

MAE 267 – Project 4

Parallel, 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 rotated in the positive z-direction by $rot = 30^\circ$. 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 1.0×10^{-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 referred 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).

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

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

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

Grids were generated according to the following (Eqn 3)

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

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.

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

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

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 a 10x10 multi-block solver. Fig ?? 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 ?. 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 unforeseen 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 Performance Comparison

Appendix B: Multi-Block Grid Decomposition Code

```
1 ! MAE 267
2 ! PROJECT 3
3 ! LOGAN HALSTROM
4 ! 03 NOVEMBER 2015
5
6 ! DESCRIPTION: Modules used for solving heat conduction of steel plate.
7 ! Initialize and store constants used in all subroutines.
8
9 ! CONTENTS:
10
11 ! CONSTANTS --> Module that reads, initializes, and stores constants.
12 !   ! Math and material constants, solver parameters, block sizing
13 !   ! CONTAINS:
14
15 !   ! read_input:
16 !     ! Reads grid/block size and other simulation parameters from
17 !     ! "config.in" file. Avoids recompiling for simple input changes
18
19 ! BLOCKMOD --> Module that contains data types and functions pertaining to
20 !   ! block mesh generation and solution. Derived data types include;
21 !   ! MESHTYPE containing node information like temperature, and area,
22 !   ! NBRTYPE containing information about cell neighbors
23 !   ! LNKLIST linked list for storing similar neighbor information
24 !   ! CONTAINS:
25
26 !     ! init_blocks
27 !     ! Assign individual block global indicies, neighbor, BCs, and
28 !     ! orientation information
29
30 !     ! write_blocks
31 !     ! Write block connectivity file with neighbor and BC info
32
33 !     ! read_blocks
34 !     ! Read block connectivity file
35
36 !     ! init_mesh
37 !     ! Create xprime/yprime non-uniform grid, then rotate by angle 'rot'.
38 !     ! Allocate arrays for node parameters (i.e. temperature, cell area, etc)
39
40 !     ! init_temp
41 !     ! Initialize temperature across mesh with dirichlet BCs
42 !     ! or constant temperature BCs for DEBUG=1
43
44 !     ! set_block_bounds
45 !     ! Calculate iteration bounds for each block to avoid overwriting BCs.
46 !     ! Call after reading in mesh data from restart file
47
48 !     ! init_linklists
49 !     ! Calculate iteration bounds for each block to avoid overwriting BCs.
50 !     ! Call after reading in mesh data from restart file
51
52 !     ! update_ghosts
53 !     ! Update ghost nodes of each block based on neighbor linked lists.
54 !     ! Ghost nodes contain solution from respective block face/corner
55 !     ! neighbor for use in current block solution.
56
57 !     ! update_ghosts_debug
58 !     ! Update ghost nodes of each block using logical statements.
59 !     ! used to debug linked lists
60
61 !     ! calc_cell_params
62 !     ! calculate areas for secondary fluxes and constant terms in heat
63 !     ! transfer eqn. Call after reading mesh data from restart file
64
65 !     ! calc_constants
66 !     ! Calculate terms that are constant regardless of iteration
```

```

67      ! (time step, secondary volumes, constant term.) This way,
68      ! they don't need to be calculated within the loop at each iteration
69
70      ! calc_temp
71      ! Calculate temperature at all points in mesh, excluding BC cells.
72      ! Calculate first and second derivatives for finite-volume scheme
73
74      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
75      !!!! CONSTANTS MODULE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
76      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
77
78  MODULE CONSTANTS
79      ! Initialize constants for simulation. Set grid size.
80      IMPLICIT NONE
81      ! CFL number, for convergence (D0 is double-precision, scientific notation)
82      REAL(KIND=8), PARAMETER :: CFL = 0.95D0
83      ! Material constants (steel): thermal conductivity [W/(m*K)],
84      ! density [kg/m^3],
85      ! specific heat ratio [J/(kg*K)]
86      ! initial temperature
87      REAL(KIND=8), PARAMETER :: k = 18.8D0, rho = 8000.D0, cp = 500.D0, T0 = 3.5D0
88      ! Thermal diffusivity [m^2/s]
89      REAL(KIND=8), PARAMETER :: alpha = k / (cp * rho)
90      ! Pi, grid rotation angle (30 deg)
91      REAL(KIND=8), PARAMETER :: pi = 3.141592654D0, rot = 30.D0*pi/180.D0
92      ! ITERATION PARAMETERS
93      ! Minimum Residual
94      REAL(KIND=8) :: min_res = 0.00001D0
95      ! Maximum number of iterations
96      INTEGER :: max_iter = 1000000
97      ! CPU Wall Times
98      REAL(KIND=8) :: wall_time_total, wall_time_solve, wall_time_iter(1:5)
99      ! read square grid size, Total grid size, size of grid on each block (local)
100     INTEGER :: nx, IMAX, JMAX, IMAXBK, JMAXBK
101     ! Dimensions of block layout, Number of Blocks,
102     INTEGER :: M, N, NBLK
103     ! Block boundary condition identifiers
104     ! If block face is on North,east,south,west of main grid, identify
105     ! INTEGER :: NBND = 1, SBND = 2, EBND = 3, WBND = 4
106     INTEGER :: NBND = -1, EBND = -2, SBND = -3, WBND = -4
107     ! Output directory
108     CHARACTER(LEN=18) :: casedir
109     ! Debug mode = 1
110     INTEGER :: DEBUG
111     ! Value for constant temperature BCs for debugging
112     REAL(KIND=8), PARAMETER :: TDEBUG = T0 - T0 * 0.5
113
114  CONTAINS
115
116  SUBROUTINE read_input()
117      ! Reads grid/block size and other simulation parameters from
118      ! "config.in" file. Avoids recompiling for simple input changes
119
120      INTEGER :: I
121      CHARACTER(LEN=3) :: strNX
122      CHARACTER(LEN=1) :: strN, strM
123
124      ! READ INPUTS FROM FILE
125      ! (So I don't have to recompile each time I change an input setting)
126      ! WRITE(*,*) ''
127      ! WRITE(*,*) 'Reading input...'
128      OPEN (UNIT = 1, FILE = 'config.in')
129      DO I = 1, 3
130          ! Skip header lines
131          READ(1,*)
132      END DO
133      ! READ GRIDSIZE (4th line)
134      READ(1,*) nx
135      ! READ BLOCKS (6th and 8th line)

```

```

136     READ(1,*)
137     READ(1,*) M
138     READ(1,*)
139     READ(1,*) N
140     ! DEBUG MODE (10th line)
141     READ(1,*)
142     READ(1,*) DEBUG
143
144     ! SET GRID SIZE
145     IMAX = nx
146     JMAX = nx
147     ! CALC NUMBER OF BLOCKS
148     NBLK = M * N
149     ! SET SIZE OF EACH BLOCK (LOCAL MAXIMUM I, J)
150     IMAXBLK = 1 + (IMAX - 1) / N
151     JMAXBLK = 1 + (JMAX - 1) / M
152
153     ! OUTPUT DIRECTORIES
154     ! write integers to strings
155     WRITE( strNX, '(I3)') nx
156     IF ( N - 10 < 0 ) THEN
157         ! N is a single digit (I1)
158         WRITE( strN, '(I1)') N
159     ELSE
160         ! N is a tens digit
161         WRITE( strN, '(I2)') N
162     END IF
163     IF ( M - 10 < 0 ) THEN
164         WRITE( strM, '(I1)') M
165     ELSE
166         WRITE( strM, '(I2)') M
167     END IF
168     ! case output directory: nx_Nxm (i.e. 'Results/101_5x4')
169     casedir = 'Results/' // strNX // '_' // strN // 'x' // strM // '/'
170     ! MAKE DIRECTORIES (IF THEY DONT ALREADY EXIST)
171     CALL EXECUTE_COMMAND_LINE ('mkdir -p " ' // TRIM(casedir) )
172
173     ! OUTPUT TO SCREEN
174     WRITE(*,*) ''
175     WRITE(*,*) 'Solving Mesh of size ixj:', IMAX, 'x', JMAX
176     WRITE(*,*) 'With MxN blocks:', M, 'x', N
177     WRITE(*,*) 'Number of blocks:', NBLK
178     WRITE(*,*) 'Block size ixj:', IMAXBLK, 'x', JMAXBLK
179     IF (DEBUG == 1) THEN
180         WRITE(*,*) 'RUNNING IN DEBUG MODE'
181     END IF
182     WRITE(*,*) ''
183 END SUBROUTINE read_input
184 END MODULE CONSTANTS
185
186 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
187 !!!! BLOCK GRID MODULE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
188 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
189
190 MODULE BLOCKMOD
191     ! Initialize grid with correct number of points and rotation,
192     ! set boundary conditions, etc.
193     USE CONSTANTS
194
195     IMPLICIT NONE
196     PUBLIC
197
198     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
199     !!!! DERIVED DATA TYPES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
200     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
201
202     ! DERIVED DATA TYPE FOR GRID INFORMATION
203
204     TYPE MESHTYPE

