on an SGI Origin 2000 shared-memory multiprocessor.

Source: `http://site.sci.hkbu.edu.hk/tdgc/tutorial.php`

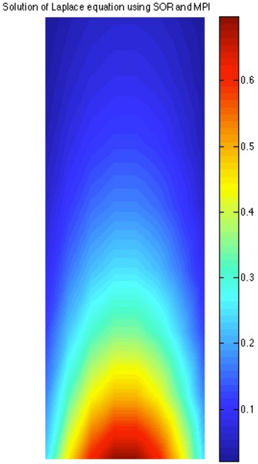
Plotting time evolution plots without using animation



Series of figures used to show time evolution of water flowing past a seamount.

Matlab code for visualizing heat propagation

- Provided with code base.
- Requires modification for file names (needs full path to file):
  define a HOME variable.
- "laplace.m" outputs image shown on right.



P1: Extra Credit

- Compare performance of send/recv, Isend, Irecv, and sendrecv collective communicator.
- Compare serial Jacobi, ser JacobiSOR, parJacobi, parJacobiSOR.

2D Laplacian - Heat Equation

2D Laplacian:
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

Boundary Conditions:

u(x,0) $= sin\,(\pi x)$ $\quad 0 <= x <= 1$

u(x,1) $= sin\,(\pi x)\, e^{-x}$ $\quad 0 \leq x \leq 1$

u(1,y) $= 0$ $\quad 0 <= y <= 1$

Analytical solution: $sin\,(\pi x)\, e^{-xy}$ $\quad (0 \leq x \leq 1); (0 \leq y \leq 1)$.

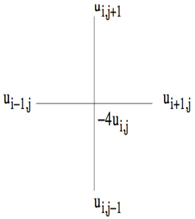
Jacobi Iterative Scheme

Jacobi Iteration - Finite Difference Approximation

Use Taylor Series expansion on uniform grid to yield linear system of equations

$$\nabla^2 u_{i,j} = \frac{1}{h^2}[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}] = 0$$

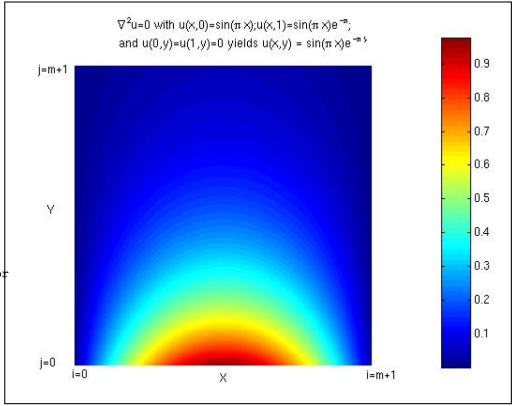
```
c => u(1:m  ,1:m  )    ! i  ,j  Current/Central
                              ! for 1<=i<=m; 1<=j<=m
n => u(1:m  ,2:m+1) ! i  ,j+1  North (of Current)
e => u(2:m+1,1:m )  ! i+1,j    East  (of Current)
w => u(0:m-1,1:m  ) ! i-1,j    West  (of Current)
s => u(1:m  ,0:m-1)  ! i  ,j-1  South (of Current)
```



Source: http://www.eng.utah.edu/~cs3200/notes/cs3200-Finite-Differences.pdf

Serial Jacobi Iterative Scheme - Boundary Conditions

```
  SUBROUTINE bc(u, m)
! PDE: Laplacian u = 0;      0<=x<=1;  0<=y<=1
! B.C.: u(x,0)=sin(pi*x);
!        u(x,1)=sin(pi*x)*exp(-pi); u(0,y)=u(1,y)=0
! SOLUTION: u(x,y)=sin(pi*x)*exp(-pi*y)
    IMPLICIT NONE
    INTEGER m, j
    REAL(real8), DIMENSION(0:m+1,0:m+1) :: u
    REAL(real8), DIMENSION(:,:), POINTER :: c
    REAL(real8), DIMENSION(0:m+1) :: y0
    y0 = sin(3.141593*(/(j,j=0,m+1)/)/(m+1))
    u = 0.0d0     ! at x=0,1; all y plus initialize interior
    u(:,  0) = y0               ! at y = 0; all x
    u(:,m+1) = y0*exp(-3.141593) ! at y = 1; all x
    RETURN
  END SUBROUTINE bc
```



