MAE 267 – Project 3
Serial, Multi-Block, Finite-Volume Methods For Solving 2D Heat Conduction

Logan Halstrom
PhD Graduate Student Researcher
Center for Human/Robot/Vehicle Integration and Performance
Department of Mechanical and Aerospace Engineering
University of California, Davis
Davis, California 95616
Email: ldhalstrom@ucdavis.edu

1 Statement of Problem

This analysis details the solution of the steady-state temperature distribution on a 1m x 1m block of steel with Dirichlet boundary conditions (Eqn 2). Single-processor solutions were previously performed on a square, non-uniform grids rotatated in the positive z-direction by $rot = 30^o$. Two grids of 101x101 points and 501x501 points were used to solve the equation of heat transfer. Temperature was uniformly initialized to a value of 3.5 and the solution was iterated until the maximum residual found was less than $1.0x10^{-5}$. The equation for heat conduction (Eqn 1) was solved using an explicit, node-centered, finite-volume scheme, with an alternative distributive scheme for the second-derivative operator. Steady-state temperature distribution was saved in a PLOT3D unformatted file, and CPU wall time of the solver was recorded.

Now, the code has been modified to decompose the domain into sub-domains refered to as blocks. Boundary and neighbor information for each block is stored so that connectivity can be accurately assessed when communication between blocks is required. The block domain, associated meshes, and initial temperature distribution are initialized and then saved to restart files. These are read in at the beginning of the solver.

2 Equations and Algorithms

The solver developed for this analysis utilizes a finite-volume numerical solution method to solve the transient heat conduction equation (Eqn 1).

$$\rho c_p \frac{\partial T}{\partial t} = k\left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right] \tag{1}$$

The solution is initialized with the Dirichlet boundary conditions (Eqn 2).

$$T = \begin{cases} 5.0\left[\sin(\pi x_p) + 1.0\right] & \text{for } j = j_{max} \\ |\cos(\pi x_p)| + 1.0 & \text{for } j = 0 \\ 3.0 y_p + 2.0 & \text{for } i = 0,\, i_{max} \end{cases} \tag{2}$$

Grids were generated according to the following (Eqn 3)

$$\begin{aligned} rot &= 30.0\frac{\pi}{180.0} \\ x_p &= \cos\left[0.5\pi\frac{i_{max} - i}{i_{max} - 1}\right] \\ y_p &= \cos\left[0.5\pi\frac{j_{max} - j}{j_{max} - 1}\right] \\ x(i,j) &= x_p\cos(rot) + (1.0 - y_p)\sin(rot) \\ y(i,j) &= y_p\cos(rot) + x_p\sin(rot) \end{aligned} \tag{3}$$

To solve Eqn 1 numerically, the equation is discretized according to a node-centered finite-volume scheme, where first-derivatives at the nodes are found using Green's theorem integrating around the secondary control volumes. Trapezoidal, counter-clockwise integration for the first-derivative in the x-direction is achieved with Eqn 4.

$$\begin{aligned} \frac{\partial T}{\partial x} = \frac{1}{2Vol_{i+\frac{1}{2},j+\frac{1}{2}}}\big[&(T_{i+1,j} + T_{i+1,j+1})Ayi_{i+1,j} \\ &- (T_{i,j} + T_{i,j+1})Ayi_{i,j} \\ &- (T_{i,j+1} + T_{i+1,j+1})Ayi_{i,j+1} \\ &- (T_{i,j} + T_{i+1,j})Ayi_{i,j}\big] \end{aligned} \tag{4}$$

A similar scheme is used to find the first-derivative in the y-direction.

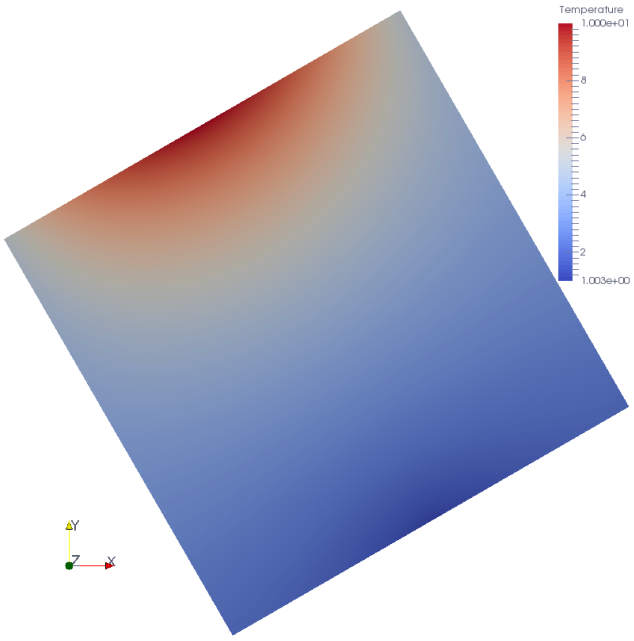

Fig. 1: Steady-state temperature solution for 501x501 grid decomposed into 10x10 blocks

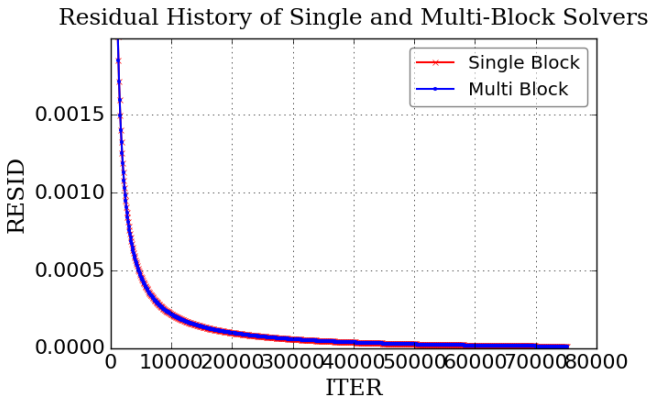

Fig. 2: Residual history for solving a 501x501 grid with a single block solver and a 10x10 multi-block solver

3 Results and Discussion

All simulations in this analysis were run on a 501x501 point grid, once with a single block solver and once with at 10x10 multi-block solver. Fig 1 portrays the multi-block solution, which is comparable to that of the single block solver. Convergence histories of the two solvers are compared in Fig 2. It can be seen that the two solvers are comparable in performance, both following a similar convergence path and converging at almost the same iteration.

Actual solver times are compared in Appendix A. The multi-block solver was found to be approximately 11 seconds (2.6%) faster than the single block solver. This may be due to more code streamlining in the later project. It can be expected that the speed of the multi-block solver will improve even further when linked-lists are employed to navigate neighbor boundary actions (this capability is currently functional in the code, but does not work on HPC1, so a logic-based approach was used for this project.)

4 Conclusion

Decomposing the domain introduced unforseen complications in adapting the single block solver. In some cases, it was as simple as adding a third loop for the block number, but in others (especially in updating the ghost nodes) considerable thought and error-checking was required. This implies that adapting the code for parallel processing will be an equally complicated step, so it is beneficial that we are adapting our codes modularly in stages.

Appendix A: Solver Performace Comparison

```
 Running a         501 by         501 grid took:
       75128 iterations
   428.74140000343323      seconds (Total CPU walltime)
   428.61560106277466      seconds (Solver CPU walltime)

 Found max residual of    9.9999670980684130E-006
 At ij of          191         242
```

Listing 1: Single block solver performance

```
 Running a         501 by         501 grid,
 With NxM:          10 x          10 blocks took:
       75128 iterations
   417.04543089866638      seconds (Total CPU walltime)
   415.74189305305481      seconds (Solver CPU walltime)

 Found max residual of    9.9999670985857943E-006
 on block id          44
 At ij of           41          42
```