```

```

205     ! Grid points, see coordinate rotaion equations in problem statement
206     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xp, yp, x, y
207     ! Temperature at each point, temporary variable to hold temperature sum
208     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: T, Ttmp
209     ! Iteration Parameters: timestep, cell volume, secondary cell volume,
210     ! equation constant term
211     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: dt, V, V2nd, term
212     ! Areas used in alternative scheme to get fluxes for second-derivative
213     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: Ayi, Axi, Ayj, Axj
214     ! Second-derivative weighting factors for alternative distribution scheme
215     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: yPP, yNP, yNN, yPN
216     REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xNN, xPN, xPP, xNP
217 END TYPE MESHTYPE
218
219 ! DATA TYPE FOR INFORMATION ABOUT NEIGHBORS
220
221 TYPE NBRTYPE
222     ! Information about face neighbors (north, east, south, west)
223     ! And corner neighbors (Northeast, southeast, southwest, northwest)
224     INTEGER :: N, E, S, W, NE, SE, SW, NW
225 END TYPE NBRTYPE
226
227 ! DERIVED DATA TYPE WITH INFORMATION PERTAINING TO SPECIFIC BLOCK
228
229 TYPE BLKTYPE
230     ! DER. DATA TYPE STORES LOCAL MESH INFO
231     TYPE(MESHTYPE) :: mesh
232     ! IDENTIFY FACE AND CORNER NEIGHBOR BLOCKS AND PROCESSORS
233     TYPE(NBRTYPE) :: NB, NP
234     ! BLOCK NUMBER
235     INTEGER :: ID
236     ! GLOBAL INDICIES OF MINIMUM AND MAXIMUM INDICIES OF BLOCK
237     INTEGER :: IMIN, IMAX, JMIN, JMAX
238     ! LOCAL ITERATION BOUNDS TO AVOID UPDATING BC'S + UTILIZE GHOST NODES
239     INTEGER :: IMINLOC, JMINLOC, IMAXLOC, JMAXLOC, IMINUPD, JMINUPD
240     ! BLOCK ORIENTATION
241     INTEGER :: ORIENT
242 END TYPE BLKTYPE
243
244 ! LINKED LIST: RECURSIVE POINTER THAT POINTS THE NEXT ELEMENT IN THE LIST
245
246 TYPE LNKLIST
247     ! Next element in linked list
248     TYPE(LNKLIST), POINTER :: next
249     ! Identify what linked list belongs to
250     INTEGER :: ID
251 END TYPE LNKLIST
252
253 ! Collection of linked lists for faces and corners
254
255 TYPE NBRLIST
256     TYPE(LNKLIST), POINTER :: N, E, S, W, NE, SE, SW, NW
257 END TYPE NBRLIST
258
259 CONTAINS
260
261 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
262 !!! INITIALIZE GRID AND WRITE TO FILE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
263 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
264
265 SUBROUTINE init_blocks(b)
266     ! Assign individual block global indicies, neighbor, BCs, and
267     ! orientation information
268
269     ! BLOCK DATA TYPE
270     TYPE(BLKTYPE), TARGET :: b(:)
271     ! Neighbor information pointer
272     TYPE(NBRTYPE), POINTER :: NB
273     ! COUNTER VARIABLES

```

```

274      ! IM, IN COUNT BLOCK INDICIES
275      ! (IBLK COUNTS BLOCK NUMBERS, INBR IS BLOCK NEIGHBOR INDEX)
276      INTEGER :: I, J, IBLK, INBR
277
278      ! STEP THROUGH BLOCKS, ASSIGN IDENTIFYING INFO
279      !
280      !           |           |
281      !           |   North   |
282      !       NW| (IBLK + N) |NE
283      ! (IBLK + N - 1)|           | (IBLK + N + 1)
284      ! -----
285      !           |           |
286      !       West |   Current   |   East
287      ! (IBLK - 1) |   (IBLK)   | (IBLK + 1)
288      !           |           |
289      ! -----
290      !           |           |
291      !       SW|           |SE
292      ! (IBLK - N - 1)|   South   | (IBLK - N + 1)
293      !           |   (IBLK - N) |
294      !           |           |
295
296      ! START AT BLOCK 1 (INCREMENT IN LOOP)
297      IBLK = 0
298
299      DO J = 1, M
300          DO I = 1, N
301              ! INCREMENT BLOCK NUMBER
302              IBLK = IBLK + 1
303
304              ! Neighbor information pointer
305              NB => b(IBLK)%NB
306
307              ! ASSIGN BLOCK NUMBER
308              b(IBLK)%ID = IBLK
309              ! ASSIGN GLOBAL MIN/MAX INDICIES OF LOCAL GRID
310              b(IBLK)%IMIN = 1 + (IMAXBK - 1) * (I - 1)
311              b(IBLK)%JMIN = 1 + (JMAXBK - 1) * (J - 1)
312              b(IBLK)%IMAX = b(IBLK)%IMIN + (IMAXBK - 1)
313              b(IBLK)%JMAX = b(IBLK)%JMIN + (JMAXBK - 1)
314
315              ! ASSIGN NUMBERS OF FACE AND CORNER NEIGHBOR BLOCKS
316              !if boundary face, assign bc later
317              NB%N = IBLK + N
318              NB%S = IBLK - N
319              NB%E = IBLK + 1
320              NB%W = IBLK - 1
321              NB%NE = IBLK + N + 1
322              NB%NW = IBLK + N - 1
323              NB%SW = IBLK - N - 1
324              NB%SE = IBLK - N + 1
325
326              ! Assign faces and corners on boundary of the actual
327              ! computational grid with number corresponding to which
328              ! boundary they are on.
329              ! Corners on actual corners of the computational grid are
330              ! ambiguously assigned.
331              IF ( b(IBLK)%JMAX == JMAX ) THEN
332                  ! NORTH BLOCK FACE AND CORNERS ARE ON MESH NORTH BOUNDARY
333                  ! AT ACTUAL CORNERS OF MESH, CORNERS ARE AMBIGUOUS
334                  NB%N = NBND
335                  NB%NE = NBND
336                  NB%NW = NBND
337              END IF
338              IF ( b(IBLK)%IMAX == IMAX ) THEN
339                  ! EAST BLOCK FACE IS ON MESH EAST BOUNDARY
340                  NB%E = EBND
341                  NB%NE = EBND
342                  NB%SE = EBND

```

```

343
344         END IF
345         IF ( b(IBLK)%JMIN == 1 ) THEN
346             ! SOUTH BLOCK FACE IS ON MESH SOUTH BOUNDARY
347             NB%S = SBND
348             NB%SE = SBND
349             NB%SW = SBND
350         END IF
351         IF ( b(IBLK)%IMIN == 1 ) THEN
352             ! WEST BLOCK FACE IS ON MESH WEST BOUNDARY
353             NB%W = WBND
354             NB%SW = WBND
355             NB%NW = WBND
356         END IF
357
358         ! BLOCK ORIENTATION
359         ! same for all in this project
360         b(IBLK)%ORIENT = 1
361
362     END DO
363 END DO
364
365 SUBROUTINE init_blocks
366
367 SUBROUTINE write_blocks(b)
368     ! Write block connectivity file with neighbor and BC info
369
370     ! BLOCK DATA TYPE
371     TYPE(BLKTYPE) :: b(:)
372     INTEGER :: I, BLKFILE = 99
373
374     11 format(3I5)
375     22 format(33I5)
376
377 !     OPEN (UNIT = BLKFILE , FILE = TRIM(casedir) // "blockconfig.dat", form='formatted')
378 OPEN (UNIT = BLKFILE , FILE = "blockconfig.dat", form='formatted')
379 ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
380 WRITE(BLKFILE, 11) NBLK, IMAXBK, JMAXBK
381 DO I = 1, NBLK
382     ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
383     ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
384     ! NORTH EAST SOUTH WEST
385     WRITE(BLKFILE, 22) b(I)%ID, &
386         b(I)%IMIN, b(I)%JMIN, &
387         b(I)%NB%N, &
388         b(I)%NB%NE, &
389         b(I)%NB%E, &
390         b(I)%NB%SE, &
391         b(I)%NB%S, &
392         b(I)%NB%SW, &
393         b(I)%NB%W, &
394         b(I)%NB%NW, &
395         b(I)%ORIENT
396 END DO
397 CLOSE(BLKFILE)
398
399 END SUBROUTINE write_blocks
400
401 SUBROUTINE read_blocks(b)
402     ! Read block connectivity file
403
404     ! BLOCK DATA TYPE
405     TYPE(BLKTYPE) :: b(:)
406     INTEGER :: I, BLKFILE = 99
407     ! READ INFOR FOR BLOCK DIMENSIONS
408     INTEGER :: NBLKREAD, IMAXBKREAD, JMAXBKREAD
409
410     11 format(3I5)
411     22 format(33I5)
412
413 !     OPEN (UNIT = BLKFILE , FILE = TRIM(casedir) // "blockconfig.dat", form='formatted')

