

MAE 267 – Project 5

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 demonstrates the fundamentals of parallel computing through the numerical solution of the steady-state, two-dimensional temperature distribution of a 1m x 1m steel block with properties listed in Table 1.

Table 1: Steel Block Properties

| | |
|----------------------|--------------------------------|
| Dimensions | 1m x 1m |
| Thermal Conductivity | $k = 18.8 \frac{W}{m \cdot K}$ |
| Density | $\rho = 8000 \frac{kg}{m^3}$ |
| Specific Heat Ratio | $c_p = 500$ |

The demonstration of parallel computing techniques was accomplished in stages, starting with a serial (single-processor) solution of a single grid of dimensions 101x101 and 501x501, which serves as a solver basis and performance benchmark for later parallel codes.

The next stage was to divide the grid into NxM subdomains (blocks), on each of which the solution for a given iteration was calculated independently. 5x4 and 10x10 block decompositions of both previous grid dimensions were solved to demonstrate compartmentalization of solver processes, which is a necessary step for distributing processes in parallel computing.

Finally, the code was adapted to solve multi-block decompositions on multiple processors for the 501x501 grid decomposed into 10x10 blocks running on 1 to 8 processors. For this solution, the domain is decomposed and blocks are distributed on to processors. Decompositions are saved to restart files for each processor to be loaded by each processor in the parallel solver.

2 Methods and Equations

The core of this demonstration code is the heat transfer solver developed in the first project, but a number of domain decomposition functions have since been included, as will be detailed in this section.

2.1 Grid Initialization

The numerical solution is initialized with the Dirichlet boundary conditions (Eqn 1) using a single processor.

$$T_{BCs} = \begin{cases} 5.0 [\sin(\pi x_p) + 1.0] & \text{for } j = j_{max} \\ |\cos(\pi x_p)| + 1.0 & \text{for } j = 0 \\ 3.0 y_p + 2.0 & \text{for } i = 0, i_{max} \end{cases} \quad (1)$$

$$\begin{aligned} rot &= 30.0 \frac{\pi}{180.0} \\ x_p(i) &= \cos \left[0.5\pi \frac{i_{max} - i}{i_{max} - 1} \right] \\ y_p(j) &= \cos \left[0.5\pi \frac{j_{max} - j}{j_{max} - 1} \right] \\ x &= x_p \cos(rot) + (1.0 - y_p) \sin(rot) \\ y &= y_p \cos(rot) + x_p \sin(rot) \end{aligned} \quad (2)$$

Square grids are generated according to Eqn 2 to create non-uniform spacing in both the x and y directions (with finer spacing at the larger indices). The “prime” system is then rotated by angle rot to create the final grid.

2.2 Numerical Solver

The solver developed for this analysis utilizes a finite-volume numerical solution method to solve the transient heat conduction equation (Eqn 3).

$$\rho c_p \frac{\partial T}{\partial t} = k \left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right] \quad (3)$$

To solve Eqn 3 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.

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

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

2.3 Subdomain Decomposition

After grid initialization, the grid is divided into N blocks in the x/I direction and M blocks in the y/J direction, creating a total number of blocks $NBLK = N \cdot M$. All blocks are constrained to have the same number of nodes, so the dimensions of every block $IBLKMAX$ and $JBLKMAX$ are calculated in Eqn 6 as a fraction of the total number of nodes in each direction, including one point overlap at each inter-block boundary (Ghost nodes are excluded for the moment). In the I-direction, the total number of nodes including overlap (Eqn 5) is:

$$IMAX_{tot} = IMAX + (N - 1) \quad (5)$$

and the total number of nodes per block in the I-direction (Eqn 6) is:

$$IMAXBLK = \frac{IMAX_{tot}}{N} = \frac{IMAX + (N - 1)}{N} = 1 + \frac{IMAX - 1}{N} \quad (6)$$

Note: For points in J-direction, replace I with J and N with M

Blocks are distributed from 1 to NBLK starting in the lower-left corner of the grid and zipping left to right (the x/I/N direction), then up one (the y/J/M direction) starting again at the left. This is accomplished by two DO loops, the outer loop stepping through J from 1 to M and the inner loop stepping through I from 1 to N. Block locations are stored by assigning global starting indices to each block according to Eqn 7.

$$IMIN_{block} = IMIN_{global} + (IMAXBLK - 1)(I - 1) \quad (7)$$

where I counts blocks in the direction of N and $IMIN_{global} = 1$. The first block in the N-direction has a global starting index of 0, and IMAXBLK must be reduced by one to account for the single-point overlap at block boundaries.