Listing 2: Multi block solver performance

Appendix B: Multi-Block Grid Decomposition Code

```
! MAE 267
! PROJECT 3
! LOGAN HALSTROM
! 03 NOVEMBER 2015

! DESCRIPTION:  Modules used for solving heat conduction of steel plate.
! Initialize and store constants used in all subroutines.

! CONTENTS:
! CONSTANTS --> Initializes constants for simulation.  Sets grid size.
! CLOCK --> Calculates clock wall-time of a process.
! MAKEGRID --> Initialize grid with correct number of points and rotation,
!                 set boundary conditions, etc.
! CELLS -->  Initialize finite volume cells and do associated calculations
! TEMPERATURE --> Calculate and store new temperature distribution
!                     for given iteration

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! CONSTANTS MODULE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

MODULE CONSTANTS
    ! Initialize constants for simulation.  Set grid size.
    IMPLICIT NONE
    ! CFL number, for convergence (D0 is double-precision, scientific notation)
    REAL(KIND=8), PARAMETER :: CFL = 0.95D0
    ! Material constants (steel): thermal conductivity [W/(m*K)],
                                ! density [kg/m^3],
                                ! specific heat ratio [J/(kg*K)]
                                ! initial temperature
    REAL(KIND=8), PARAMETER :: k = 18.8D0, rho = 8000.D0, cp = 500.D0, T0 = 3.5D0
    ! Thermal diffusivity [m^2/s]
    REAL(KIND=8), PARAMETER :: alpha = k / (cp * rho)
    ! Pi, grid rotation angle (30 deg)
    REAL(KIND=8), PARAMETER :: pi = 3.141592654D0, rot = 30.D0*pi/180.D0
    ! ITERATION PARAMETERS
    ! Minimum Residual
    REAL(KIND=8) :: min_res = 0.00001D0
    ! Maximum number of iterations
    INTEGER :: max_iter = 1000000
    ! CPU Wall Times
```

```
    REAL(KIND=8) :: wall_time_total, wall_time_solve, wall_time_iter(1:5)
    ! read square grid size, Total grid size, size of grid on each block (local)
    INTEGER :: nx, IMAX, JMAX, IMAXBLK, JMAXBLK
    ! Dimensions of block layout, Number of Blocks,
    INTEGER :: M, N, NBLK
    ! Block boundary condition identifiers
        ! If block face is on North,east,south,west of main grid, identify
!     INTEGER :: NBND = 1, SBND = 2, EBND = 3, WBND = 4
    INTEGER :: NBND = -1, EBND = -2, SBND = -3, WBND = -4
    ! Output directory
    CHARACTER(LEN=18) :: casedir
    ! Debug mode = 1
    INTEGER :: DEBUG
    ! Value for constant temperature BCs for debugging
    REAL(KIND=8), PARAMETER :: TDEBUG = T0 - T0 * 0.5

CONTAINS

    SUBROUTINE read_input()
        INTEGER :: I
        CHARACTER(LEN=3) :: strNX
        CHARACTER(LEN=1) :: strN, strM

        ! READ INPUTS FROM FILE
            !(So I don't have to recompile each time I change an input setting)
!         WRITE(*,*) ''
!         WRITE(*,*) 'Reading input...'
        OPEN (UNIT = 1, FILE = 'config.in')
        DO I = 1, 3
            ! Skip header lines
            READ(1,*)
        END DO
        ! READ GRIDSIZE (4th line)
        READ(1,*) nx
        ! READ BLOCKS (6th and 8th line)
        READ(1,*)
        READ(1,*) M
        READ(1,*)
        READ(1,*) N
        ! DEBUG MODE (10th line)
        READ(1,*)
        READ(1,*) DEBUG

        ! SET GRID SIZE
        IMAX = nx
        JMAX = nx
        ! CALC NUMBER OF BLOCKS
        NBLK = M * N
        ! SET SIZE OF EACH BLOCK (LOCAL MAXIMUM I, J)
        IMAXBLK = 1 + (IMAX - 1) / N
        JMAXBLK = 1 + (JMAX - 1) / M

!         ! OUTPUT DIRECTORIES
!         ! write integers to strings
!         WRITE( strNX, '(I3)') nx
!         IF ( N - 10 < 0 ) THEN
!             ! N is a single digit (I1)
!             WRITE( strN,  '(I1)') N
!         ELSE
!             ! N is a tens digit
!             WRITE( strN,  '(I2)') N
!         END IF
!         IF ( M - 10 < 0 ) THEN
!             WRITE( strM,  '(I1)') M
!         ELSE
!             WRITE( strM,  '(I2)') M
!         END IF
!         ! case output directory: nx_NxM (i.e. 'Results/101_5x4')
!         casedir = 'Results/' // strNX // '_' // strN // 'x' // strM // '/'
```

```
!         ! MAKE DIRECTORIES (IF THEY DONT ALREADY EXIST)
!         CALL EXECUTE_COMMAND_LINE ("mkdir -p " // TRIM(casedir) )

        ! OUTPUT TO SCREEN
        WRITE(*,*) ''
        WRITE(*,*) 'Solving Mesh of size ixj:', IMAX, 'x', JMAX
        WRITE(*,*) 'With MxN blocks:', M, 'x', N
        WRITE(*,*) 'Number of blocks:', NBLK
        WRITE(*,*) 'Block size ixj:', IMAXBLK, 'x', JMAXBLK
        IF (DEBUG == 1) THEN
            WRITE(*,*) 'RUNNING IN DEBUG MODE'
        END IF
        WRITE(*,*) ''
    END SUBROUTINE read_input
END MODULE CONSTANTS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!! BLOCK GRID MODULE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

MODULE BLOCKMOD
    ! Initialize grid with correct number of points and rotation,
    ! set boundary conditions, etc.
    USE CONSTANTS

    IMPLICIT NONE
    PUBLIC

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!! DERIVED DATA TYPES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    ! DERIVED DATA TYPE FOR GRID INFORMATION

    TYPE MESHTYPE
        ! Grid points, see cooridinate rotaion equations in problem statement
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xp, yp, x, y
        ! Temperature at each point, temporary variable to hold temperature sum
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: T, Ttmp
        ! Iteration Parameters: timestep, cell volume, secondary cell volume,
                                ! equation constant term
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: dt, V, V2nd, term
        ! Areas used in alternative scheme to get fluxes for second-derivative
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: Ayi, Axi, Ayj, Axj
        ! Second-derivative weighting factors for alternative distribution scheme
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: yPP, yNP, yNN, yPN
        REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xNN, xPN, xPP, xNP
    END TYPE MESHTYPE

    ! DATA TYPE FOR INFORMATION ABOUT NEIGHBORS

    TYPE NBRTYPE
        ! Information about face neighbors (north, east, south, west)
            ! And corner neighbors (Northeast, southeast, southwest, northwest)
        INTEGER :: N, E, S, W, NE, SE, SW, NW
    END TYPE NBRTYPE

    ! DERIVED DATA TYPE WITH INFORMATION PERTAINING TO SPECIFIC BLOCK

    TYPE BLKTYPE
        ! DER. DATA TYPE STORES LOCAL MESH INFO
        TYPE(MESHTYPE) :: mesh
        ! IDENTIFY FACE AND CORNER NEIGHBOR BLOCKS AND PROCESSORS
        TYPE(NBRTYPE) :: NB, NP
        ! BLOCK NUMBER
        INTEGER :: ID
        ! GLOBAL INDICIES OF MINIMUM AND MAXIMUM INDICIES OF BLOCK
        INTEGER :: IMIN, IMAX, JMIN, JMAX
        ! LOCAL ITERATION BOUNDS TO AVOID UPDATING BC'S + UTILIZE GHOST NODES
```

```
        INTEGER :: IMINLOC, JMINLOC, IMAXLOC, JMAXLOC, IMINUPD, JMINUPD
        ! BLOCK ORIENTATION
        INTEGER :: ORIENT
    END TYPE BLKTYPE

    ! LINKED LIST: RECURSIVE POINTER THAT POINTS THE NEXT ELEMENT IN THE LIST

    TYPE LNKLIST
        ! Next element in linked list
        TYPE(LNKLIST), POINTER :: next
        ! Identify what linked list belongs to
        INTEGER :: ID
    END TYPE LNKLIST

    ! Collection of linked lists for faces and corners

    TYPE NBRLIST
        TYPE(LNKLIST), POINTER :: N, E, S, W, NE, SE, SW, NW
    END TYPE NBRLIST

CONTAINS

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! INITIALIZE GRID AND WRITE TO FILE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    SUBROUTINE init_blocks(b)
        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: b(:)
        ! Neighbor information pointer
        TYPE(NBRTYPE), POINTER :: NB
        ! COUNTER VARIABLES
            ! IM, IN COUNT BLOCK INDICIES
            ! (IBLK COUNTS BLOCK NUMBERS, INBR IS BLOCK NEIGHBOR INDEX)
        INTEGER :: I, J, IBLK, INBR

        ! STEP THROUGH BLOCKS, ASSIGN IDENTIFYING INFO
        !
        !               |                |
        !               |     North      |
        !             NW|  (IBLK + N)    |NE
        ! (IBLK + N - 1)|                |(IBLK + N + 1)
        ! ----------------------------------------------
        !               |                |
        !     West      |   Current      |    East