```



```

412 OPEN (UNIT = BLKFILE , FILE = "blockconfig.dat", form='formatted')
413 ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
414 READ(BLKFILE, 11) NBLK, IMAXBK, JMAXBK
415 DO I = 1, NBLK
416     ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
417     ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
418     ! NORTH EAST SOUTH WEST
419     READ(BLKFILE, 22) b(I)%ID, &
420         b(I)%IMIN, b(I)%JMIN, &
421         b(I)%NB%N, &
422         b(I)%NB%NE, &
423         b(I)%NB%E, &
424         b(I)%NB%SE, &
425         b(I)%NB%S, &
426         b(I)%NB%SW, &
427         b(I)%NB%W, &
428         b(I)%NB%NW, &
429         b(I)%ORIENT
430 END DO
431 CLOSE(BLKFILE)
432 END SUBROUTINE read_blocks
433
434 SUBROUTINE init_mesh(b)
435     ! Create xprime/yprime non-uniform grid, then rotate by angle 'rot'.
436     ! Allocate arrays for node parameters (i.e. temperature, cell area, etc)
437
438     ! BLOCK DATA TYPE
439     TYPE(BLKTYPE), TARGET :: b(:)
440     TYPE(MESHTYPE), POINTER :: m
441     INTEGER :: IBLK, I, J
442
443     DO IBLK = 1, NBLK
444
445         m => b(IBLK)%mesh
446
447         ! ALLOCATE MESH INFORMATION
448         ! ADD EXTRA INDEX AT BEGINNING AND END FOR GHOST NODES
449         ALLOCATE( m%xp( 0:IMAXBK+1, 0:JMAXBK+1) )
450         ALLOCATE( m%yp( 0:IMAXBK+1, 0:JMAXBK+1) )
451         ALLOCATE( m%x( 0:IMAXBK+1, 0:JMAXBK+1) )
452         ALLOCATE( m%y( 0:IMAXBK+1, 0:JMAXBK+1) )
453         ALLOCATE( m%T( 0:IMAXBK+1, 0:JMAXBK+1) )
454         ALLOCATE( m%Ttmp(0:IMAXBK+1, 0:JMAXBK+1) )
455         ALLOCATE( m%dt( 0:IMAXBK+1, 0:JMAXBK+1) )
456         ALLOCATE( m%V2nd(0:IMAXBK+1, 0:JMAXBK+1) )
457         ALLOCATE( m%term(0:IMAXBK+1, 0:JMAXBK+1) )
458         ALLOCATE( m%Ayi( 0:IMAXBK+1, 0:JMAXBK+1) )
459         ALLOCATE( m%Axi( 0:IMAXBK+1, 0:JMAXBK+1) )
460         ALLOCATE( m%Ayj( 0:IMAXBK+1, 0:JMAXBK+1) )
461         ALLOCATE( m%Axj( 0:IMAXBK+1, 0:JMAXBK+1) )
462         ALLOCATE( m%V( 0:IMAXBK, 0:JMAXBK ) )
463         ALLOCATE( m%yPP( 0:IMAXBK, 0:JMAXBK ) )
464         ALLOCATE( m%yNP( 0:IMAXBK, 0:JMAXBK ) )
465         ALLOCATE( m%yNN( 0:IMAXBK, 0:JMAXBK ) )
466         ALLOCATE( m%yPN( 0:IMAXBK, 0:JMAXBK ) )
467         ALLOCATE( m%xNN( 0:IMAXBK, 0:JMAXBK ) )
468         ALLOCATE( m%xPN( 0:IMAXBK, 0:JMAXBK ) )
469         ALLOCATE( m%xPP( 0:IMAXBK, 0:JMAXBK ) )
470         ALLOCATE( m%xNP( 0:IMAXBK, 0:JMAXBK ) )
471
472         ! STEP THROUGH LOCAL INDICIES OF EACH BLOCK
473         DO J = 0, JMAXBK+1
474             DO I = 0, IMAXBK+1
475                 ! MAKE SQUARE GRID
476                 ! CONVERT FROM LOCAL TO GLOBAL INDEX:
477                 ! Iglobal = Block%IMIN + (Ilocal - 1)
478                 m%xp(I, J) = COS( 0.5D0 * PI * DFLOAT(IMAX - ( b(IBLK)%IMIN + I - 1) ) / DFLOAT(IMAX - 1) )
479                 m%yp(I, J) = COS( 0.5D0 * PI * DFLOAT(JMAX - ( b(IBLK)%JMIN + J - 1) ) / DFLOAT(JMAX - 1) )
480                 ! ROTATE GRID

```

```

481         m%x(I, J) = m%xp(I, J) * COS(rot) + (1.D0 - m%yp(I, J) ) * SIN(rot)
482         m%y(I, J) = m%yp(I, J) * COS(rot) + (          m%xp(I, J) ) * SIN(rot)
483     END DO
484 END DO
485 END DO
486 END SUBROUTINE init_mesh
487
488 SUBROUTINE init_temp(blocks)
489     ! Initialize temperature across mesh with dirichlet BCs
490     ! or constant temperature BCs for DEBUG=1
491
492     ! BLOCK DATA TYPE
493     TYPE(BLKTYPE), TARGET :: blocks(:)
494     TYPE(BLKTYPE), POINTER :: b
495     TYPE(MESHTYPE), POINTER :: m
496     TYPE(NBRTYPE), POINTER :: NB
497     INTEGER :: IBLK, I, J
498
499     DO IBLK = 1, NBLK
500         b => blocks(IBLK)
501         m => blocks(IBLK)%mesh
502         NB => blocks(IBLK)%NB
503         ! FIRST, INITIALIZE ALL POINT TO INITIAL TEMPERATURE (T0)
504         m%T(0:IMAXBK+1, 0:JMAXBK+1) = T0
505         ! THEN, INITIALIZE BOUNDARIES DIRICHLET B.C.
506         IF (DEBUG /= 1) THEN
507
508             ! DIRICHLET B.C.
509             ! face on north boundary
510             IF (NB%N == NBND) THEN
511                 DO I = 1, IMAXBK
512                     m%T(I, JMAXBK) = 5.D0 * (SIN(PI * m%xp(I, JMAXBK)) + 1.D0)
513                 END DO
514             END IF
515             IF (NB%S == SBND) THEN
516                 DO I = 1, IMAXBK
517                     m%T(I, 1) = ABS(COS(PI * m%xp(I, 1))) + 1.D0
518                 END DO
519             END IF
520             IF (NB%E == EBND) THEN
521                 DO J = 1, JMAXBK
522                     m%T(IMAXBK, J) = 3.D0 * m%yp(IMAXBK, J) + 2.D0
523                 END DO
524             END IF
525             IF (NB%W == WBND) THEN
526                 DO J = 1, JMAXBK
527                     m%T(1, J) = 3.D0 * m%yp(1, J) + 2.D0
528                 END DO
529             END IF
530
531         ELSE
532
533             ! DEBUG BCS
534             IF (NB%N < 0) THEN
535                 DO I = 1, IMAXBK
536                     m%T(I, JMAXBK) = TDEBUG
537                 END DO
538             END IF
539             IF (NB%S < 0) THEN
540                 DO I = 1, IMAXBK
541                     m%T(I, 1) = TDEBUG
542                 END DO
543             END IF
544             IF (NB%E < 0) THEN
545                 DO J = 1, JMAXBK
546                     m%T(IMAXBK, J) = TDEBUG
547                 END DO
548             END IF
549             IF (NB%W < 0) THEN

```

```

550         DO J = 1, JMAXBLK
551             m%T(1, J) = TDEBUG
552         END DO
553     END IF
554 END IF
555 END DO
556 END SUBROUTINE init_temp
557
558 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
559 !!! INITIALIZE SOLUTION AFTER RESTART FILE READ IN !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
560 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
561
562 SUBROUTINE set_block_bounds(blocks)
563     ! Calculate iteration bounds for each block to avoid overwriting BCs.
564     ! Call after reading in mesh data from restart file
565
566     TYPE(BLKTYPE), TARGET :: blocks(:)
567     TYPE(BLKTYPE), POINTER :: b
568     TYPE(NBRTYPE), POINTER :: NB
569     INTEGER :: IBLK, I, J
570
571     DO IBLK = 1, NBLK
572         b => blocks(IBLK)
573         NB => b%NB
574
575         ! Set iteration bounds of each block to preserve BCs
576         ! south and west boundaries:
577             ! interior: iminloc, jminloc = 0 (use ghost)
578             ! boundary: iminloc, jminloc = 2 (1st index is BC)
579         ! north and east boundaries:
580             ! interior: imaxloc, jmaxloc = maxblk (use ghost)
581             ! boundary: imaxloc, jmaxloc = maxblk-1 (max index is BC)
582
583         ! NORTH
584         IF (NB%N > 0) THEN
585             ! Interior faces have positive ID neighbors
586             b%JMAXLOC = JMAXBLK
587         ELSE
588             ! At North Boundary
589             b%JMAXLOC = JMAXBLK - 1
590         END IF
591
592         ! EAST
593         IF (NB%E > 0) THEN
594             ! Interior
595             b%IMAXLOC = IMAXBLK
596         ELSE
597             ! At east Boundary
598             b%IMAXLOC = IMAXBLK - 1
599         END IF
600
601         ! SOUTH
602         IF (NB%S > 0) THEN
603             ! Interior
604             b%JMINLOC = 0
605         ELSE
606             ! At south Boundary
607             b%JMINLOC = 1
608             ! boundary for updating temperature (dont update BC)
609             b%JMINUPD = 2
610         END IF
611
612         ! WEST
613         IF (NB%W > 0) THEN
614             ! Interior
615             b%IMINLOC = 0
616         ELSE
617             ! At west Boundary
618             b%IMINLOC = 1