Information for each block is stored as an element in an array of BLKTYPE derived data types. BLKTYPE stores local mesh, temperature, and solver information as well as the block ID, global indices, iteration bounds to prevent overwriting boundary conditions (discussed in Section 2.5), and neighbor identification information.

2.4 Processor Distribution

For the parallel code, blocks are distributed among $NPROCS$ processors (determined in 'miprun call), with the goal of equal load balancing for all processors (Eqn 8). Load balance is the ratio of a processor's workload to the "Perfect Load Balance" (PLB), the total load of all blocks divided by $NPROCS$. In this code, a block's load is referred to as its $SIZE$, so a processor's work load is equal to the sum of the $SIZEs$ of its blocks.

$$P_{LoadBalance} = \frac{SUM(SIZES)}{PLB} \quad (8)$$

The workload of each block ($SIZE$) is calculated as a weighted sum (Eqn 11) of its geometric cost $GEOM$ due to grid size (Eqn 9) and communication cost $COMM$ due to boundary size (10). Geometric cost is essentially the node area of the block iteration bounds:

$$GEOM = (IMAXLOC - IMINLOC) \cdot (JMAXLOC - JMINLOC) \quad (9)$$

Geometric cost will be greater for cells that are not on physical boundaries as they require more ghosts nodes for their inter-block boundaries. Communication cost is calculated as the total length of all faces and corners at interblock boundaries:

$$COMM(i) = \begin{cases} 0, & \text{if BC} \\ IMAXBLK - IMINBLK, & \text{if N or S Face Neighbor} \\ JMAXBLK - JMINBLK, & \text{if E or W Face Neighbor} \\ 1, & \text{if Corner Neighbor} \end{cases}$$

$$COMM = SUM(COMM(i)) \quad (10)$$

where Eqn 10 must be evaluated for all faces and corners of a given block and the results must be summed.

Weights of each type of cost are currently set to make the maximum possible geometric cost equal to the maximum possible communication cost, as accomplished by Eqn 11.

$$WGEOM = 1$$

$$WCOMM = FACTOR \cdot \frac{(IMAXBLK + 2)(JMAXBLK + 2)}{(2 \cdot IMAXBLK) + (2 \cdot IMAXBLK) + 4}$$

$$SIZE = (WGEOM \cdot GEOM) + (WCOMM \cdot COMM) \quad (11)$$

where *FACTOR* is a number that can be varied to tune cost weighting, but is currently set to 1.

Once block loads are calculated, they are sorted by size in order of greatest to least. They are then distributed to the processors in this order, where each block is assigned to the current processor with the least load. This produces the theoretical load balancing presented in Section 3. Actual load balancing performance will be determined in Project 5 and tuning will be performed to optimized load balancing.

2.5 Ghost Nodes and Neighbor Identification

In order for each block to function independently for a given iteration of the solver, it must know information about the nodes immediately outside of its boundaries, or, in other words, the interior nodes of its neighbors. To preserve block independence, each block stores the information it needs from its neighbor at the beginning of each iteration in extra, off-block nodes called ghost nodes. These nodes change the local size of each block and necessitate the local iteration parameters *ILOCMIN*, *ILOCMAX*, etc. discussed earlier.

To update each ghost boundary, the identity of the neighbor block for each face is stored in a variable *NB*, which is a neighbor derived data type *NBRTYPE*, which contains IDs for the north, south, east, and west faces and the north east, south east, south west, and north west corners. If the block boundary is a physical boundary instead of an inter-block boundary, the corresponding neighbor identifier is instead set to 0 to indicate a BC boundary. For parallel computing, if a neighbor block is on a different processor (indicating a processor boundary), the neighbor block ID is negated to indicate as such while still preserving the neighbor block ID.

Neighbor information is used to populate a linked list for each boundary type with block IDs so that all similar types of boundaries may be looped through in sequence, rather than using logical sorting at the beginning of each iteration. (Linked lists were shown to produce a 25% speed-up compared to logical sorting for the serial, multi-block code).

When moving to parallel computing, the ID of the neighbor blocks processor must also be bookeeped, as it is required information for accessing the neighbor block for ghost updating. In addition to the neighbor blocks processor, the local index of the neighbor block on its processor must also be stored for this same reason. Thus, this data is stored in corresponding *NBRTYPEs*. Neighbor processor IDs are stored in the variable *NP*. If a block boundary is a BC, the processor ID is negated to indicate as such. Local

indices of neighbor blocks on neighbor processors are stored in *NBLOC* and are set to 0 if a boundary is a BC.