        !   (IBLK - 1)  |     (IBLK)     |  (IBLK + 1)
        !               |                |
        ! ----------------------------------------------
        !             SW|                |SE
        ! (IBLK - N - 1)|     South      |(IBLK - N + 1)
        !               |  (IBLK - N)    |
        !               |                |
        !

        ! START AT BLOCK 1 (INCREMENT IN LOOP)
        IBLK = 0

        DO J = 1, M
            DO I = 1, N
                ! INCREMENT BLOCK NUMBER
                IBLK = IBLK + 1

                ! Neighbor information pointer
                NB => b(IBLK)%NB

                ! ASSIGN BLOCK NUMBER
                b(IBLK)%ID = IBLK
                ! ASSIGN GLOBAL MIN/MAX INDICIES OF LOCAL GRID
                b(IBLK)%IMIN = 1 + (IMAXBLK - 1) * (I - 1)
```

```
                b(IBLK)%JMIN = 1 + (JMAXBLK - 1) * (J - 1)
                b(IBLK)%IMAX = b(IBLK)%IMIN + (IMAXBLK - 1)
                b(IBLK)%JMAX = b(IBLK)%JMIN + (JMAXBLK - 1)

                ! ASSIGN NUMBERS OF FACE AND CORNER NEIGHBOR BLOCKS
                    !if boundary face, assign bc later
                NB%N  = IBLK + N
                NB%S  = IBLK - N
                NB%E  = IBLK + 1
                NB%W  = IBLK - 1
                NB%NE = IBLK + N + 1
                NB%NW = IBLK + N - 1
                NB%SW = IBLK - N - 1
                NB%SE = IBLK - N + 1

                ! Assign faces and corners on boundary of the actual
                ! computational grid with number corresponding to which
                ! boundary they are on.
                    ! Corners on actual corners of the computational grid are
                    ! ambiguously assigned.
                IF ( b(IBLK)%JMAX == JMAX ) THEN
                    ! NORTH BLOCK FACE AND CORNERS ARE ON MESH NORTH BOUNDARY
                        ! AT ACTUAL CORNERS OF MESH, CORNERS ARE AMBIGUOUS
                    NB%N  = NBND
                    NB%NE = NBND
                    NB%NW = NBND
                END IF
                IF ( b(IBLK)%IMAX == IMAX ) THEN
                    ! EAST BLOCK FACE IS ON MESH EAST BOUNDARY
                    NB%E  = EBND
                    NB%NE = EBND
                    NB%SE = EBND

                END IF
                IF ( b(IBLK)%JMIN == 1 ) THEN
                    ! SOUTH BLOCK FACE IS ON MESH SOUTH BOUNDARY
                    NB%S  = SBND
                    NB%SE = SBND
                    NB%SW = SBND
                END IF
                IF ( b(IBLK)%IMIN == 1 ) THEN
                    ! WEST BLOCK FACE IS ON MESH WEST BOUNDARY
                    NB%W  = WBND
                    NB%SW = WBND
                    NB%NW = WBND
                END IF

                ! BLOCK ORIENTATION
                    ! same for all in this project
                b(IBLK)%ORIENT = 1

            END DO
        END DO
    END SUBROUTINE init_blocks

    SUBROUTINE write_blocks(b)
        ! WRITE BLOCK CONNECTIVITY FILE

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE) :: b(:)
        INTEGER :: I, BLKFILE = 99

        11 format(3I5)
        22 format(33I5)

!         OPEN (UNIT = BLKFILE , FILE = TRIM(casedir) // "blockconfig.dat", form='formatted')
        OPEN (UNIT = BLKFILE , FILE = "blockconfig.dat", form='formatted')
        ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
        WRITE(BLKFILE, 11) NBLK, IMAXBLK, JMAXBLK
```

```
318         DO I = 1, NBLK
319             ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
320             ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
321             ! NORTH EAST SOUTH WEST
322             WRITE(BLKFILE, 22) b(I)%ID, &
323                 b(I)%IMIN, b(I)%JMIN, &
324                 b(I)%NB%N, &
325                 b(I)%NB%NE, &
326                 b(I)%NB%E, &
327                 b(I)%NB%SE, &
328                 b(I)%NB%S, &
329                 b(I)%NB%SW, &
330                 b(I)%NB%W, &
331                 b(I)%NB%NW, &
332                 b(I)%ORIENT
333         END DO
334         CLOSE(BLKFILE)
335     END SUBROUTINE write_blocks
336
337     SUBROUTINE read_blocks(b)
338         ! READ BLOCK CONNECTIVITY FILE
339
340         ! BLOCK DATA TYPE
341         TYPE(BLKTYPE) :: b(:)
342         INTEGER :: I, BLKFILE = 99
343         ! READ INFOR FOR BLOCK DIMENSIONS
344         INTEGER :: NBLKREAD, IMAXBLKREAD, JMAXBLKREAD
345
346         11 format(3I5)
347         22 format(33I5)
348
349 !         OPEN (UNIT = BLKFILE , FILE = TRIM(casedir) // "blockconfig.dat", form='formatted')
350         OPEN (UNIT = BLKFILE , FILE = "blockconfig.dat", form='formatted')
351         ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
352         READ(BLKFILE, 11) NBLK, IMAXBLK, JMAXBLK
353         DO I = 1, NBLK
354             ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
355             ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
356             ! NORTH EAST SOUTH WEST
357             READ(BLKFILE, 22) b(I)%ID, &
358                 b(I)%IMIN, b(I)%JMIN, &
359                 b(I)%NB%N, &
360                 b(I)%NB%NE, &
361                 b(I)%NB%E, &
362                 b(I)%NB%SE, &
363                 b(I)%NB%S, &
364                 b(I)%NB%SW, &
365                 b(I)%NB%W, &
366                 b(I)%NB%NW, &
367                 b(I)%ORIENT
368         END DO
369         CLOSE(BLKFILE)
370     END SUBROUTINE read_blocks
371
372     SUBROUTINE init_mesh(b)
373         ! BLOCK DATA TYPE
374         TYPE(BLKTYPE), TARGET :: b(:)
375         TYPE(MESHTYPE), POINTER :: m
376         INTEGER :: IBLK, I, J
377
378         DO IBLK = 1, NBLK
379
380             m => b(IBLK)%mesh
381
382             ! ALLOCATE MESH INFORMATION
383                 ! ADD EXTRA INDEX AT BEGINNING AND END FOR GHOST NODES
384             ALLOCATE( m%xp(  0:IMAXBLK+1,   0:JMAXBLK+1) )
385             ALLOCATE( m%yp(  0:IMAXBLK+1,   0:JMAXBLK+1) )
386             ALLOCATE( m%x(   0:IMAXBLK+1,   0:JMAXBLK+1) )
```

```
            ALLOCATE( m%y(   0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%T(   0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%Ttmp(0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%dt(  0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%V2nd(0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%term(0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%Ayi( 0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%Axi( 0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%Ayj( 0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%Axj( 0:IMAXBLK+1,   0:JMAXBLK+1) )
            ALLOCATE( m%V(   0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%yPP( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%yNP( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%yNN( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%yPN( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%xNN( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%xPN( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%xPP( 0:IMAXBLK,     0:JMAXBLK  ) )
            ALLOCATE( m%xNP( 0:IMAXBLK,     0:JMAXBLK  ) )

            ! STEP THROUGH LOCAL INDICIES OF EACH BLOCK
            DO J = 0, JMAXBLK+1
                DO I = 0, IMAXBLK+1
                    ! MAKE SQUARE GRID
                        ! CONVERT FROM LOCAL TO GLOBAL INDEX:
                            ! Iglobal = Block%IMIN + (Ilocal - 1)
                    m%xp(I, J) = COS( 0.5D0 * PI * DFLOAT(IMAX - ( b(IBLK)%IMIN + I - 1) ) / DFLOAT(IMAX - 1) )
                    m%yp(I, J) = COS( 0.5D0 * PI * DFLOAT(JMAX - ( b(IBLK)%JMIN + J - 1) ) / DFLOAT(JMAX - 1) )
                    ! ROTATE GRID
                    m%x(I, J) = m%xp(I, J) * COS(rot) + (1.D0 - m%yp(I, J) ) * SIN(rot)
                    m%y(I, J) = m%yp(I, J) * COS(rot) + (       m%xp(I, J) ) * SIN(rot)
                END DO
            END DO
        END DO
    END SUBROUTINE init_mesh

    SUBROUTINE init_temp(blocks)
        ! Initialize temperature across mesh
        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET  :: blocks(:)
        TYPE(BLKTYPE), POINTER :: b
        TYPE(MESHTYPE), POINTER :: m
        TYPE(NBRTYPE), POINTER :: NB
        INTEGER :: IBLK, I, J

        DO IBLK = 1, NBLK