```

```

619         b%IMINUPD = 2
620     END IF
621 END DO
622 END SUBROUTINE set_block_bounds
623
624 SUBROUTINE init_linklists(blocks, nbrlists)
625     ! Create linked lists governing block boundary communication.
626     ! Separate list for each neighbor type so we can avoid logic when
627     ! updating ghost nodes.
628
629     ! BLOCK DATA TYPE
630     TYPE(BLKTYPE), TARGET :: blocks(:)
631     ! Neighbor information pointer
632     TYPE(NBRTYPE), POINTER :: NB
633     ! Linked lists of neighbor communication instructions
634     TYPE(NBRLIST) :: nbrlists
635     TYPE(NBRLIST) :: nbrl
636     INTEGER :: IBLK
637
638     ! INITIALIZE LINKED LISTS (HPC1 REQUIRES THIS)
639     NULLIFY(nbrlists%N)
640     NULLIFY(nbrlists%S)
641     NULLIFY(nbrlists%E)
642     NULLIFY(nbrlists%W)
643     NULLIFY(nbrlists%NW)
644     NULLIFY(nbrlists%NE)
645     NULLIFY(nbrlists%SE)
646     NULLIFY(nbrlists%SW)
647
648     DO IBLK = 1, NBLK
649         NB => blocks(IBLK)%NB
650
651         ! NORTH
652         ! If block north face is internal, add it to appropriate linked list
653         ! for north internal faces.
654         IF (NB%N > 0) THEN
655             IF ( .NOT. ASSOCIATED(nbrlists%N) ) THEN
656                 ! Allocate linked list if it hasnt been accessed yet
657                 ALLOCATE(nbrlists%N)
658                 ! Pointer linked list that will help iterate through the
659                 ! primary list in this loop
660                 nbrl%N => nbrlists%N
661             ELSE
662                 ! linked list already allocated (started). Allocate next
663                 ! link as assign current block to it
664                 ALLOCATE(nbrl%N%next)
665                 nbrl%N => nbrl%N%next
666             END IF
667
668             ! associate this linked list entry with the current block
669             nbrl%N%ID = IBLK
670             ! break link to pre-existing pointer target. We will
671             ! allocated this target later as the next item in the linked list
672             NULLIFY(nbrl%N%next)
673         END IF
674
675         ! SOUTH
676         IF (NB%S > 0) THEN
677             IF ( .NOT. ASSOCIATED(nbrlists%S) ) THEN
678                 ALLOCATE(nbrlists%S)
679                 nbrl%S => nbrlists%S
680             ELSE
681                 ALLOCATE(nbrl%S%next)
682                 nbrl%S => nbrl%S%next
683             END IF
684             nbrl%S%ID = IBLK
685             NULLIFY(nbrl%S%next)
686         END IF
687

```

```

688 ! EAST
689 IF (NB%E > 0) THEN
690     IF ( .NOT. ASSOCIATED(nbrlists%E) ) THEN
691         ALLOCATE(nbrlists%E)
692         nbrl%E => nbrlists%E
693     ELSE
694         ALLOCATE(nbrl%E%next)
695         nbrl%E => nbrl%E%next
696     END IF
697     nbrl%E%ID = IBLK
698     NULLIFY(nbrl%E%next)
699 END IF
700
701 ! WEST
702 IF (NB%W > 0) THEN
703     IF ( .NOT. ASSOCIATED(nbrlists%W) ) THEN
704         ALLOCATE(nbrlists%W)
705         nbrl%W => nbrlists%W
706     ELSE
707         ALLOCATE(nbrl%W%next)
708         nbrl%W => nbrl%W%next
709     END IF
710     nbrl%W%ID = IBLK
711     NULLIFY(nbrl%W%next)
712 END IF
713
714 ! NORTH EAST
715 IF (NB%NE > 0) THEN
716     IF ( .NOT. ASSOCIATED(nbrlists%NE) ) THEN
717         ALLOCATE(nbrlists%NE)
718         nbrl%NE => nbrlists%NE
719     ELSE
720         ALLOCATE(nbrl%NE%next)
721         nbrl%NE => nbrl%NE%next
722     END IF
723     nbrl%NE%ID = IBLK
724     NULLIFY(nbrl%NE%next)
725 END IF
726
727 ! SOUTH EAST
728 IF (NB%SE > 0) THEN
729     IF ( .NOT. ASSOCIATED(nbrlists%SE) ) THEN
730         ALLOCATE(nbrlists%SE)
731         nbrl%SE => nbrlists%SE
732     ELSE
733         ALLOCATE(nbrl%SE%next)
734         nbrl%SE => nbrl%SE%next
735     END IF
736     nbrl%SE%ID = IBLK
737     NULLIFY(nbrl%SE%next)
738 END IF
739
740 ! SOUTH WEST
741 IF (NB%SW > 0) THEN
742     IF ( .NOT. ASSOCIATED(nbrlists%SW) ) THEN
743         ALLOCATE(nbrlists%SW)
744         nbrl%SW => nbrlists%SW
745     ELSE
746         ALLOCATE(nbrl%SW%next)
747         nbrl%SW => nbrl%SW%next
748     END IF
749     nbrl%SW%ID = IBLK
750     NULLIFY(nbrl%SW%next)
751 END IF
752
753 ! NORTH WEST
754 IF (NB%NW > 0) THEN
755     IF ( .NOT. ASSOCIATED(nbrlists%NW) ) THEN
756         ALLOCATE(nbrlists%NW)

```

```

757         nbrl%NW => nbrlists%NW
758     ELSE
759         ALLOCATE(nbrl%NW%next)
760         nbrl%NW => nbrl%NW%next
761     END IF
762     nbrl%NW%ID = IBLK
763     NULLIFY(nbrl%NW%next)
764 END IF
765 END DO
766 END SUBROUTINE init_linklists
767
768 SUBROUTINE update_ghosts(b, nbrlists)
769     ! Update ghost nodes of each block based on neighbor linked lists.
770     ! Ghost nodes contain solution from respective block face/corner
771     ! neighbor for use in current block solution.
772
773     ! BLOCK DATA TYPE
774     TYPE(BLKTYPE), TARGET :: b(:)
775     ! temperature information pointers for ghost and neighbor nodes
776     REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
777     ! Linked lists of neighbor communication instructions
778     TYPE(NBRLIST) :: nbrlists
779     TYPE(NBRLIST) :: nbrl
780     ! iteration parameters, index of neighbor
781     INTEGER :: I, J, INBR
782
783     !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
784
785     ! NORTH FACE GHOST NODES
786     nbrl%N => nbrlists%N
787     ! Step through linked list of north faces with ghosts until end of list
788     DO
789         ! If next link in list doesnt exist (end of list), stop loop
790         IF ( .NOT. ASSOCIATED(nbrl%N) ) EXIT
791
792         ! Otherwise, assign neighbor values to all ghost nodes:
793
794         ! TEMPERATURE OF CURRENT BLOCK (CONTAINS GHOST NODES)
795         ! (identified by linked list id)
796         Tgh => b( nbrl%N%ID )%mesh%T
797
798         ! index of north neighbor
799         INBR = b( nbrl%N%ID )%NB%N
800         ! TEMPERATURE OF NEIGHBOR BLOCK (UPDATE GHOSTS WITH THIS)
801         Tnb => b( INBR )%mesh%T
802
803         DO I = 1, IMAXBLK
804             ! NORTH FACE GHOST NODE TEMPERATURE IS EQUAL TO TEMPERATURE OF
805             ! SECOND-FROM-SOUTH FACE OF NORTH NEIGHBOR
806             ! (Remember face nodes are shared between blocks)
807             Tgh(I, JMAXBLK+1) = Tnb(I, 2)
808         END DO
809         ! switch pointer to next link in list
810         nbrl%N => nbrl%N%next
811     END DO
812
813     ! SOUTH FACE GHOST NODES
814     nbrl%S => nbrlists%S
815     DO
816         IF ( .NOT. ASSOCIATED(nbrl%S) ) EXIT
817         Tgh => b( nbrl%S%ID )%mesh%T
818         INBR = b( nbrl%S%ID )%NB%S
819         Tnb => b( INBR )%mesh%T
820
821         DO I = 1, IMAXBLK
822             ! ADD NORTH FACE OF SOUTH NEIGHBOR TO CURRENT SOUTH FACE GHOSTS
823             Tgh(I, 0) = Tnb(I, JMAXBLK-1)
824         END DO
825         nbrl%S => nbrl%S%next