2.6 Configuration Restart Files

After all of the above mentioned initialization processes have been completed, this information is stored in individual restart files for each processor so that the solver may start up independently from these files without needing to determine boundary procedures. Neighbor information, grid, and temperature files are written **for each processor**.

2.7 Parallel Performance

Performance of the parallel solver is measured in the results section using Amdahl's Law and the concepts of parallel speedup S_P (Eqn 12) and parallel computational efficiency E_P (Eqn 13).

$$S_P = \frac{t_s}{t_p} \quad (12)$$

where t_s is the serial solution time and t_p is the parallel solution time.

Ideal speedup produces a line with a slope of 1, with the serial solution time decreasing by exactly the amount of processors used. Ideal efficiency is 1, and corresponds to the ideal speedup.

$$E_P = \frac{S_P}{N_P} \quad (13)$$

where N_P is the number of processors in the parallel solution.

Speedup can be predicted as a function of the fraction of the code that is run in parallel according to Eqn 14. Because only the parallel portion of this code is timed, optimal speedup is expected to be near-ideal.

$$S_{P,opt} = \frac{N_P}{f + (1 - f)N_P} \quad (14)$$

3 Results and Discussion

Solutions of a 501x501 grid decomposed into 10x10 blocks were solved on 2, 4, 6, and 8 processors on the hpc1 front end (Batch jobs on hpc1 took significantly longer, which seemed to be due to running more processes than allocated cores).

Another important lesson learned from this project was the importance of shared variables in MPI. Many days of debugging ultimately resulted in discovering that the global grid size was only read in by Processor 0, and was not available to the other processors for calculation of mesh parameters. This was solved with a simple `MPI_BCAST`.

Another pitfall was in creating unique tag IDs for MPI sends and receives. Each send is identified by the sending and receiving processor, as well as a tag, so the first attempt

used 8 independent tags for the 8 directions of communication. This proved to be non-unique as one processor could send to another processor more than once in the same direction for a given iteration. In the end, a unique tag was created for each send as the concatenation of the direction number with the global sending block ID.

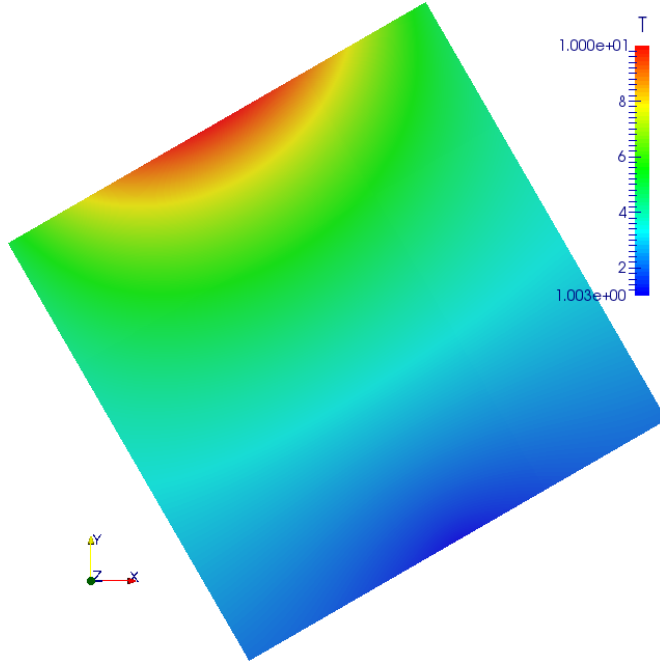


Fig. 1: Steady-state heat transfer solution for a 501x501 grid decomposed into 10x10 blocks solved on 4 processors