            b => blocks(IBLK)
            m => blocks(IBLK)%mesh
            NB => blocks(IBLK)%NB
            ! FIRST, INITIALIZE ALL POINT TO INITIAL TEMPERATURE (T0)
            m%T(0:IMAXBLK+1, 0:JMAXBLK+1) = T0
            ! THEN, INITIALIZE BOUNDARIES DIRICHLET B.C.
            IF (DEBUG /= 1) THEN

                ! DIRICHLET B.C.
                ! face on north boundary
                IF (NB%N == NBND) THEN
                    DO I = 1, IMAXBLK
                        m%T(I, JMAXBLK) = 5.D0 * (SIN(PI * m%xp(I, JMAXBLK)) + 1.D0)
                    END DO
                END IF
                IF (NB%S == SBND) THEN
                    DO I = 1, IMAXBLK
                        m%T(I, 1) = ABS(COS(PI * m%xp(I, 1))) + 1.D0
                    END DO
                END IF
                IF (NB%E == EBND) THEN
                    DO J = 1, JMAXBLK
                        m%T(IMAXBLK, J) = 3.D0 * m%yp(IMAXBLK, J) + 2.D0
```

```
                    END DO
                END IF
                IF (NB%W == WBND) THEN
                    DO J = 1, JMAXBLK
                        m%T(1, J) = 3.D0 * m%yp(1, J) + 2.D0
                    END DO
                END IF

            ELSE

                ! DEBUG BCS
                IF (NB%N < 0) THEN
                    DO I = 1, IMAXBLK
                        m%T(I, JMAXBLK) = TDEBUG
                    END DO
                END IF
                IF (NB%S < 0) THEN
                    DO I = 1, IMAXBLK
                        m%T(I, 1) = TDEBUG
                    END DO
                END IF
                IF (NB%E < 0) THEN
                    DO J = 1, JMAXBLK
                        m%T(IMAXBLK, J) = TDEBUG
                    END DO
                END IF
                IF (NB%W < 0) THEN
                    DO J = 1, JMAXBLK
                        m%T(1, J) = TDEBUG
                    END DO
                END IF
            END IF
        END DO
    END SUBROUTINE init_temp



    ! WRITE GRID HERE!!!!!!!!!!!!!!!!!!!!!

    ! WRITE TEMPERATURE HERE!!!!!!!!!!!!!!!!!!!!!!


    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! INITIALIZE SOLUTION AFTER RESTART FILE READ IN !!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


    ! READ BLOCKS HERE!!!!!!!!!!!!!!!!!!!!!!




    SUBROUTINE set_block_bounds(blocks)
        ! Calculate iteration bounds for each block to avoid updating BCs.
        ! Populate block ghost nodes from initial temperature distribution
        ! call after reading in mesh data from restart file
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(BLKTYPE), POINTER :: b
        TYPE(NBRTYPE), POINTER :: NB
        INTEGER :: IBLK, I, J

        DO IBLK = 1, NBLK
            b => blocks(IBLK)
            NB => b%NB

            ! Set iteration bounds of each block to preserve BCs
                ! south and west boundaries:
                    ! interior: iminloc, jminloc = 0 (use ghost)
```

```
525                     ! boundary: iminloc, jminloc = 2 (1st index is BC)
526                 ! north and east boundaries:
527                     ! interior: imaxloc, jmaxloc = maxblk (use ghost)
528                     ! boundary: imaxloc, jmaxloc = maxblk-1 (max index is BC)
529
530             ! NORTH
531             IF (NB%N > 0) THEN
532                 ! Interior faces have positive ID neighbors
533                 b%JMAXLOC = JMAXBLK
534             ELSE
535                 ! At North Boundary
536                 b%JMAXLOC = JMAXBLK - 1
537             END IF
538
539             ! EAST
540             IF (NB%E > 0) THEN
541                 ! Interior
542                 b%IMAXLOC = IMAXBLK
543             ELSE
544                 ! At east Boundary
545                 b%IMAXLOC = IMAXBLK - 1
546             END IF
547
548             ! SOUTH
549             IF (NB%S > 0) THEN
550                 ! Interior
551                 b%JMINLOC = 0
552             ELSE
553                 ! At south Boundary
554                 b%JMINLOC = 1
555                 ! boundary for updating temperature (dont update BC)
556                 b%JMINUPD = 2
557             END IF
558
559             ! WEST
560             IF (NB%W > 0) THEN
561                 ! Interior
562                 b%IMINLOC = 0
563             ELSE
564                 ! At west Boundary
565                 b%IMINLOC = 1
566                 b%IMINUPD = 2
567             END IF
568         END DO
569     END SUBROUTINE set_block_bounds
570
571     SUBROUTINE init_linklists(blocks, nbrlists)
572         ! Create linked lists governing block boundary communication
573         ! BLOCK DATA TYPE
574         TYPE(BLKTYPE), TARGET :: blocks(:)
575         ! Neighbor information pointer
576         TYPE(NBRTYPE), POINTER :: NB
577         ! Linked lists of neighbor communication instructions
578         TYPE(NBRLIST) :: nbrlists
579         TYPE(NBRLIST) :: nbrl
580         INTEGER :: IBLK
581
582         DO IBLK = 1, NBLK
583             NB => blocks(IBLK)%NB
584
585             ! NORTH
586             ! If block north face is internal, add it to appropriate linked list
587             ! for north internal faces.
588             IF (NB%N > 0) THEN
589
590                 IF ( .NOT. ASSOCIATED(nbrlists%N) ) THEN
591                     ! Allocate linked list if it hasnt been accessed yet
592                     ALLOCATE(nbrlists%N)
593                     ! Pointer linked list that will help iterate through the
```

```
                    ! primary list in this loop
                    nbrl%N => nbrlists%N
                ELSE
                    ! linked list already allocated (started).  Allocate next
                    ! link as assign current block to it
                    ALLOCATE(nbrl%N%next)
                    nbrl%N => nbrl%N%next
                END IF

                ! associate this linked list entry with the current block
                nbrl%N%ID = IBLK
                ! break link to pre-existing pointer target.  We will
                ! allocated this target later as the next item in the linked list
                NULLIFY(nbrl%N%next)
            END IF

            ! SOUTH
            IF (NB%S > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%S) ) THEN
                    ALLOCATE(nbrlists%S)
                    nbrl%S => nbrlists%S
                ELSE
                    ALLOCATE(nbrl%S%next)
                    nbrl%S => nbrl%S%next
                END IF
                nbrl%S%ID = IBLK
                NULLIFY(nbrl%S%next)
            END IF

            ! EAST
            IF (NB%E > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%E) ) THEN
                    ALLOCATE(nbrlists%E)
                    nbrl%E => nbrlists%E
                ELSE
                    ALLOCATE(nbrl%E%next)
                    nbrl%E => nbrl%E%next
                END IF
                nbrl%E%ID = IBLK
                NULLIFY(nbrl%E%next)
            END IF

            ! WEST
            IF (NB%W > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%W) ) THEN
                    ALLOCATE(nbrlists%W)
                    nbrl%W => nbrlists%W
                ELSE
                    ALLOCATE(nbrl%W%next)
                    nbrl%W => nbrl%W%next
                END IF
                nbrl%W%ID = IBLK
                NULLIFY(nbrl%W%next)
            END IF

            ! NORTH EAST
            IF (NB%NE > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%NE) ) THEN
                    ALLOCATE(nbrlists%NE)
                    nbrl%NE => nbrlists%NE
                ELSE
                    ALLOCATE(nbrl%NE%next)
                    nbrl%NE => nbrl%NE%next
                END IF
                nbrl%NE%ID = IBLK
                NULLIFY(nbrl%NE%next)
            END IF

            ! SOUTH EAST
```

```
            IF (NB%SE > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%SE) ) THEN
                    ALLOCATE(nbrlists%SE)
                    nbrl%SE => nbrlists%SE
                ELSE
                    ALLOCATE(nbrl%SE%next)
                    nbrl%SE => nbrl%SE%next
                END IF
                nbrl%SE%ID = IBLK
                NULLIFY(nbrl%SE%next)
            END IF

            ! SOUTH WEST