```

```

826 END DO
827
828 ! EAST FACE GHOST NODES
829 nbrl%E => nbrlists%E
830 DO
831   IF ( .NOT. ASSOCIATED(nbrl%E) ) EXIT
832   Tgh => b( nbrl%E%ID )%mesh%T
833   INBR = b( nbrl%E%ID )%NB%E
834   Tnb => b( INBR )%mesh%T
835
836   DO J = 1, JMAXBLK
837     ! ADD WEST FACE OF EAST NEIGHBOR TO CURRENT WEST FACE GHOSTS
838     Tgh(IMAXBLK+1, J) = Tnb(2, J)
839   END DO
840   nbrl%E => nbrl%E%next
841 END DO
842
843 ! WEST FACE GHOST NODES
844 nbrl%W => nbrlists%W
845 DO
846   IF ( .NOT. ASSOCIATED(nbrl%W) ) EXIT
847   Tgh => b( nbrl%W%ID )%mesh%T
848   INBR = b( nbrl%W%ID )%NB%W
849   Tnb => b( INBR )%mesh%T
850
851   DO J = 1, JMAXBLK
852     ! ADD EAST FACE OF WEST NEIGHBOR TO CURRENT EAST FACE GHOSTS
853     Tgh(0, J) = Tnb(IMAXBLK-1, J)
854   END DO
855   nbrl%W => nbrl%W%next
856 END DO
857
858 !!! CORNERS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
859
860 ! NORTH EAST CORNER GHOST NODES
861 nbrl%NE => nbrlists%NE
862 DO
863   IF ( .NOT. ASSOCIATED(nbrl%NE) ) EXIT
864   Tgh => b( nbrl%NE%ID )%mesh%T
865   INBR = b( nbrl%NE%ID )%NB%NE
866   Tnb => b( INBR )%mesh%T
867   ! ADD SW CORNER OF NE NEIGHBOR TO CURRENT NE CORNER GHOSTS
868   Tgh(IMAXBLK+1, JMAXBLK+1) = Tnb(2, 2)
869   nbrl%NE => nbrl%NE%next
870 END DO
871
872 ! SOUTH EAST CORNER GHOST NODES
873 nbrl%SE => nbrlists%SE
874 DO
875   IF ( .NOT. ASSOCIATED(nbrl%SE) ) EXIT
876   Tgh => b( nbrl%SE%ID )%mesh%T
877   INBR = b( nbrl%SE%ID )%NB%SE
878   Tnb => b( INBR )%mesh%T
879   ! ADD NW CORNER OF SE NEIGHBOR TO CURRENT SE CORNER GHOSTS
880   Tgh(IMAXBLK+1, 0) = Tnb(2, JMAXBLK-1)
881   nbrl%SE => nbrl%SE%next
882 END DO
883
884 ! SOUTH WEST CORNER GHOST NODES
885 nbrl%SW => nbrlists%SW
886 DO
887   IF ( .NOT. ASSOCIATED(nbrl%SW) ) EXIT
888   Tgh => b( nbrl%SW%ID )%mesh%T
889   INBR = b( nbrl%SW%ID )%NB%SW
890   Tnb => b( INBR )%mesh%T
891   ! ADD NE CORNER OF SW NEIGHBOR TO CURRENT SW CORNER GHOSTS
892   Tgh(0, 0) = Tnb(IMAXBLK-1, JMAXBLK-1)
893   nbrl%SW => nbrl%SW%next
894 END DO

```

```

895
896 ! NORTH WEST CORNER GHOST NODES
897 nbrl%NW => nbrlists%NW
898 DO
899     IF ( .NOT. ASSOCIATED(nbrl%NW) ) EXIT
900     Tgh => b( nbrl%NW%ID )%mesh%T
901     INBR = b( nbrl%NW%ID )%NB%NW
902     Tnb => b( INBR )%mesh%T
903     ! ADD SE CORNER OF NW NEIGHBOR TO CURRENT NW CORNER GHOSTS
904     Tgh(0, JMAXBLK+1) = Tnb(IMAXBLK-1, 2)
905     nbrl%NW => nbrl%NW%next
906 END DO
907 END SUBROUTINE update_ghosts
908
909 SUBROUTINE update_ghosts_debug(b)
910     ! Update ghost nodes of each block using logical statements.
911     ! used to debug linked lists
912
913     ! BLOCK DATA TYPE
914     TYPE(BLKTYPE), TARGET :: b(:)
915     TYPE(NBRTYPE), POINTER :: NB
916     ! temperature information pointers for ghost and neighbor nodes
917     REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
918     ! iteration parameters, index of neighbor
919     INTEGER :: I, J, INBR, IBLK
920
921 DO IBLK = 1, NBLK
922     NB => b(iblk)%NB
923
924
925
926     !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
927
928     IF ( NB%N > 0 ) THEN
929         ! TEMPERATURE OF CURRENT BLOCK (CONTAINS GHOST NODES)
930         Tgh => b( IBLK )%mesh%T
931         ! index of north neighbor
932         INBR = NB%N
933         ! TEMPERATURE OF NEIGHBOR BLOCK (UPDATE GHOSTS WITH THIS)
934         Tnb => b( INBR )%mesh%T
935
936         DO I = 1, IMAXBLK
937             !
938             Tgh(I, JMAXBLK+1) = Tnb(I, 2)
939             b(iblk)%mesh%T(I, JMAXBLK+1) = b(NB%N)%mesh%T(I, 2)
940         END DO
941     END IF
942
943     !south
944     IF ( NB%S > 0 ) THEN
945         Tgh => b( IBLK )%mesh%T
946         INBR = NB%S
947         Tnb => b( INBR )%mesh%T
948
949         DO I = 1, IMAXBLK
950             ! ADD NORTH FACE OF SOUTH NEIGHBOR TO CURRENT SOUTH FACE GHOSTS
951             Tgh(I, 0) = Tnb(I, JMAXBLK-1)
952         END DO
953     END IF
954
955     !EAST
956     IF ( NB%E > 0 ) THEN
957         Tgh => b( IBLK )%mesh%T
958         INBR = NB%E
959         Tnb => b( INBR )%mesh%T
960         DO J = 1, JMAXBLK
961             ! ADD WEST FACE OF EAST NEIGHBOR TO CURRENT WEST FACE GHOSTS
962             Tgh(IMAXBLK+1, J) = Tnb(2, J)
963         END DO
964     END IF
965

```



```

964      ! WEST FACE GHOST NODES
965      IF ( NB%W > 0 ) THEN
966          Tgh => b( IBLK )%mesh%T
967          INBR = b( IBLK )%NB%W
968          Tnb => b( INBR )%mesh%T
969          DO J = 1, JMAXBLK
970              ! ADD EAST FACE OF WEST NEIGHBOR TO CURRENT EAST FACE GHOSTS
971              Tgh(0, J) = Tnb(IMAXBLK-1, J)
972          END DO
973      END IF
974
975      !!! CORNERS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
976
977      ! NORTH EAST CORNER GHOST NODES
978      IF ( NB%NE > 0 ) THEN
979          Tgh => b( IBLK )%mesh%T
980          INBR = b( IBLK )%NB%NE
981          Tnb => b( INBR )%mesh%T
982          ! ADD SW CORNER OF NE NEIGHBOR TO CURRENT NE CORNER GHOSTS
983          Tgh(IMAXBLK+1, JMAXBLK+1) = Tnb(2, 2)
984      END IF
985
986      ! SOUTH EAST CORNER GHOST NODE
987      IF ( NB%SE > 0 ) THEN
988          Tgh => b( IBLK )%mesh%T
989          INBR = b( IBLK )%NB%SE
990          Tnb => b( INBR )%mesh%T
991          ! ADD NW CORNER OF SE NEIGHBOR TO CURRENT SE CORNER GHOSTS
992          Tgh(IMAXBLK+1, 0) = Tnb(2, JMAXBLK-1)
993      END IF
994
995      ! SOUTH WEST CORNER GHOST NODES
996      IF ( NB%SW > 0 ) THEN
997          Tgh => b( IBLK )%mesh%T
998          INBR = b( IBLK )%NB%SW
999          Tnb => b( INBR )%mesh%T
1000          ! ADD NE CORNER OF SW NEIGHBOR TO CURRENT SW CORNER GHOSTS
1001          Tgh(0, 0) = Tnb(IMAXBLK-1, JMAXBLK-1)
1002      END IF
1003
1004      ! NORTH WEST CORNER GHOST NODES
1005      IF ( NB%NW > 0 ) THEN
1006          Tgh => b( IBLK )%mesh%T
1007          INBR = b( IBLK )%NB%NW
1008          Tnb => b( INBR )%mesh%T
1009          ! ADD SE CORNER OF NW NEIGHBOR TO CURRENT NW CORNER GHOSTS
1010          Tgh(0, JMAXBLK+1) = Tnb(IMAXBLK-1, 2)
1011      END IF
1012  END DO
1013
1014  END SUBROUTINE update_ghosts_debug
1015
1016  SUBROUTINE calc_cell_params(blocks)
1017      ! calculate areas for secondary fluxes and constant terms in heat
1018      ! transfer eqn. Call after reading mesh data from restart file
1019
1020      ! BLOCK DATA TYPE
1021      TYPE(BLKTYPE), TARGET :: blocks(:)
1022      TYPE(MESHTYPE), POINTER :: m
1023      INTEGER :: IBLK, I, J
1024      ! Areas used in counter-clockwise trapezoidal integration to get
1025      ! x and y first-derivatives for center of each cell (Green's thm)
1026      REAL(KIND=8) :: Ayi_half, Axi_half, Ayj_half, Axj_half
1027
1028      DO IBLK = 1, NBLK
1029          m => blocks(IBLK)%mesh
1030
1031          DO J = 0, JMAXBLK
1032              DO I = 0, IMAXBLK