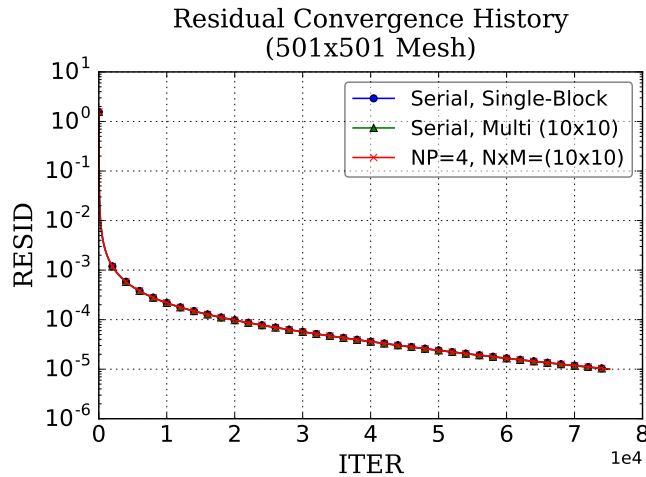


Fig. 2: Serial and parallel convergence history demonstrating similar performance between all solution methods

Convergence history for the serial, single-block and multi-block cases are compared to the parallel case in Fig 2.

It can be seen from the convergence comparison that the parallel solver performs almost identically to its serial analogs, and Fig 1 demonstrates that the parallel solution is accurate.

After much deliberation with hpc1, results demonstrating actual parallel speedup were produced and total solution wall times are presented for serial and parallel runs of the Project 5 code in Table 2.

Table 2: **Serial (Left) and Parallel (Right) Solutions Times**

| N_P | (1x1) | (10x10) | 2 | 4 | 6 | 8 |
|---------|-------|---------|-------|-------|-------|-------|
| t (s) | 360.7 | 377.8 | 179.4 | 91.95 | 70.56 | 76.09 |

It can be seen that the parallel code does indeed reach a solution faster than its serial analogs. The speedup and efficiency of the parallel calculations are further demonstrated in Figures 3 and 4.

Because near-perfect load balancing was accomplished (shown in Table 3) and because the timed portion of the code was entirely parallel (see Eqn 14), it was expected that optimal speedups would be near the ideal case, as Fig 3 demonstrates.

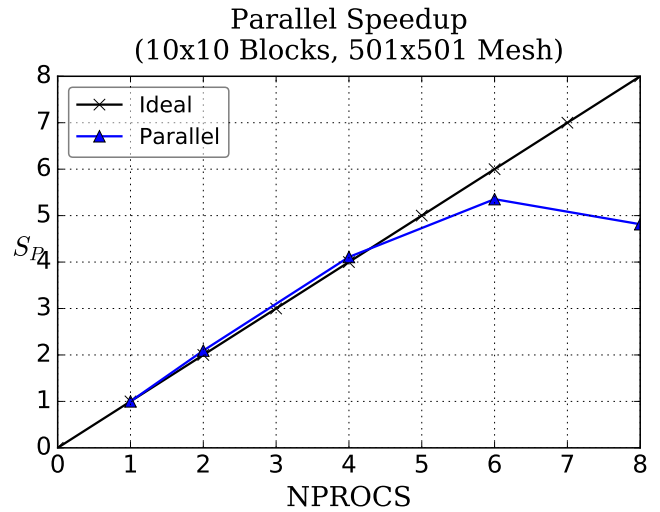


Fig. 3: Parallel speedup for a 501x501 grid decomposed into 10x10 blocks demonstrating an ideal speedup and sometimes super-linear that tapers off as the number of processors increases

Fig 3 also demonstrates that the maximum possible speedup for this case is approximately 5.25 running on 6 processors, which is a significant improvement on the performance of the serial version.

The parallel efficiency of the solver is greatest for lower processor numbers. It actually exceeds maximum efficiency for the 2 and 4 processor cases, but this phenomenon may be due to variation in loading of the hpc1 front-end cores between runs.

A steep drop-off in efficiency is noted as the number of processors is increased. To extend the curve further out, the solver could be applied to a larger grid system decomposed into more sub-domains.

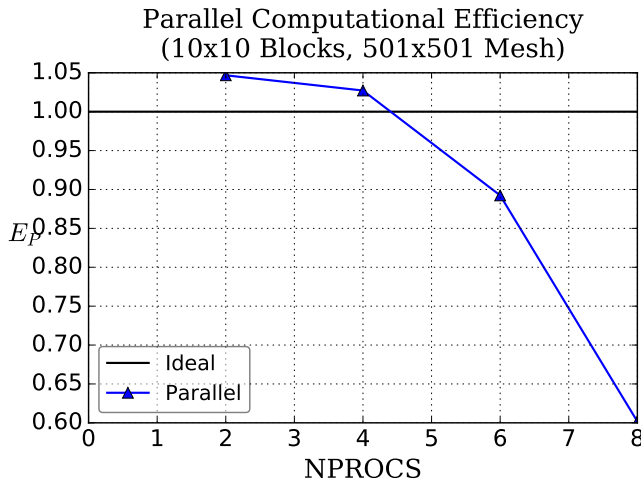


Fig. 4: Parallel efficiency for a 501x501 grid decomposed into 10x10 blocks demonstrating a greater than ideal efficiency for lower amounts of processors that may be due to variations in core loading