            IF (NB%SW > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%SW) ) THEN
                    ALLOCATE(nbrlists%SW)
                    nbrl%SW => nbrlists%SW
                ELSE
                    ALLOCATE(nbrl%SW%next)
                    nbrl%SW => nbrl%SW%next
                END IF
                nbrl%SW%ID = IBLK
                NULLIFY(nbrl%SW%next)
            END IF

            ! NORTH WEST
            IF (NB%NW > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%NW) ) THEN
                    ALLOCATE(nbrlists%NW)
                    nbrl%NW => nbrlists%NW
                ELSE
                    ALLOCATE(nbrl%NW%next)
                    nbrl%NW => nbrl%NW%next
                END IF
                nbrl%NW%ID = IBLK
                NULLIFY(nbrl%NW%next)
            END IF
        END DO
    END SUBROUTINE init_linklists

    SUBROUTINE update_ghosts(b, nbrlists)
        ! Update ghost nodes of each block based on neightbor linked lists

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: b(:)
        ! temperature information pointers for ghost and neighbor nodes
        REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
        ! Linked lists of neighbor communication instructions
        TYPE(NBRLIST) :: nbrlists
        TYPE(NBRLIST) :: nbrl
        ! iteration parameters, index of neighbor
        INTEGER :: I, J, INBR

        !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! NORTH FACE GHOST NODES
        nbrl%N => nbrlists%N
        ! Step through linked list of north faces with ghosts until end of list
        DO
            ! If next link in list doesnt exist (end of list), stop loop
            IF ( .NOT. ASSOCIATED(nbrl%N) ) EXIT

            ! Otherwise, assign neighbor values to all ghost nodes:

            ! TEMPERATURE OF CURRENT BLOCK (CONTAINS GHOST NODES)
                ! (identified by linked list id)
            Tgh => b( nbrl%N%ID )%mesh%T

            ! index of north neighbor
```

```
            INBR = b( nbrl%N%ID )%NB%N
            ! TEMPERATURE OF NEIGHBOR BLOCK (UPDATE GHOSTS WITH THIS)
            Tnb => b( INBR )%mesh%T

            DO I = 1, IMAXBLK
                ! NORTH FACE GHOST NODE TEMPERATURE IS EQUAL TO TEMPERATURE OF
                ! SECOND-FROM-SOUTH FACE OF NORTH NEIGHBOR
                ! (Remember face nodes are shared between blocks)
                Tgh(I, JMAXBLK+1) = Tnb(I, 2)
            END DO
            ! switch pointer to next link in list
            nbrl%N => nbrl%N%next
        END DO

        ! SOUTH FACE GHOST NODES
        nbrl%S => nbrlists%S
        DO
            IF ( .NOT. ASSOCIATED(nbrl%S) ) EXIT
            Tgh => b( nbrl%S%ID )%mesh%T
            INBR = b( nbrl%S%ID )%NB%S
            Tnb => b( INBR )%mesh%T

            DO I = 1, IMAXBLK
                ! ADD NORTH FACE OF SOUTH NEIGHBOR TO CURRENT SOUTH FACE GHOSTS
                Tgh(I, 0) = Tnb(I, JMAXBLK-1)
            END DO
            nbrl%S => nbrl%S%next
        END DO

        ! EAST FACE GHOST NODES
        nbrl%E => nbrlists%E
        DO
            IF ( .NOT. ASSOCIATED(nbrl%E) ) EXIT
            Tgh => b( nbrl%E%ID )%mesh%T
            INBR = b( nbrl%E%ID )%NB%E
            Tnb => b( INBR )%mesh%T

            DO J = 1, JMAXBLK
                ! ADD WEST FACE OF EAST NEIGHBOR TO CURRENT WEST FACE GHOSTS
                Tgh(IMAXBLK+1, J) = Tnb(2, J)
            END DO
            nbrl%E => nbrl%E%next
        END DO

        ! WEST FACE GHOST NODES
        nbrl%W => nbrlists%W
        DO
            IF ( .NOT. ASSOCIATED(nbrl%W) ) EXIT
            Tgh => b( nbrl%W%ID )%mesh%T
            INBR = b( nbrl%W%ID )%NB%W
            Tnb => b( INBR )%mesh%T

            DO J = 1, JMAXBLK
                ! ADD EAST FACE OF WEST NEIGHBOR TO CURRENT EAST FACE GHOSTS
                Tgh(0, J) = Tnb(IMAXBLK-1, J)
            END DO
            nbrl%W => nbrl%W%next
        END DO

        !!! CORNERS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! NORTH EAST CORNER GHOST NODES
        nbrl%NE => nbrlists%NE
        DO
            IF ( .NOT. ASSOCIATED(nbrl%NE) ) EXIT
            Tgh => b( nbrl%NE%ID )%mesh%T
            INBR = b( nbrl%NE%ID )%NB%NE
            Tnb => b( INBR )%mesh%T
            ! ADD SW CORNER OF NE NEIGHBOR TO CURRENT NE CORNER GHOSTS
```

```
            Tgh(IMAXBLK+1, JMAXBLK+1) = Tnb(2, 2)
            nbrl%NE => nbrl%NE%next
        END DO

        ! SOUTH EAST CORNER GHOST NODES
        nbrl%SE => nbrlists%SE
        DO
            IF ( .NOT. ASSOCIATED(nbrl%SE) ) EXIT
            Tgh => b( nbrl%SE%ID )%mesh%T
            INBR = b( nbrl%SE%ID )%NB%SE
            Tnb => b( INBR )%mesh%T
            ! ADD NW CORNER OF SE NEIGHBOR TO CURRENT SE CORNER GHOSTS
            Tgh(IMAXBLK+1, 0) = Tnb(2, JMAXBLK-1)
            nbrl%SE => nbrl%SE%next
        END DO

        ! SOUTH WEST CORNER GHOST NODES
        nbrl%SW => nbrlists%SW
        DO
            IF ( .NOT. ASSOCIATED(nbrl%SW) ) EXIT
            Tgh => b( nbrl%SW%ID )%mesh%T
            INBR = b( nbrl%SW%ID )%NB%SW
            Tnb => b( INBR )%mesh%T
            ! ADD NE CORNER OF SW NEIGHBOR TO CURRENT SW CORNER GHOSTS
            Tgh(0, 0) = Tnb(IMAXBLK-1, JMAXBLK-1)
            nbrl%SW => nbrl%SW%next
        END DO

        ! NORTH WEST CORNER GHOST NODES
        nbrl%NW => nbrlists%NW
        DO
            IF ( .NOT. ASSOCIATED(nbrl%NW) ) EXIT
            Tgh => b( nbrl%NW%ID )%mesh%T
            INBR = b( nbrl%NW%ID )%NB%NW
            Tnb => b( INBR )%mesh%T
            ! ADD SE CORNER OF NW NEIGHBOR TO CURRENT NW CORNER GHOSTS
            Tgh(0, JMAXBLK+1) = Tnb(IMAXBLK-1, 2)
            nbrl%NW => nbrl%NW%next
        END DO
    END SUBROUTINE update_ghosts

    SUBROUTINE update_ghosts_debug(b)
        ! Update ghost nodes of each block using logical statements.
        ! used to debug linked lists

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: b(:)
        TYPE(NBRTYPE), POINTER :: NB
        ! temperature information pointers for ghost and neighbor nodes
        REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
        ! iteration parameters, index of neighbor
        INTEGER :: I, J, INBR, IBLK


        DO IBLK = 1, NBLK
            NB => b(iblk)%NB


            !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

            IF ( NB%N > 0 ) THEN
                ! TEMPERATURE OF CURRENT BLOCK (CONTAINS GHOST NODES)
                Tgh => b( IBLK )%mesh%T
                ! index of north neighbor
                INBR = NB%N
                ! TEMPERATURE OF NEIGHBOR BLOCK (UPDATE GHOSTS WITH THIS)
                Tnb => b( INBR )%mesh%T

                DO I = 1, IMAXBLK
```

```
!                     Tgh(I, JMAXBLK+1) = Tnb(I, 2)
                    b(iblk)%mesh%T(I, JMAXBLK+1) = b(NB%N)%mesh%T(I, 2)
                END DO
            END IF

            !south
            IF ( NB%S > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = NB%S
                Tnb => b( INBR )%mesh%T

                DO I = 1, IMAXBLK
                    ! ADD NORTH FACE OF SOUTH NEIGHBOR TO CURRENT SOUTH FACE GHOSTS
                    Tgh(I, 0) = Tnb(I, JMAXBLK-1)
                END DO
            END IF

            !EAST
            IF ( NB%E > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = NB%E
                Tnb => b( INBR )%mesh%T
                DO J = 1, JMAXBLK
                    ! ADD WEST FACE OF EAST NEIGHBOR TO CURRENT WEST FACE GHOSTS
                    Tgh(IMAXBLK+1, J) = Tnb(2, J)
                END DO
            END IF

            ! WEST FACE GHOST NODES
            IF ( NB%W > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%W
                Tnb => b( INBR )%mesh%T
                DO J = 1, JMAXBLK
                    ! ADD EAST FACE OF WEST NEIGHBOR TO CURRENT EAST FACE GHOSTS
                    Tgh(0, J) = Tnb(IMAXBLK-1, J)
                END DO
            END IF

            !!! CORNERS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

            ! NORTH EAST CORNER GHOST NODES
            IF ( NB%NE > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%NE
                Tnb => b( INBR )%mesh%T
                ! ADD SW CORNER OF NE NEIGHBOR TO CURRENT NE CORNER GHOSTS
                Tgh(IMAXBLK+1, JMAXBLK+1) = Tnb(2, 2)