```

```

1033         ! CALC CELL VOLUME
1034         ! cross product of cell diagonals p, q
1035         ! where p has x,y components px, py and q likewise.
1036         ! Thus, p cross q = px*qy - qx*py
1037         ! where, px = x(i+1,j+1) - x(i,j), py = y(i+1,j+1) - y(i,j)
1038         ! and    qx = x(i,j+1) - x(i+1,j), qy = y(i,j+1) - y(i+1,j)
1039         m%V(I,J) = ( m%x(I+1,J+1) - m%x(I, J) ) &
1040         * ( m%y(I, J+1) - m%y(I+1,J) ) &
1041         - ( m%x(I, J+1) - m%x(I+1,J) ) &
1042         * ( m%y(I+1,J+1) - m%y(I, J) )
1043     END DO
1044 END DO
1045
1046 ! CALC CELL AREAS (FLUXES) IN J-DIRECTION
1047 DO J = 0, JMAXBLK+1
1048     DO I = 0, IMAXBLK
1049         m%Axi(I,J) = m%x(I+1,J) - m%x(I,J)
1050         m%Ayj(I,J) = m%y(I+1,J) - m%y(I,J)
1051     END DO
1052 END DO
1053 ! CALC CELL AREAS (FLUXES) IN I-DIRECTION
1054 DO J = 0, JMAXBLK
1055     DO I = 0, IMAXBLK+1
1056         ! CALC CELL AREAS (FLUXES)
1057         m%Axi(I,J) = m%x(I,J+1) - m%x(I,J)
1058         m%Ayi(I,J) = m%y(I,J+1) - m%y(I,J)
1059     END DO
1060 END DO
1061
1062 ! Actual finite-volume scheme equation parameters
1063 DO J = 0, JMAXBLK
1064     DO I = 0, IMAXBLK
1065
1066         Axi_half = ( m%Axi(I+1,J) + m%Axi(I,J) ) * 0.25D0
1067         Axj_half = ( m%Axi(I,J+1) + m%Axi(I,J) ) * 0.25D0
1068         Ayi_half = ( m%Ayi(I+1,J) + m%Ayi(I,J) ) * 0.25D0
1069         Ayj_half = ( m%Ayi(I,J+1) + m%Ayi(I,J) ) * 0.25D0
1070
1071         ! (NN = 'negative-negative', PN = 'positive-negative',
1072         ! see how fluxes are summed)
1073         m%xNN(I, J) = ( -Axi_half - Axj_half )
1074         m%xPN(I, J) = ( Axi_half - Axj_half )
1075         m%xPP(I, J) = ( Axi_half + Axj_half )
1076         m%xNP(I, J) = ( -Axi_half + Axj_half )
1077         m%yPP(I, J) = ( Ayi_half + Ayj_half )
1078         m%yNP(I, J) = ( -Ayi_half + Ayj_half )
1079         m%yNN(I, J) = ( -Ayi_half - Ayj_half )
1080         m%yPN(I, J) = ( Ayi_half - Ayj_half )
1081     END DO
1082 END DO
1083 END DO
1084 END SUBROUTINE calc_cell_params
1085
1086 SUBROUTINE calc_constants(blocks)
1087     ! Calculate terms that are constant regardless of iteration
1088     !(time step, secondary volumes, constant term.) This way,
1089     ! they don't need to be calculated within the loop at each iteration
1090
1091     TYPE(BLKTYPE), TARGET :: blocks(:)
1092     TYPE(MESHTYPE), POINTER :: m
1093     INTEGER :: IBLK, I, J
1094     DO IBLK = 1, NBLK
1095         m => blocks(IBLK)%mesh
1096         DO J = 0, JMAXBLK + 1
1097             DO I = 0, IMAXBLK + 1
1098                 ! CALC TIMESTEP FROM CFL
1099                 m%dt(I,J) = ((CFL * 0.5D0) / alpha) * m%V(I,J) ** 2 &
1100                 / ( (m%xp(I+1,J) - m%xp(I,J))**2 &
1101                 + (m%yp(I,J+1) - m%yp(I,J))**2 )

```

```

1102         ! CALC SECONDARY VOLUMES
1103         ! (for rectangular mesh, just average volumes of the 4 cells
1104         ! surrounding the point)
1105         m%V2nd(I,J) = ( m%V(I, J) + m%V(I-1, J) &
1106                     + m%V(I,J-1) + m%V(I-1,J-1) ) * 0.25D0
1107         ! CALC CONSTANT TERM
1108         ! (this term remains constant in the equation regardless of
1109         ! iteration number, so only calculate once here,
1110         ! instead of in loop)
1111         m%term(I,J) = m%dt(I,J) * alpha / m%V2nd(I,J)
1112     END DO
1113 END DO
1114 END DO
1115 END SUBROUTINE calc_constants
1116
1117 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1118 !!!! SOLVER !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1119 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1120
1121 SUBROUTINE calc_temp(b)
1122     ! Calculate temperature at all points in mesh, excluding BC cells.
1123     ! Calculate first and second derivatives for finite-volume scheme
1124
1125     TYPE(BLKTYPE), TARGET :: b(:)
1126     TYPE(MESHTYPE), POINTER :: m
1127     ! First partial derivatives of temperature in x and y directions
1128     REAL(KIND=8) :: dTdx, dTdy
1129     INTEGER :: IBLK, I, J
1130
1131     DO IBLK = 1, NBLK
1132         m => b(IBLK)%mesh
1133
1134         ! RESET SUMMATION
1135         m%Ttmp = 0.D0
1136
1137         ! PREVIOUSLY SET ITERATION LIMITS TO UTILIZE GHOST NODES ONLY
1138         !ON INTERIOR FACES
1139         DO J = b(IBLK)%JMINLOC, b(IBLK)%JMAXLOC
1140             DO I = b(IBLK)%IMINLOC, b(IBLK)%IMAXLOC
1141                 ! CALC FIRST DERIVATIVES
1142                 dTdx = + 0.5d0 &
1143                     * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Ayi(I+1,J) &
1144                     - ( m%T(I, J) + m%T(I, J+1) ) * m%Ayi(I, J) &
1145                     - ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Ayj(I,J+1) &
1146                     + ( m%T(I, J) + m%T(I+1, J) ) * m%Ayj(I, J) &
1147                     ) / m%V(I,J)
1148                 dTdy = - 0.5d0 &
1149                     * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Axi(I+1,J) &
1150                     - ( m%T(I, J) + m%T(I, J+1) ) * m%Axi(I, J) &
1151                     - ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Axj(I,J+1) &
1152                     + ( m%T(I, J) + m%T(I+1, J) ) * m%Axj(I, J) &
1153                     ) / m%V(I,J)
1154
1155                 ! Alternate distributive scheme second-derivative operator.
1156                 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 )
1157                 m%Ttmp(I, J) = m%Ttmp(I, J) + m%term(I, J) * ( m%yPN(I,J) * dTdx + m%xNP(I,J) * dTdy )
1158                 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 )
1159                 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 )
1160             END DO
1161         END DO
1162         ! SAVE NEW TEMPERATURE DISTRIBUTION
1163         ! (preserve Ttmp for residual calculation in solver loop)
1164
1165         ! Previously set bounds, add one to lower limit so as not to
1166         ! update BC. (dont need to for upper limit because explicit scheme)
1167         DO J = b(IBLK)%JMINLOC + 1, b(IBLK)%JMAXLOC
1168             DO I = b(IBLK)%IMINLOC + 1, b(IBLK)%IMAXLOC
1169                 m%T(I,J) = m%T(I,J) + m%Ttmp(I,J)
1170             END DO

```

```

1171         END DO
1172     END DO
1173     END SUBROUTINE calc_temp
1174
1175 END MODULE BLOCKMOD

```

Listing 1: Grids are decomposed into blocks and information pertaining to neighbors is stored using the GRIDMOD module

Appendix C: Multi-Block Solver Subroutines

```

1  ! MAE 267
2  ! PROJECT 3
3  ! LOGAN HALSTROM
4  ! 03 NOVEMBER 2015
5
6  ! DESCRIPTION: Subroutines used for solving heat conduction of steel plate.
7  ! Subroutines utilizing linked lists are here so that linked lists do not need
8  ! to be function inputs.
9  ! Utilizes modules from 'modules.f90'
10
11 ! CONTENTS:
12   ! init_gridsystem
13   ! Initialize the solution with dirichlet B.C.s. Save to restart files.
14
15   ! init_solution
16   ! Read initial conditions from restart files. Then calculate parameters
17   ! used in solution
18
19   ! solve
20   ! Solve heat conduction equation with finite volume scheme
21   ! (within iteration loop)
22
23   ! output
24   ! Save solution performance parameters to file
25
26 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
27
28 MODULE subroutines
29     USE CONSTANTS
30     USE BLOCKMOD
31     USE IO
32
33     IMPLICIT NONE
34
35 CONTAINS
36     SUBROUTINE init_gridsystem(blocks)
37         ! Initialize the solution with dirichlet B.C.s. Save to restart files.
38
39         TYPE(BLKTYPE) :: blocks(:)
40
41         ! INITIALIZE BLOCKS
42         CALL init_blocks(blocks)
43         ! WRITE BLOCK CONNECTIVITY FILE
44         CALL write_blocks(blocks)
45         ! INITIALIZE MESH
46         CALL init_mesh(blocks)
47         ! INITIALIZE TEMPERATURE WITH DIRICHLET B.C.
48         CALL init_temp(blocks)
49         ! WRITE GRID AND INITIAL TEMPERATURE TO PLOT3D RESTART FILES
50         CALL plot3D(blocks)
51
52     END SUBROUTINE init_gridsystem
53
54     SUBROUTINE init_solution(blocks, nbrlists)
55         ! Read initial conditions from restart files. Then calculate parameters
56         ! used in solution