Table 3: **Processor Theoretical Load Balances**

| <i>NPROCS</i> | 4 | 8 |
|---------------|--------|---------|
| Proc0 | 1.0000 | 1.0306 |
| Proc1 | 1.0000 | 1.0306 |
| Proc2 | 1.0000 | 1.0306 |
| Proc3 | 1.0000 | 1.0306 |
| Proc4 | N/A | 0.96943 |
| Proc5 | N/A | 0.96943 |
| Proc6 | N/A | 0.96943 |
| Proc7 | N/A | 0.96943 |

4 Conclusion

The product of this project is a parallel, multi-block heat conduction solver demonstrating the fundamental principles of computations in parallel. It increased knowledge of parallel information passing, parallel speedup performance, load balancing optimization.

The working code is just a beginning of what can be accomplished with parallel computing. This code itself could be improved with better sub-domain decomposition optimization and optimization of calculations like send/recieve tags. Beyond that, the principles of this code could be applied to numerical solvers of different principles such as fluid

dynamics or structures. Parallel computing could be adapted to GPUs instead of CPU cores. The list is endless.

These projects served as an effective introduction to parallel computing by gradually building on principles; starting with a standard numeric heat transfer solver and then incrementally incorporating parallel computing concepts like domain decomposition and solution restart files so that each concept could be solidified before the next was attempted. Aside from developing a strong parallel computing basis from which to build off of, the course also developed skills for acquiring further knowledge, leaving the student prepared to delve further into the field of parallel computing.

This course is highly recommended for students interested in numerical solutions of engineering problems, as the principles of these projects can be adapted to any application that may be of interest.

Appendix A: Parallel, Multi-Block Grid Wrapper Code

```

1 ! MAE 267
2 ! PROJECT 5
3 ! LOGAN HALSTROM
4 ! 29 NOVEMBER 2015
5
6
7 ! DESCRIPTION: Solve heat conduction equation for single block of steel.
8
9 ! INPUTS: Set grid size, block decomposition, debug in 'config.in'
10 !         Set number of processors in 'run.sh'
11
12 ! TO COMPILE:
13 ! mpif90 -o main -O3 modules.f90 inout.f90 subroutines.f90 main.f90
14 ! makes executable file 'main'
15 ! 'rm *.mod' afterward to clean up unneeded compiled files
16
17 ! TO RUN:
18 ! on hpcl nodes: sbatch run.sh
19 ! on hpcl front end: ./main or ./run.sh
20
21 PROGRAM heatTrans
22     USE CONSTANTS
23     USE subroutines
24
25     IMPLICIT NONE
26
27     !! INITIALIZE VARIABLES !!
28     !!! INITIALIZE VARIABLES !!!
29     !!!! INITIALIZE VARIABLES !!!!
30
31     ! ALL BLOCKS IN ONE LIST
32     TYPE(BLKTYPE), ALLOCATABLE :: allblocks(:)
33
34     ! PROCESSORS
35     TYPE(PROCTYPE), ALLOCATABLE :: procs(:)
36     CHARACTER(2) :: procname
37     CHARACTER(20) :: xfile, qfile
38     ! ITERATION PARAMETERS
39     ! Residual history linked list
40     TYPE(RESLIST), POINTER :: res_hist
41     ! Maximum number of iterations
42     INTEGER :: iter = 1, IBLK, IP
43     REAL(KIND=8) :: start_total, end_total
44     REAL(KIND=8) :: start_solve, end_solve
45     ! CLOCK TOTAL TIME OF RUN
46     start_total = MPI_Wtime()
47
48
49     write(*,*) 'starting mpi'
50
51     !!! START MPI !!!
52     !! INITIALIZE MPI !!
53     CALL MPI_Init(IERROR)
54     ! DETERMINE MY PROCESSOR ID
55     ! ARGUMENTS: COMM, MYID, IERROR
56     CALL MPI_Comm_rank(MPI_COMM_WORLD, MYID, IERROR)
57     ! write(*,*) mpi_comm_world
58     ! FIND OUT HOW MANY PROCESSORS ARE USED
59     ! ARGUMENTS: COMM, NPROCS, IERROR
60     CALL MPI_Comm_size(MPI_COMM_WORLD, NPROCS, IERROR)
61
62     !!! INITIALIZE !!!