            END IF

            ! SOUTH EAST CORNER GHOST NODE
            IF ( NB%SE > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%SE
                Tnb => b( INBR )%mesh%T
                ! ADD NW CORNER OF SE NEIGHBOR TO CURRENT SE CORNER GHOSTS
                Tgh(IMAXBLK+1, 0) = Tnb(2, JMAXBLK-1)
            END IF

            ! SOUTH WEST CORNER GHOST NODES
            IF ( NB%SW > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%SW
                Tnb => b( INBR )%mesh%T
                ! ADD NE CORNER OF SW NEIGHBOR TO CURRENT SW CORNER GHOSTS
                Tgh(0, 0) = Tnb(IMAXBLK-1, JMAXBLK-1)
            END IF

            ! NORTH WEST CORNER GHOST NODES
```

```
            IF ( NB%NW > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%NW
                Tnb => b( INBR )%mesh%T
                ! ADD SE CORNER OF NW NEIGHBOR TO CURRENT NW CORNER GHOSTS
                Tgh(0, JMAXBLK+1) = Tnb(IMAXBLK-1, 2)
            END IF
        END DO
    END SUBROUTINE update_ghosts_debug

    SUBROUTINE calc_cell_params(blocks)
        ! calculate areas for secondary fluxes. ! Call after reading mesh data
        ! from restart file
        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(MESHTYPE), POINTER :: m
        INTEGER :: IBLK, I, J
        ! Areas used in counter-clockwise trapezoidal integration to get
        ! x and y first-derivatives for center of each cell (Green's thm)
        REAL(KIND=8) :: Ayi_half, Axi_half, Ayj_half, Axj_half

        DO IBLK = 1, NBLK
            m => blocks(IBLK)%mesh

            DO J = 0, JMAXBLK
                DO I = 0, IMAXBLK
                    ! CALC CELL VOLUME
                        ! cross product of cell diagonals p, q
                        ! where p has x,y components px, py and q likewise.
                        ! Thus, p cross q = px*qy - qx*py
                        ! where, px = x(i+1,j+1) - x(i,j), py = y(i+1,j+1) - y(i,j)
                        ! and    qx = x(i,j+1) - x(i+1,j), qy = y(i,j+1) - y(i+1,j)
                    m%V(I,J) = ( m%x(I+1,J+1) - m%x(I,  J) ) &
                             * ( m%y(I,  J+1) - m%y(I+1,J) ) &
                             - ( m%x(I,  J+1) - m%x(I+1,J) ) &
                             * ( m%y(I+1,J+1) - m%y(I,  J) )
                END DO
            END DO

            ! CALC CELL AREAS (FLUXES) IN J-DIRECTION
            DO J = 0, JMAXBLK+1
                DO I = 0, IMAXBLK
                    m%Axj(I,J) = m%x(I+1,J) - m%x(I,J)
                    m%Ayj(I,J) = m%y(I+1,J) - m%y(I,J)
                END DO
            END DO
            ! CALC CELL AREAS (FLUXES) IN I-DIRECTION
            DO J = 0, JMAXBLK
                DO I = 0, IMAXBLK+1
                    ! CALC CELL AREAS (FLUXES)
                    m%Axi(I,J) = m%x(I,J+1) - m%x(I,J)
                    m%Ayi(I,J) = m%y(I,J+1) - m%y(I,J)
                END DO
            END DO

            ! Actual finite-volume scheme equation parameters
            DO J = 0, JMAXBLK
                DO I = 0, IMAXBLK

                    Axi_half = ( m%Axi(I+1,J) + m%Axi(I,J) ) * 0.25D0
                    Axj_half = ( m%Axj(I,J+1) + m%Axj(I,J) ) * 0.25D0
                    Ayi_half = ( m%Ayi(I+1,J) + m%Ayi(I,J) ) * 0.25D0
                    Ayj_half = ( m%Ayj(I,J+1) + m%Ayj(I,J) ) * 0.25D0

                    ! (NN = 'negative-negative', PN = 'positive-negative',
                        ! see how fluxes are summed)
                    m%xNN(I, J) = ( -Axi_half - Axj_half )
                    m%xPN(I, J) = (  Axi_half - Axj_half )
                    m%xPP(I, J) = (  Axi_half + Axj_half )
```

```
                    m%xNP(I, J) = ( -Axi_half + Axj_half )
                    m%yPP(I, J) = (  Ayi_half + Ayj_half )
                    m%yNP(I, J) = ( -Ayi_half + Ayj_half )
                    m%yNN(I, J) = ( -Ayi_half - Ayj_half )
                    m%yPN(I, J) = (  Ayi_half - Ayj_half )
                END DO
            END DO
        END DO
    END SUBROUTINE calc_cell_params

    SUBROUTINE calc_constants(blocks)
        ! Calculate constants for a given iteration loop.  This way,
        ! they don't need to be calculated within the loop at each iteration
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(MESHTYPE), POINTER :: m
        INTEGER :: IBLK, I, J
        DO IBLK = 1, NBLK
            m => blocks(IBLK)%mesh
            DO J = 0, JMAXBLK + 1
                DO I = 0, IMAXBLK + 1
                    ! CALC TIMESTEP FROM CFL
                    m%dt(I,J) = ((CFL * 0.5D0) / alpha) * m%V(I,J) ** 2 &
                                    / ( (m%xp(I+1,J) - m%xp(I,J))**2 &
                                      + (m%yp(I,J+1) - m%yp(I,J))**2 )
                    ! CALC SECONDARY VOLUMES
                    ! (for rectangular mesh, just average volumes of the 4 cells
                    !  surrounding the point)
                    m%V2nd(I,J) = ( m%V(I,  J) + m%V(I-1,  J) &
                                  + m%V(I,J-1) + m%V(I-1,J-1) ) * 0.25D0
                    ! CALC CONSTANT TERM
                    ! (this term remains constant in the equation regardless of
                    !  iteration number, so only calculate once here,
                    !  instead of in loop)
                    m%term(I,J) = m%dt(I,J) * alpha / m%V2nd(I,J)
                END DO
            END DO
        END DO
    END SUBROUTINE calc_constants

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!! SOLVER !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    SUBROUTINE calc_temp(b)
        ! Calculate first and second derivatives for finite-volume scheme
        TYPE(BLKTYPE), TARGET :: b(:)
        TYPE(MESHTYPE), POINTER :: m
        ! First partial derivatives of temperature in x and y directions
        REAL(KIND=8) :: dTdx, dTdy
        INTEGER :: IBLK, I, J

        DO IBLK = 1, NBLK
            m => b(IBLK)%mesh

            ! RESET SUMMATION
            m%Ttmp = 0.D0

            ! PREVIOUSLY SET ITERATION LIMITS TO UTILIZE GHOST NODES ONLY
                !ON INTERIOR FACES
            DO J = b(IBLK)%JMINLOC, b(IBLK)%JMAXLOC
                DO I = b(IBLK)%IMINLOC, b(IBLK)%IMAXLOC
                    ! CALC FIRST DERIVATIVES
                    dTdx = + 0.5d0 &
                                * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Ayi(I+1,J) &
                                -  ( m%T(I,  J) + m%T(I,  J+1) ) * m%Ayi(I,  J) &
                                -  ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Ayj(I,J+1) &
                                +  ( m%T(I,  J) + m%T(I+1,  J) ) * m%Ayj(I,  J) &
                                    ) / m%V(I,J)
                    dTdy = - 0.5d0 &
```

```
                                * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Axi(I+1,J) &
                                -  ( m%T(I,  J) + m%T(I,  J+1) ) * m%Axi(I,  J) &
                                -  ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Axj(I,J+1) &
                                +  ( m%T(I,  J) + m%T(I+1,  J) ) * m%Axj(I,  J) &
                                    ) / m%V(I,J)

                    ! Alternate distributive scheme second-derivative operator.
                    m%Ttmp(I+1,  J) = m%Ttmp(I+1,  J) + m%term(I+1,  J) * ( m%yNN(I,J) * dTdx + m%xPP(I,J) * dTdy )
                    m%Ttmp(I,    J) = m%Ttmp(I,    J) + m%term(I,    J) * ( m%yPN(I,J) * dTdx + m%xNP(I,J) * dTdy )
                    m%Ttmp(I,  J+1) = m%Ttmp(I,  J+1) + m%term(I,  J+1) * ( m%yPP(I,J) * dTdx + m%xNN(I,J) * dTdy )
                    m%Ttmp(I+1,J+1) = m%Ttmp(I+1,J+1) + m%term(I+1,J+1) * ( m%yNP(I,J) * dTdx + m%xPN(I,J) * dTdy )
                END DO
            END DO
            ! SAVE NEW TEMPERATURE DISTRIBUTION
                ! (preserve Ttmp for residual calculation in solver loop)

            ! Previously set bounds, add one to lower limit so as not to
            ! update BC. (dont need to for upper limit because explicit scheme)
            DO J = b(IBLK)%JMINLOC + 1, b(IBLK)%JMAXLOC
                DO I = b(IBLK)%IMINLOC + 1, b(IBLK)%IMAXLOC
                    m%T(I,J) = m%T(I,J) + m%Ttmp(I,J)
                END DO
            END DO
        END DO
    END SUBROUTINE calc_temp

END MODULE BLOCKMOD
```