```

```

57
58     TYPE(BLKTYPE)  :: blocks(:)
59     ! LINKED LISTS STORING NEIGHBOR INFO
60     TYPE(NBRLIST) :: nbrlists
61
62     ! READ BLOCK CONFIGURATION INFORMATION FROM CONFIG FILE
63     CALL read_blocks(blocks)
64
65     ! READ GRID AND INITIAL TEMPERATURE FROM PLOT3D RESTART FILE
66     CALL readPlot3D(blocks)
67
68
69     ! CALC LOCAL BOUNDARIES OF CELLS
70     write(*,*) 'set local bounds'
71     CALL set_block_bounds(blocks)
72
73
74
75     ! INITIALIZE LINKED LISTS CONTAINING BOUNDARY INFORMATION
76     write(*,*) 'make linked lists'
77     CALL init_linklists(blocks, nbrlists)
78     ! POPULATE BLOCK GHOST NODES
79     write(*,*) 'update ghosts'
80     CALL update_ghosts(blocks, nbrlists)
81
82 !     CALL update_ghosts_debug(blocks)
83
84     ! CALC AREAS FOR SECONDARY FLUXES
85     write(*,*) 'calc solution stuff'
86     CALL calc_cell_params(blocks)
87     ! CALC CONSTANTS OF INTEGRATION
88     CALL calc_constants(blocks)
89
90 END SUBROUTINE init_solution
91
92
93 SUBROUTINE solve(blocks, nbrlists, iter, res_hist)
94     ! Solve heat conduction equation with finite volume scheme
95     ! (within iteration loop)
96
97     TYPE(BLKTYPE) :: blocks(:)
98     ! LINKED LISTS STORING NEIGHBOR INFO
99     TYPE(NBRLIST) :: nbrlists
100    ! Residual history linked list
101    TYPE(RESLIST), POINTER :: res_hist
102    ! pointer to iterate linked list
103    TYPE(RESLIST), POINTER :: hist
104    ! Minimum residual criteria for iteration, actual residual
105    REAL(KIND=8) :: res = 1000.D0, resloc, resmax
106    ! iter in function inputs so it can be returned to main
107    INTEGER :: iter, IBLK, IBLKRES
108
109    INCLUDE "mpif.h"
110    REAL(KIND=8) :: start_solve, end_solve
111    WRITE(*,*) 'Starting clock for solver...'
112    start_solve = MPI_Wtime()
113
114    ! residual history
115    ALLOCATE(res_hist)
116    hist => res_hist
117
118    iter_loop: DO WHILE (res >= min_res .AND. iter <= max_iter)
119        ! Iterate FV solver until residual becomes less than cutoff or
120        ! iteration count reaches given maximum
121
122        ! CALC NEW TEMPERATURE AT ALL POINTS
123        CALL calc_temp(blocks)
124
125        ! UPDATE GHOST NODES WITH NEW TEMPERATURE SOLUTION

```

```

126      CALL update_ghosts(blocks, nbrlists)
127      !      CALL update_ghosts_debug(blocks)
128
129      ! CALC RESIDUAL
130      resmax = 0.D0
131      DO IBLK = 1, NBLK
132          ! Find max of each block
133          resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Ttmp(2:IMAXBK-1, 2:JMAXBK-1) ) )
134          ! keep biggest residual
135          IF (resmax < resloc) THEN
136              resmax = resloc
137          END IF
138      END DO
139      ! FINAL RESIDUAL
140      res = resmax
141
142      ! SWITCH TO NEXT LINK
143      ! (skip first entry)
144      ALLOCATE(hist%next)
145      hist => hist%next
146      NULLIFY(hist%next)
147      ! STORE RESIDUAL HISTORY
148      hist%iter = iter
149      hist%res = res
150
151
152      ! INCREMENT ITERATION COUNT
153      iter = iter + 1
154
155  END DO iter_loop
156
157      ! there was an extra increment after final iteration we need to subtract
158      iter = iter - 1
159
160      ! CACL SOLVER WALL CLOCK TIME
161      end_solve = MPI_Wtime()
162      wall_time_solve = end_solve - start_solve
163
164      IF (iter > max_iter) THEN
165          WRITE(*,*) 'DID NOT CONVERGE (NUMBER OF ITERATIONS:', iter, ')'
166      ELSE
167          WRITE(*,*) 'CONVERGED (NUMBER OF ITERATIONS:', iter, ')'
168          WRITE(*,*) '          (MAXIMUM RESIDUAL      :', res, ')'
169      END IF
170  END SUBROUTINE solve
171
172  SUBROUTINE output(blocks, iter)
173      ! Save solution performance parameters to file
174
175      TYPE(BLKTYPE), TARGET :: blocks(:)
176      REAL(KIND=8), POINTER :: tmpT(:,,:), tempTemperature(:, :)
177      REAL(KIND=8) :: resloc, resmax
178      INTEGER :: iter, I, J, IBLK, IRES
179
180      !      Temperature => mesh%T(2:IMAX-1, 2:JMAX-1)
181      !      tempTemperature => mesh%Ttmp(2:IMAX-1, 2:JMAX-1)
182
183      ! CALC RESIDUAL
184      resmax = 0.D0
185      DO IBLK = 1, NBLK
186          ! Find max of each block
187          resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Ttmp(2:IMAXBK-1, 2:JMAXBK-1) ) )
188          ! keep biggest residual
189          IF (resmax < resloc) THEN
190              resmax = resloc
191              IRES = IBLK
192          END IF
193      END DO
194

```

```

195
196 ! Write final maximum residual and location of max residual
197 ! OPEN(UNIT = 1, FILE = casedir // "SteadySoln.dat")
198 ! DO i = 1, IMAX
199 !     DO j = 1, JMAX
200 !         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)
201 !     END DO
202 ! END DO
203 ! CLOSE (1)
204
205 ! Screen output
206 tmpT => blocks(IRES)%mesh%Tmp
207 WRITE (*,*), "IMAX/JMAX", IMAX, JMAX
208 WRITE (*,*), "N/M", N, M
209 WRITE (*,*), "iters", iter
210 WRITE (*,*), "max residual", MAXVAL(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
211 WRITE (*,*), "on block id", IRES
212 WRITE (*,*), "residual ij", MAXLOC(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
213
214 ! Write to file
215 ! OPEN (UNIT = 2, FILE = TRIM(casedir) // "SolnInfo.dat")
216 OPEN (UNIT = 2, FILE = "SolnInfo.dat")
217 WRITE (2,*), "Running a", IMAX, "by", JMAX, "grid,"
218 WRITE (2,*), "With NxM:", N, "x", M, "blocks took:"
219 WRITE (2,*), iter, "iterations"
220 WRITE (2,*), wall_time_total, "seconds (Total CPU walltime)"
221 WRITE (2,*), wall_time_solve, "seconds (Solver CPU walltime)"
222 ! WRITE (2,*), wall_time_iter, "seconds (Iteration CPU walltime)"
223 WRITE (2,*)
224 WRITE (2,*), "Found max residual of ", MAXVAL(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
225 WRITE (2,*), "on block id", IRES
226 WRITE (2,*), "At ij of ", MAXLOC(tmpT(2:IMAXBLK-1, 2:JMAXBLK-1))
227 CLOSE (2)
228 END SUBROUTINE output
229
230
231
232 END MODULE subroutines