```

```

68
69 ! READ INPUTS FROM FILE
70 CALL read_input()
71
72 ! have the first processor only set up problem
73 IF(MYID == 0) THEN
74
75     write(*,*) 'initializing'
76
77 !         ! READ INPUTS FROM FILE
78 !         CALL read_input()
79     ALLOCATE( allblocks(NBLK) )
80     ALLOCATE( procs(NPROCS) )
81     ! INITIALIZE GRID SYSTEM
82     WRITE(*,*) 'Making mesh...'
83     CALL init_gridsystem(allblocks, procs)
84
85     ! CLEAN UP INITIALIZATION
86     DEALLOCATE(allblocks, procs)
87 END IF
88
89 !     ! ONLY PROC 0 READS IN CONFIG DATA, SO BRODCAST TO ALL PROCS
90 !     ! (syntax: variable to broadcast, size, type, which proc to bcast from, otherstuff)
91 !     CALL MPI_Bcast(IMAX, 1, MPI_INT, 0, mpi_comm_world, ierror)
92 !     CALL MPI_Bcast(JMAX, 1, MPI_INT, 0, mpi_comm_world, ierror)
93
94 ! HOLD ALL PROCESSORS UNTIL INITIALIZATION IS COMPLETE
95 CALL MPI_Barrier(MPI_COMM_WORLD, IERROR)
96
97 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
98 !!! SOLVER !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
99 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
100
101 ! INITIALIZE SOLUTION
102 write(*,*) "Initialize for proc ", MYID
103 CALL init_solution(blocks, nbrlists, mpilists)
104
105 !     if (nprocs == 4) then
106 !         if (myid == 3) then
107 !             write(*,*) "block ", blocks(4)%ID
108 !             write(*,*) "iminloc ", blocks(3)%IMINLOC
109 !             write(*,*) "Imaxloc ", blocks(3)%IMaxLOC
110 !             write(*,*) "jminloc ", blocks(3)%jMINLOC
111 !             write(*,*) "jmaxloc ", blocks(3)%jmaxLOC
112
113 !             write(*,*) blocks(4)%mesh%term( imaxblk+1, jmaxblk+1)
114 !             write(*,*) blocks(4)%mesh%V2nd( imaxblk+1, jmaxblk+1)
115 !             write(*,*) blocks(4)%mesh%V ( imaxblk+1, jmaxblk+1)
116 !             write(*,*) blocks(4)%mesh%dt( imaxblk+1, jmaxblk+1)
117 !             write(*,*) blocks(4)%mesh%xp( imaxblk+1, jmaxblk+1)
118 !             write(*,*) blocks(4)%mesh%x( imaxblk+1, jmaxblk+1)
119
120 !         end if
121
122
123
124 !     else if (nprocs == 1) then
125 !         if (myid == 0) then
126 !             write(*,*) "block ", blocks(9)%ID
127 !             write(*,*) "iminloc ", blocks(10)%IMINLOC
128 !             write(*,*) "Imaxloc ", blocks(10)%IMaxLOC
129 !             write(*,*) "jminloc ", blocks(10)%jMINLOC
130 !             write(*,*) "jmaxloc ", blocks(10)%jmaxLOC
131 !             write(*,*) blocks(14)%mesh%term( imaxblk+1, jmaxblk+1)
132 !             write(*,*) blocks(14)%mesh%V2nd( imaxblk+1, jmaxblk+1)
133 !             write(*,*) blocks(14)%mesh%V ( imaxblk+1, jmaxblk+1)
134 !             write(*,*) blocks(14)%mesh%dt( imaxblk+1, jmaxblk+1)
135 !             write(*,*) blocks(14)%mesh%xp( imaxblk+1, jmaxblk+1)
136 !             write(*,*) blocks(14)%mesh%x( imaxblk+1, jmaxblk+1)