Listing 3: Grids are decomposed into blocks and information pertaining to neighbors is stored using the GRIDMOD module

Appendix C: Multi-Block Solver Subroutines

```
! MAE 267
! PROJECT 3
! LOGAN HALSTROM
! 03 NOVEMBER 2015

! DESCRIPTION:  Subroutines used for solving heat conduction of steel plate.
! Subroutines utilizing linked lists are here so that linked lists do not need
! to be function inputs.
! Utilizes modules from 'modules.f90'
! CONTENTS:
! init --> Initialize the solution with dirichlet B.C.s
! solve --> Solve heat conduction equation with finite volume scheme
! output --> Save solution parameters to file

MODULE subroutines
    USE CONSTANTS
    USE BLOCKMOD
    USE IO

    IMPLICIT NONE

CONTAINS
    SUBROUTINE init_gridsystem(blocks)
        ! Initialize the solution with dirichlet B.C.s.  Save to restart files.
        TYPE(BLKTYPE)  :: blocks(:)

        ! INITIALIZE BLOCKS
        CALL init_blocks(blocks)
        ! WRITE BLOCK CONNECTIVITY FILE
        CALL write_blocks(blocks)
        ! INITIALIZE MESH
        CALL init_mesh(blocks)
        ! INITIALIZE TEMPERATURE WITH DIRICHLET B.C.
        CALL init_temp(blocks)
```

```
        ! WRITE GRID AND INITIAL TEMPERATURE TO PLOT3D RESTART FILES
        CALL plot3D(blocks)

    END SUBROUTINE init_gridsystem

    SUBROUTINE init_solution(blocks, nbrlists)
        ! Read initial conditions from restart files.  Then calculate parameters
        ! used in solution
        TYPE(BLKTYPE)  :: blocks(:)
        ! LINKED LISTS STORING NEIGHBOR INFO
        TYPE(NBRLIST) :: nbrlists

        ! READ BLOCK CONFIGURATION INFORMATION FROM CONFIG FILE
        CALL read_blocks(blocks)

        ! READ GRID AND INITIAL TEMPERATURE FROM PLOT3D RESTART FILE
        CALL readPlot3D(blocks)


        ! CALC LOCAL BOUNDARIES OF CELLS
        write(*,*) 'set local bounds'
        CALL set_block_bounds(blocks)



!         ! INITIALIZE LINKED LISTS CONTAINING BOUNDARY INFORMATION
!         write(*,*) 'make linked lists'
!         CALL init_linklists(blocks, nbrlists)
!         ! POPULATE BLOCK GHOST NODES
!         write(*,*) 'update ghosts'
!         CALL update_ghosts(blocks, nbrlists)

        CALL update_ghosts_debug(blocks)

        ! CALC AREAS FOR SECONDARY FLUXES
        write(*,*) 'calc solution stuff'
        CALL calc_cell_params(blocks)
        ! CALC CONSTANTS OF INTEGRATION
        CALL calc_constants(blocks)

    END SUBROUTINE init_solution


    SUBROUTINE solve(blocks, nbrlists, iter, res_hist)
        ! Solve heat conduction equation with finite volume scheme
        TYPE(BLKTYPE) :: blocks(:)
        ! LINKED LISTS STORING NEIGHBOR INFO
        TYPE(NBRLIST) :: nbrlists
        ! Residual history linked list
        TYPE(RESLIST), POINTER :: res_hist
        ! pointer to iterate linked list
        TYPE(RESLIST), POINTER :: hist
        ! Minimum residual criteria for iteration, actual residual
        REAL(KIND=8) :: res = 1000.D0, resloc, resmax
        ! iter in function inputs so it can be returned to main
        INTEGER :: iter, IBLK, IBLKRES

        INCLUDE "mpif.h"
        REAL(KIND=8) :: start_solve, end_solve
        WRITE(*,*) 'Starting clock for solver...'
        start_solve = MPI_Wtime()

        ! residual history
        ALLOCATE(res_hist)
        hist => res_hist

        iter_loop: DO WHILE (res >= min_res .AND. iter <= max_iter)
            ! Iterate FV solver until residual becomes less than cutoff or
            ! iteration count reaches given maximum
```

```
104
105             ! CALC NEW TEMPERATURE AT ALL POINTS
106             CALL calc_temp(blocks)
107
108             ! UPDATE GHOST NODES WITH NEW TEMPERATURE SOLUTION
109 !             CALL update_ghosts(blocks, nbrlists)
110             CALL update_ghosts_debug(blocks)
111
112             ! CALC RESIDUAL
113             resmax = 0.D0
114             DO IBLK = 1, NBLK
115                 ! Find max of each block
116                 resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Ttmp(2:IMAXBLK-1, 2:JMAXBLK-1) ) )
117                 ! keep biggest residual
118                 IF (resmax < resloc) THEN
119                     resmax = resloc
120                 END IF
121             END DO
122             ! FINAL RESIDUAL
123             res = resmax
124
125             ! SWITCH TO NEXT LINK
126                 ! (skip first entry)
127             ALLOCATE(hist%next)
128             hist => hist%next
129             NULLIFY(hist%next)
130             ! STORE RESIDUAL HISTORY
131             hist%iter = iter
132             hist%res = res
133
134
135             ! INCREMENT ITERATION COUNT
136             iter = iter + 1
137
138         END DO iter_loop
139
140         ! there was an extra increment after final iteration we need to subtract
141         iter = iter - 1
142
143         ! CACL SOLVER WALL CLOCK TIME
144         end_solve = MPI_Wtime()
145         wall_time_solve = end_solve - start_solve
146
147         IF (iter > max_iter) THEN
148           WRITE(*,*) 'DID NOT CONVERGE (NUMBER OF ITERATIONS:', iter, ')'
149         ELSE
150           WRITE(*,*) 'CONVERGED (NUMBER OF ITERATIONS:', iter, ')'
151           WRITE(*,*) '          (MAXIMUM RESIDUAL    :', res,  ')'
152         END IF
153     END SUBROUTINE solve
154
155     SUBROUTINE output(blocks, iter)
156         ! Save solution parameters to file
157         TYPE(BLKTYPE), TARGET :: blocks(:)
158         REAL(KIND=8), POINTER :: tmpT(:,:), tempTemperature(:,:)
159         REAL(KIND=8) :: resloc, resmax
160         INTEGER :: iter, I, J, IBLK, IRES
161
162 !         Temperature => mesh%T(2:IMAX-1, 2:JMAX-1)
163 !         tempTemperature => mesh%Ttmp(2:IMAX-1, 2:JMAX-1)
164
165         ! CALC RESIDUAL
166         resmax = 0.D0
167         DO IBLK = 1, NBLK
168             ! Find max of each block
169             resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Ttmp(2:IMAXBLK-1, 2:JMAXBLK-1) ) )
170             ! keep biggest residual
171             IF (resmax < resloc) THEN
172                 resmax = resloc
```

```
                IRES = IBLK
            END IF
        END DO


        ! Write final maximum residual and location of max residual
!         OPEN(UNIT = 1, FILE = casedir // "SteadySoln.dat")
!         DO i = 1, IMAX
!             DO j = 1, JMAX
!                 WRITE(1,'(F10.7, 5X, F10.7, 5X, F10.7, I5, F10.7)'), mesh%x(i,j), mesh%y(i,j), mesh%T(i,j)
!             END DO
!         END DO
!         CLOSE (1)

        ! Screen output
        tmpT => blocks(IRES)%mesh%Ttmp
        WRITE (*,*), "IMAX/JMAX", IMAX, JMAX
        WRITE (*,*), "N/M", N, M
        WRITE (*,*), "iters", iter
        WRITE (*,*), "max residual", MAXVAL(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
        WRITE (*,*), "on block id", IRES
        WRITE (*,*), "residual ij", MAXLOC(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))

        ! Write to file
!         OPEN (UNIT = 2, FILE = TRIM(casedir) // "SolnInfo.dat")
        OPEN (UNIT = 2, FILE = "SolnInfo.dat")
        WRITE (2,*), "Running a", IMAX, "by", JMAX, "grid,"
        WRITE (2,*), "With NxM:", N, "x", M, "blocks took:"
        WRITE (2,*), iter, "iterations"
        WRITE (2,*), wall_time_total, "seconds (Total CPU walltime)"
        WRITE (2,*), wall_time_solve, "seconds (Solver CPU walltime)"
!         WRITE (2,*), wall_time_iter, "seconds (Iteration CPU walltime)"
        WRITE (2,*)
        WRITE (2,*), "Found max residual of ", MAXVAL(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
        WRITE (2,*), "on block id", IRES
        WRITE (2,*), "At ij of ", MAXLOC(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
        CLOSE (2)
    END SUBROUTINE output



END MODULE subroutines
```