```

Listing 2: Main subroutines used for solving heat transfer on a multi-block grid

Appendix D: Multi-Block Plot3D Reader-Writer

```
1  ! MAE 267
2  ! PROJECT 3
3  ! LOGAN HALSTROM
4  ! 03 NOVEMBER 2015
5
6  ! DESCRIPTION: This module contains functions for information input and output.
7  ! Write grid and temperature files in PLOT3D format.
8  ! Write and read block grid configuration file
9
10 ! NOTE: How to Visualize Blocks in Paraview:
11 ! open unformatted PLOT3D file.
12 ! Change 'Coloring' from 'Solid' to 'vtkCompositeIndex'
13
14 MODULE IO
15   USE CONSTANTS
16   USE BLOCKMOD
17   IMPLICIT NONE
18
19   ! VARIABLES
20   INTEGER :: gridUnit = 30 ! Unit for grid file
21   INTEGER :: tempUnit = 21 ! Unit for temp file
22   INTEGER :: resUnit = 23
23   REAL(KIND=8) :: tRef = 1.D0 ! tRef number
24   REAL(KIND=8) :: dum = 0.D0 ! dummy values
25
26   ! LINKED LIST OF RESIDUAL HISTORY
27
28   TYPE RESLIST
29     ! Next element in linked list
30     TYPE(RESLIST), POINTER :: next
31     ! items in link:
32     REAL(KIND=8) :: res
33     INTEGER :: iter
34   END TYPE RESLIST
35
36   CONTAINS
37   SUBROUTINE plot3D(blocks)
38     IMPLICIT NONE
39
40     TYPE(BLKTYPE) :: blocks(:)
41     INTEGER :: IBLK, I, J
42
43     ! FORMAT STATEMENTS
44     ! I --> Integer, number following is number of sig figs
45     ! E --> scientific notation,
46     ! before decimal is sig figs of exponent?
47     ! after decimal is sig figs of value
48     ! number before letter is how many entries on single line
49     ! before newline (number of columns)
50     10 FORMAT(I10)
51     20 FORMAT(10I10)
52     30 FORMAT(10E20.8)
53
54     !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
55
56     ! OPEN FILES
57     ! OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid_form.xyz',FORM='formatted')
58     ! OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T_form.dat',FORM='formatted')
59     OPEN(UNIT=gridUnit,FILE= 'grid_form.xyz',FORM='formatted')
60     OPEN(UNIT=tempUnit,FILE= 'T_form.dat',FORM='formatted')
61
62     ! WRITE TO GRID FILE
63     WRITE(gridUnit, 10) NBLK
64     WRITE(gridUnit, 20) ( IMAXBK, JMAXBK, IBLK=1, NBLK)
65     ! WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
66     DO IBLK = 1, NBLK
67       WRITE(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBK), J=1,JMAXBK), &
```



```

68          ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
69
70 END DO
71
72 ! WRITE TO TEMPERATURE FILE
73 ! When read in paraview, 'density' will be equivalent to temperature
74 WRITE(tempUnit, 10) NBLK
75 WRITE(tempUnit, 20) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
76 DO IBLK = 1, NBLK
77
78     WRITE(tempUnit, 30) tRef,dum,dum,dum
79     WRITE(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
80                          ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
81                          ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
82                          ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
83
84 END DO
85
86 ! CLOSE FILES
87 CLOSE(gridUnit)
88 CLOSE(tempUnit)
89
90 !!! UNFORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
91
92 ! OPEN FILES
93 ! OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid.xyz',FORM='unformatted')
94 ! OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T.dat',FORM='unformatted')
95 OPEN(UNIT=gridUnit,FILE = 'grid.xyz',FORM='unformatted')
96 OPEN(UNIT=tempUnit,FILE = 'T.dat',FORM='unformatted')
97
98 ! WRITE TO GRID FILE (UNFORMATTED)
99 ! (Paraview likes unformatted better)
100 WRITE(gridUnit) NBLK
101 WRITE(gridUnit) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
102 ! WRITE(gridUnit) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
103 DO IBLK = 1, NBLK
104     WRITE(gridUnit) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
105                     ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
106
107 END DO
108
109 ! WRITE TO TEMPERATURE FILE
110 ! When read in paraview, 'density' will be equivalent to temperature
111 WRITE(tempUnit) NBLK
112 WRITE(tempUnit) ( IMAXBLK, JMAXBLK, IBLK=1, NBLK)
113 DO IBLK = 1, NBLK
114
115     WRITE(tempUnit) tRef,dum,dum,dum
116     WRITE(tempUnit) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
117                     ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
118                     ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
119                     ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
120
121 END DO
122
123 ! CLOSE FILES
124 CLOSE(gridUnit)
125 CLOSE(tempUnit)
126 END SUBROUTINE plot3D
127
128 SUBROUTINE readPlot3D(blocks)
129 IMPLICIT NONE
130
131 TYPE(BLKTYPE) :: blocks(:)
132 INTEGER :: IBLK, I, J
133 ! READ INFO FOR BLOCK DIMENSIONS
134 INTEGER :: NBLKREAD, IMAXBLKREAD, JMAXBLKREAD
135
136 ! FORMAT STATEMENTS
137 ! I --> Integer, number following is number of sig figs
138 ! E --> scientific notation,

```

```

137         ! before decimal is sig figs of exponent?
138         ! after decimal is sig figs of value
139         ! number before letter is how many entries on single line
140         ! before newline (number of columns)
141     10     FORMAT(I10)
142     20     FORMAT(10I10)
143     30     FORMAT(10E20.8)
144
145     !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
146
147     ! OPEN FILES
148     ! OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid_form.xyz',FORM='formatted')
149     ! OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T_form.dat',FORM='formatted')
150     OPEN(UNIT=gridUnit,FILE= 'grid_form.xyz',FORM='formatted')
151     OPEN(UNIT=tempUnit,FILE= 'T_form.dat',FORM='formatted')
152
153     ! READ GRID FILE
154     READ(gridUnit, 10) NBLKREAD
155     READ(gridUnit, 20) ( IMAXBKREAD, JMAXBKREAD, IBLK=1, NBLKREAD)
156     ! WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
157     DO IBLK = 1, NBLKREAD
158         READ(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBK), J=1,JMAXBK), &
159                             ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBK), J=1,JMAXBK)
160     END DO
161
162
163     ! READ TEMPERATURE FILE
164     ! When read in paraview, 'density' will be equivalent to temperature
165     READ(tempUnit, 10) NBLKREAD
166     READ(tempUnit, 20) ( IMAXBKREAD, JMAXBKREAD, IBLK=1, NBLKREAD)
167     DO IBLK = 1, NBLKREAD
168
169         READ(tempUnit, 30) tRef,dum,dum,dum
170         READ(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBK), J=1,JMAXBK), &
171                             ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBK), J=1,JMAXBK), &
172                             ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBK), J=1,JMAXBK), &
173                             ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBK), J=1,JMAXBK)
174     END DO
175
176     ! CLOSE FILES
177     CLOSE(gridUnit)
178     CLOSE(tempUnit)
179 END SUBROUTINE readPlot3D
180
181 SUBROUTINE write_res(res_hist)
182     TYPE(RESLIST), POINTER :: res_hist
183     ! pointer to iterate linked list
184     TYPE(RESLIST), POINTER :: hist
185
186     ! open residual file
187     ! OPEN(UNIT=resUnit,FILE= TRIM(casedir) // 'res_hist.dat')
188     OPEN(UNIT=resUnit,FILE = 'res_hist.dat')
189     ! column headers
190     WRITE(resUnit,*) 'ITER      RESID'
191
192     ! point to residual linked list
193     hist => res_hist
194     ! skip first link, empty from iteration loop design
195     hist => hist%next
196     ! write residual history to file until list ends
197     DO
198         IF ( .NOT. ASSOCIATED(hist) ) EXIT
199         ! write iteration and residual in two columns
200         WRITE(resUnit,*) hist%iter, hist%res
201         hist => hist%next
202     END DO
203
204     CLOSE(resUnit)
205 END SUBROUTINE write_res

```

```

206
207
208 END MODULE IO

```

Listing 3: Code for saving formatted multiblock PLOT3D solution files and reading restart files

Appendix E: Other Relevant Codes

[illegible]

```

59
60 ! INITIALIZE SOLUTION
61 CALL init_solution(blocks, nbrlists)
62 ! SOLVE
63 WRITE(*,*) 'Solving heat conduction...'
64 CALL solve(blocks, nbrlists, iter, res_hist)
65
66 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
67 !!! SAVE RESULTS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
68 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
69
70 WRITE(*,*) 'Writing results...'
71 ! SAVE SOLUTION AS PLOT3D FILES
72 CALL plot3D(blocks)
73 ! CALC TOTAL WALL TIME
74 end_total = MPI_Wtime()
75 wall_time_total = end_total - start_total
76
77 ! SAVE RESIDUAL HISTORY
78 CALL write_res(res_hist)
79 ! SAVE SOLVER PERFORMANCE PARAMETERS
80 CALL output(blocks, iter)
81
82 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
83 !!! CLEAN UP !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
84 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
85
86 DO IBLK = 1, NBLK
87     DEALLOCATE( blocks(IBLK)%mesh%xp )
88     DEALLOCATE( blocks(IBLK)%mesh%yp )
89     DEALLOCATE( blocks(IBLK)%mesh%x )
90     DEALLOCATE( blocks(IBLK)%mesh%y )
91     DEALLOCATE( blocks(IBLK)%mesh%T )
92     DEALLOCATE( blocks(IBLK)%mesh%Ttmp )
93     DEALLOCATE( blocks(IBLK)%mesh%dt )
94     DEALLOCATE( blocks(IBLK)%mesh%V )
95     DEALLOCATE( blocks(IBLK)%mesh%V2nd )
96     DEALLOCATE( blocks(IBLK)%mesh%term )
97     DEALLOCATE( blocks(IBLK)%mesh%yPP )
98     DEALLOCATE( blocks(IBLK)%mesh%yNP )
99     DEALLOCATE( blocks(IBLK)%mesh%yNN )
100    DEALLOCATE( blocks(IBLK)%mesh%yPN )
101    DEALLOCATE( blocks(IBLK)%mesh%xNN )
102    DEALLOCATE( blocks(IBLK)%mesh%xPN )
103    DEALLOCATE( blocks(IBLK)%mesh%xPP )
104    DEALLOCATE( blocks(IBLK)%mesh%xNP )
105 END DO
106
107 WRITE(*,*) 'Done!'
108
109 ! MOVE OUTPUT FILE TO OUTPUT DIRECTORY
110 ! CALL EXECUTE_COMMAND_LINE ("mv a.out " // casedir // '.')
111
112
113 END PROGRAM heatTrans

```

Listing 4: Wrapper program