```

```

137 !           write(*,*) Imax
138 !       end if
139
140 !   end if
141
142
143 CALL MPI_Barrier(MPI_COMM_WORLD, IERROR)
144 ! SOLVE
145 WRITE(*,*) 'Solving heat conduction with Processor ', MYID
146 CALL solve(blocks, nbrlists, mpilists, iter, res_hist)
147
148 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
149 !!! SAVE RESULTS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
150 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
151
152 WRITE(*,*) 'Writing results...'
153
154 !TURN THIS ON FOR PJ5!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
155
156 ! SAVE SOLUTION AS PLOT3D FILES
157 ! MAKE FILE NAME
158 IF (MYID<10) THEN
159     ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
160     WRITE(procname, '(A,I1)') '0', MYID
161 ELSE
162     WRITE(procname, '(I2)') MYID
163 END IF
164 xfile = "p" // procname // ".grid"
165 qfile = "p" // procname // ".T"
166 CALL plot3D(blocks, MYNBLK, xfile, qfile)
167 ! CALC TOTAL WALL TIME
168 !   end_total = MPI_Wtime()
169 !   wall_time_total = end_total - start_total
170
171 !TURN THIS ON FOR PJ5!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
172
173 IF (MYID == 0) THEN
174     ! SAVE RESIDUAL HISTORY
175     CALL write_res(res_hist)
176 END IF
177 ! SAVE SOLVER PERFORMANCE PARAMETERS
178 CALL output(blocks, iter)
179
180
181 !   if (myid == 0) then
182 !       call compositePlot3D()
183 !   end if
184
185 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
186 !!! CLEAN UP !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
187 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
188
189 DEALLOCATE(blocks)
190
191 IF (MYID == 0) THEN
192     WRITE(*,*) 'Done!'
193 END IF
194
195 CALL MPI_Finalize(ierr)
196
197 END PROGRAM heatTrans

```

Listing 1: Wrapper program that performs domain decomposition on a single processor and then independently reads in solution for solver

Appendix B: Parallel, Multi-Block Grid Solver Code

```
1  ! MAE 267
2  ! PROJECT 5
3  ! LOGAN HALSTROM
4  ! 29 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   ! SOLUTION BLOCKS
36   ! (initialized individually for each parallel processor,
37   ! holds specific blocks distributed to each specific processor)
38   TYPE(BLKTYPE), POINTER :: blocks(:)
39   ! LINKED LISTS STORING NEIGHBOR INFO
40   TYPE(NBRLIST) :: nbrlists
41   ! neighbors on other processors
42   TYPE(NBRLIST) :: mpilists
43
44 CONTAINS
45   SUBROUTINE init_gridsystem(blocks, procs)
46     ! Initialize the solution with dirichlet B.C.s. Save to restart files.
47
48     TYPE(BLKTYPE) :: blocks(:)
49     TYPE(PROCTYPE) :: procs(:)
50
51     ! INITIALIZE BLOCKS
52     CALL init_blocks(blocks)
53
54     ! CALC LOCAL BOUNDARIES OF CELLS
55     CALL set_block_bounds(blocks)
56
57     ! INITIALIZE MESH
58     CALL init_mesh(blocks)
59     ! INITIALIZE TEMPERATURE WITH DIRICHLET B.C.
60     CALL init_temp(blocks)
61
62     ! DISTRIBUTE BLOCKS TO PROCESSORS
63     CALL dist_blocks(blocks, procs)
64     ! DETERMIN NEIGHBOR PROCESSOR INFORMATION
65     CALL init_neighbor_procs(blocks, procs)
66
67
```

```

68      ! WRITE BLOCK CONNECTIVITY FILE
69      CALL write_config(procs)
70
71  END SUBROUTINE init_gridsystem
72
73  SUBROUTINE init_solution(blocks, nbrlists, mpilists)
74      ! Read initial conditions from restart files. Then calculate parameters
75      ! used in solution
76
77      TYPE(BLKTYPE), POINTER :: blocks(:)
78      ! LINKED LISTS STORING NEIGHBOR INFO
79      TYPE(NBRLIST) :: nbrlists, mpilists
80
81      !
82      write(*,*) "read config", MYID
83      ! READ BLOCK CONFIGURATION INFORMATION FROM CONFIG FILE
84      CALL read_config(blocks)
85
86      ! INITIALIZE LINKED LISTS CONTAINING BOUNDARY INFORMATION
87      !
88      write(*,*) 'make linked lists', MYID
89      CALL init_linklists(blocks, nbrlists, mpilists)
90      ! POPULATE BLOCK GHOST NODES
91      !
92      write(*,*) 'update ghosts', MYID
93      CALL update_ghosts_sameproc(blocks, nbrlists)
94      CALL update_ghosts_diffproc_send(blocks, mpilists)
95      CALL update_ghosts_diffproc_recv(blocks, mpilists)
96
97      ! CALC AREAS FOR SECONDARY FLUXES
98      !
99      write(*,*) 'calc solution stuff', MYID
100     CALL calc_cell_params(blocks)
101     ! CALC CONSTANTS OF INTEGRATION
102     CALL calc_constants(blocks)
103
104  END SUBROUTINE init_solution
105
106  SUBROUTINE solve(blocks, nbrlists, mpilists, iter, res_hist)
107      ! Solve heat conduction equation with finite volume scheme
108      ! (within iteration loop)
109
110      TYPE(BLKTYPE) :: blocks(:)
111      ! LINKED LISTS STORING NEIGHBOR INFO
112      TYPE(NBRLIST) :: nbrlists, mpilists
113      ! Residual history linked list
114      TYPE(RESLIST), POINTER :: res_hist
115      ! pointer to iterate linked list
116      TYPE(RESLIST), POINTER :: hist
117      ! Minimum residual criteria for iteration, actual residual
118      REAL(KIND=8) :: res = 1000.D0, resloc=0.D0, resmax=0.D0
119      ! iter in function inputs so it can be returned to main
120      INTEGER :: iter, IBLK, IBLKRES
121
122      REAL(KIND=8) :: start_solve, end_solve
123      IF (MYID == 0) THEN
124          ! START SOLVER CLOCK
125          start_solve = MPI_Wtime()
126      END IF
127
128      ! residual history
129      ALLOCATE(res_hist)
130      hist => res_hist
131
132      iter_loop: DO WHILE (res >= min_res .AND. iter <= max_iter)
133          ! Iterate FV solver until residual becomes less than cutoff or
134          ! iteration count reaches given maximum
135
136      !
137          if (myid == 2 .or. myid == 3) then