Listing 4: Main subroutines used for solving heat transfer on a multi-block grid

Appendix D: Multi-Block Plot3D Reader-Writer

```
! MAE 267
! PROJECT 3
! LOGAN HALSTROM
! 03 NOVEMBER 2015

! DESCRIPTION:  This module contains functions for information input and output.
! Write grid and temperature files in PLOT3D format.
! Write and read block grid configuration file

! NOTE: How to Visualize Blocks in Paraview:
    ! open unformatted PLOT3D file.
    ! Change 'Coloring' from 'Solid' to 'vtkCompositeIndex'

MODULE IO
    USE CONSTANTS
    USE BLOCKMOD
    IMPLICIT NONE

    ! VARIABLES
    INTEGER :: gridUnit  = 30   ! Unit for grid file
    INTEGER :: tempUnit = 21    ! Unit for temp file
    INTEGER :: resUnit = 23
    REAL(KIND=8) :: tRef = 1.D0          ! tRef number
    REAL(KIND=8) :: dum = 0.D0          ! dummy values

    ! LINKED LIST OF RESIDUAL HISTORY

    TYPE RESLIST
        ! Next element in linked list
        TYPE(RESLIST), POINTER :: next
        ! items in link:
        REAL(KIND=8) :: res
        INTEGER :: iter
    END TYPE RESLIST

    CONTAINS
    SUBROUTINE plot3D(blocks)
        IMPLICIT NONE

        TYPE(BLKTYPE) :: blocks(:)
        INTEGER :: IBLK, I, J

        ! FORMAT STATEMENTS
            ! I --> Integer, number following is number of sig figs
            ! E --> scientific notation,
                        ! before decimal is sig figs of exponent?
                        ! after decimal is sig figs of value
            ! number before letter is how many entries on single line
                ! before newline (number of columns)
        10     FORMAT(I10)
        20     FORMAT(10I10)
        30     FORMAT(10E20.8)

        !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! OPEN FILES
!         OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid_form.xyz',FORM='formatted')
!         OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T_form.dat',FORM='formatted')
        OPEN(UNIT=gridUnit,FILE= 'grid_form.xyz',FORM='formatted')
        OPEN(UNIT=tempUnit,FILE= 'T_form.dat',FORM='formatted')

        ! WRITE TO GRID FILE
        WRITE(gridUnit, 10) NBLK
        WRITE(gridUnit, 20) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
!         WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
        DO IBLK = 1, NBLK
            WRITE(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
```

```
                                ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO


        ! WRITE TO TEMPERATURE FILE
            ! When read in paraview, 'density' will be equivalent to temperature
        WRITE(tempUnit, 10) NBLK
        WRITE(tempUnit, 20) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
        DO IBLK = 1, NBLK

            WRITE(tempUnit, 30) tRef,dum,dum,dum
            WRITE(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO

        ! CLOSE FILES
        CLOSE(gridUnit)
        CLOSE(tempUnit)

        !!! UNFORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! OPEN FILES
!         OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid.xyz',FORM='unformatted')
!         OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T.dat',FORM='unformatted')
        OPEN(UNIT=gridUnit,FILE = 'grid.xyz',FORM='unformatted')
        OPEN(UNIT=tempUnit,FILE = 'T.dat',FORM='unformatted')

        ! WRITE TO GRID FILE (UNFORMATTED)
            ! (Paraview likes unformatted better)
        WRITE(gridUnit) NBLK
        WRITE(gridUnit) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
!         WRITE(gridUnit) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
        DO IBLK = 1, NBLK
            WRITE(gridUnit) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                            ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO


        ! WRITE TO TEMPERATURE FILE
            ! When read in paraview, 'density' will be equivalent to temperature
        WRITE(tempUnit) NBLK
        WRITE(tempUnit) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
        DO IBLK = 1, NBLK

            WRITE(tempUnit) tRef,dum,dum,dum
            WRITE(tempUnit) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                                ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO

        ! CLOSE FILES
        CLOSE(gridUnit)
        CLOSE(tempUnit)
    END SUBROUTINE plot3D

    SUBROUTINE readPlot3D(blocks)
        IMPLICIT NONE

        TYPE(BLKTYPE) :: blocks(:)
        INTEGER :: IBLK, I, J
        ! READ INFO FOR BLOCK DIMENSIONS
        INTEGER :: NBLKREAD, IMAXBLKREAD, JMAXBLKREAD

        ! FORMAT STATEMENTS
            ! I --> Integer, number following is number of sig figs
            ! E --> scientific notation,
```

```
                        ! before decimal is sig figs of exponent?
                        ! after decimal is sig figs of value
            ! number before letter is how many entries on single line
                ! before newline (number of columns)
        10     FORMAT(I10)
        20     FORMAT(10I10)
        30     FORMAT(10E20.8)

        !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! OPEN FILES
!         OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid_form.xyz',FORM='formatted')
!         OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T_form.dat',FORM='formatted')
        OPEN(UNIT=gridUnit,FILE= 'grid_form.xyz',FORM='formatted')
        OPEN(UNIT=tempUnit,FILE= 'T_form.dat',FORM='formatted')

        ! READ GRID FILE
        READ(gridUnit, 10) NBLKREAD
        READ(gridUnit, 20) ( IMAXBLKREAD, JMAXBLKREAD, IBLK=1, NBLKREAD)
!         WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
        DO IBLK = 1, NBLKREAD
            READ(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                               ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO


        ! READ TEMPERATURE FILE
            ! When read in paraview, 'density' will be equivalent to temperature
        READ(tempUnit, 10) NBLKREAD
        READ(tempUnit, 20) ( IMAXBLKREAD, JMAXBLKREAD, IBLK=1, NBLKREAD)
        DO IBLK = 1, NBLKREAD

            READ(tempUnit, 30) tRef,dum,dum,dum
            READ(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                               ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                               ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
                               ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
        END DO

        ! CLOSE FILES
        CLOSE(gridUnit)
        CLOSE(tempUnit)
    END SUBROUTINE readPlot3D

    SUBROUTINE write_res(res_hist)
        TYPE(RESLIST), POINTER :: res_hist
        ! pointer to iterate linked list
        TYPE(RESLIST), POINTER :: hist

        ! open residual file
!         OPEN(UNIT=resUnit,FILE= TRIM(casedir) // 'res_hist.dat')
        OPEN(UNIT=resUnit,FILE = 'res_hist.dat')
        ! column headers
        WRITE(resUnit,*) 'ITER      RESID'

        ! point to residual linked list
        hist => res_hist
        ! skip first link, empty from iteration loop design
        hist => hist%next
        ! write residual history to file until list ends
        DO
            IF ( .NOT. ASSOCIATED(hist) ) EXIT
            ! write iteration and residual in two columns
            WRITE(resUnit,*) hist%iter, hist%res
            hist => hist%next
        END DO

        CLOSE(resUnit)
    END SUBROUTINE write_res
```

```
206
207
208 END MODULE IO
```

Listing 5: Code for saving formatted multiblock PLOT3D solution files and reading restart files

Appendix E: Other Relevant Codes

```
1  ! MAE 267
2  ! PROJECT 3
3  ! LOGAN HALSTROM
4  ! 03 NOVEMBER 2015
5
6
7  ! DESCRIPTION:  Solve heat conduction equation for single block of steel.
8  ! To compile: mpif90 -o main -O3 modules.f90 plot3D_module.f90 subroutines.f90 main.f90
9      ! makes executable file 'main'
10     ! 'rm *.mod' afterward to clean up unneeded compiled files
11 ! To run: ./main or ./run.sh or sbatch run.sh on hpc1
12
13
14 PROGRAM heatTrans
15 !     USE CLOCK
16     USE CONSTANTS
17     USE subroutines
18     USE IO
19
20     IMPLICIT NONE
21
22     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
23     !!! INITIALIZE VARIABLES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
24     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
25
26     ! BLOCKS
27     TYPE(BLKTYPE), ALLOCATABLE :: blocks(:)
28     ! LINKED LISTS STORING NEIGHBOR INFO
29     TYPE(NBRLIST) :: nbrlists
30     ! ITERATION PARAMETERS
31     ! Residual history linked list
32     TYPE(RESLIST), POINTER :: res_hist
33     ! Maximum number of iterations
34     INTEGER :: iter = 1, IBLK
35
36     INCLUDE "mpif.h"
37     REAL(KIND=8) :: start_total, end_total
38     REAL(KIND=8) :: start_solve, end_solve
39     ! CLOCK TOTAL TIME OF RUN
40     start_total = MPI_Wtime()
41
42     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
43     !!! INITIALIZE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
44     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
45
46     ! READ INPUTS FROM FILE
47     CALL read_input()
48     ALLOCATE( blocks(NBLK) )
49     ! INIITIALIZE GRID SYSTEM
50     WRITE(*,*) 'Making mesh...'
51     CALL init_gridsystem(blocks)
52
53     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
54     !!! SOLVER !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
55     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
56
57     ! INITIALIZE SOLUTION
58     CALL init_solution(blocks, nbrlists)
```

```
    ! SOLVE
    WRITE(*,*) 'Solving heat conduction...'
    CALL solve(blocks, nbrlists, iter, res_hist)

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! SAVE RESULTS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    WRITE(*,*) 'Writing results...'
    ! SAVE SOLUTION AS PLOT3D FILES
    CALL plot3D(blocks)
    ! CALC TOTAL WALL TIME
    end_total = MPI_Wtime()
    wall_time_total = end_total - start_total

    ! SAVE RESIDUAL HISTORY
    CALL write_res(res_hist)
    ! SAVE SOLVER PERFORMANCE PARAMETERS
    CALL output(blocks, iter)

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! CLEAN UP !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    DO IBLK = 1, NBLK
        DEALLOCATE( blocks(IBLK)%mesh%xp   )
        DEALLOCATE( blocks(IBLK)%mesh%yp   )
        DEALLOCATE( blocks(IBLK)%mesh%x    )
        DEALLOCATE( blocks(IBLK)%mesh%y    )
        DEALLOCATE( blocks(IBLK)%mesh%T    )
        DEALLOCATE( blocks(IBLK)%mesh%Ttmp )
        DEALLOCATE( blocks(IBLK)%mesh%dt   )
        DEALLOCATE( blocks(IBLK)%mesh%V  )
        DEALLOCATE( blocks(IBLK)%mesh%V2nd )
        DEALLOCATE( blocks(IBLK)%mesh%term )
        DEALLOCATE( blocks(IBLK)%mesh%yPP)
        DEALLOCATE( blocks(IBLK)%mesh%yNP)
        DEALLOCATE( blocks(IBLK)%mesh%yNN)
        DEALLOCATE( blocks(IBLK)%mesh%yPN)
        DEALLOCATE( blocks(IBLK)%mesh%xNN)
        DEALLOCATE( blocks(IBLK)%mesh%xPN)
        DEALLOCATE( blocks(IBLK)%mesh%xPP)
        DEALLOCATE( blocks(IBLK)%mesh%xNP)
    END DO

    WRITE(*,*) 'Done!'

    ! MOVE OUTPUT FILE TO OUTPUT DIRECTORY
!     CALL EXECUTE_COMMAND_LINE ("mv a.out " // casedir // '.')


END PROGRAM heatTrans
```

Listing 6: Wrapper program