Source: Kaden Notes: http://scv.bu.edu/~kadin/alliance/apply/solvers/

Serial Jacobi Iterative Scheme - Boundary Conditions

```
PROGRAM Jacobi
USE serial_jacobi_module
REAL(real8), DIMENSION(:,:), POINTER :: c, n, e, w, s

write(*,*)'Enter matrix size, m:'
read(*,*)m
! start timer, measured in seconds
CALL cpu_time(start_time)
 ! mem for unew, u
ALLOCATE ( unew(m,m), u(0:m+1,0:m+1) )

c => u(1:m  ,1:m  )    ! i  ,j  Current/Central
                              ! for 1<=i<=m; 1<=j<=m
n => u(1:m  ,2:m+1) ! i  ,j+1  North (of Current)
e => u(2:m+1,1:m ) ! i+1,j    East  (of Current)
w => u(0:m-1,1:m  ) ! i-1,j    West  (of Current)
s => u(1:m  ,0:m-1)  ! i  ,j-1  South (of Current)

CALL bc(u, m)          ! set up boundary values
```

```
 ! iterate until error below threshold
DO WHILE (gdel > tol)
  ! increment iteration counter
  iter = iter + 1
  IF(iter > 5000) THEN
    WRITE(*,*)'Iteration terminated (exceeds 5000)'
    STOP                       ! nonconvergent solution
  ENDIF
  unew = ( n + e + w + s )*0.25 ! new solution, Eq. 3
  gdel = MAXVAL(DABS(unew-c))    ! find local max error
  IF(MOD(iter,10)==0) WRITE(*,"('iter,gdel:',i6,e12.4)")iter,gdel
  c = unew                     ! update interior u
ENDDO

CALL CPU_TIME(end_time)      ! stop timer
PRINT *,'Total cpu time =',end_time - start_time,' x 1'
PRINT *,'Stopped at iteration =',iter
PRINT *,'The maximum error =',gdel

write(40,"(3i5)")m,m,1
write(41,"(6e13.4)")u
DEALLOCATE (unew, u)

END PROGRAM Jacobi
```

Source: Kaden Notes: http://scv.bu.edu/~kadin/alliance/apply/solvers/

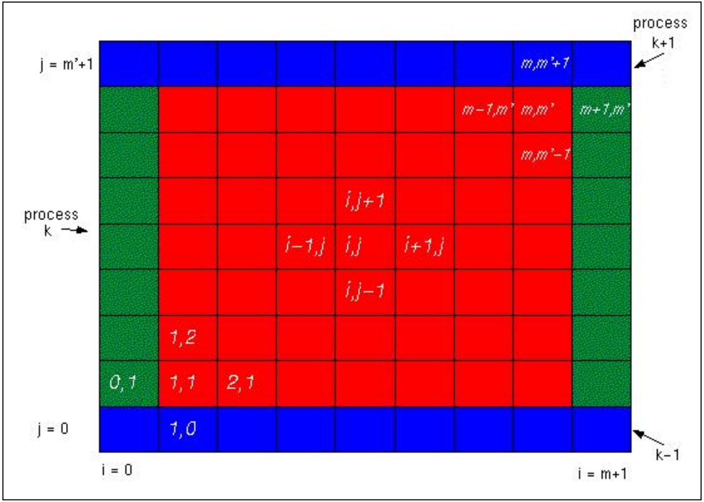
Parallel Jacobi Approach (in kadin code)

- Divide work evenly among processors (mxm/p),
- Divide work into P (number of PEs) horizontal strips
- Rewrite FD equation for solving u on PE k:

$$u_{i,j}^{n+1,k} = \frac{u_{i+1,j}^{n,k} \quad + \quad u_{i-1,j}^{n,k} \quad + \quad u_{i,j+1}^{n,k} \quad + \quad u_{i,j-1}^{n,k}}{4}$$

- n is the iteration number
- **Red** cells hold solution at iteration $(n+1)$
- **Blue** cells on top/bottom are the neighbor cells –¿ need to get them from other processor
- Green cells hold boundary conditions

Ghost Cell Layout



Source: Kaden Notes: `http://scv.bu.edu/~kadin/alliance/apply/solvers/`

Reading

- See papers listed in topics at: `https://edoras.sdsu.edu/~mthomas/f17.705/topics/iter_solv/`
- Useful Gropp Iterative Solvers and Advanced MPI Notes:
  - Kjoldstad and Gropp (2010), "Ghost Cell Pattern"
  - Gropp (2003), The 2-D Poisson Problem