```

```

137 !           write(*,*) "Proc, iter: ", myid, iter
138 !       end if
139
140 !       if (nprocs == 4) then
141
142 !           ! 4 proc 5x4
143 !           if (myid == 0) then
144 !               write(*,*) "blk3 east interior values ", blocks(3)%mesh%T(IMAXBK-1, 2)
145 !               write(*,*) "blk3 east face values ",      blocks(3)%mesh%T(IMAXBK, 2)
146 !               write(*,*) "blk3 east ghost values ",      blocks(3)%mesh%T(IMAXBK+1, 2)
147 !           end if
148
149 !           if (myid == 3) then
150 !               write(*,*) "blk4 west ghost values ",      blocks(3)%mesh%T(0, 2)
151 !               write(*,*) "blk4 west face values ",      blocks(3)%mesh%T(1, 2)
152 !               write(*,*) "blk4 west interior values ", blocks(3)%mesh%T(2, 2)
153 !           end if
154
155 !           ! compare node value, should be the same
156 !           if (myid == 2) then
157 !               write(*,*) "          block", blocks(3)%ID, &
158 !                   "SW node", blocks(3)%mesh%T(1, 1)
159 !               write(*,*) "          block", blocks(1)%ID, &
160 !                   "NW node", blocks(1)%mesh%T(1, jmaxblk)
161 !           end if
162 !           if (myid == 3) then
163 !               write(*,*) "          block", blocks(4 )%ID, &
164 !                   "NE node", blocks(4 )%mesh%T(IMAXBK, JMAXBK)
165 !               write(*,*) "          block", blocks(5)%ID, &
166 !                   "SE node", blocks(5)%mesh%T(IMAXBK, 1)
167 !           end if
168
169 !           ! compare ghost info transfer
170 !           if (myid == 2) then
171 !               write(*,*) "          block", blocks(3)%ID, &
172 !                   "send SW node", blocks(3)%mesh%T(2, 2)
173 !           end if
174 !           if (myid == 3) then
175 !               write(*,*) "          block", blocks(4)%ID, &
176 !                   "recv NE node", blocks(4)%mesh%T(imaxblk+1, jmaxblk+1)
177 !           end if
178
179
180
181
182 !       else if (nprocs == 1) then
183
184 !           ! 1 proc 5x4
185 !           if (myid == 0) then
186 !               write(*,*) "blk3 east interior values ", blocks(11)%mesh%T(IMAXBK-1, 2)
187 !               write(*,*) "blk3 east face values ",      blocks(11)%mesh%T(IMAXBK, 2)
188 !               write(*,*) "blk3 east ghost values ",      blocks(11)%mesh%T(IMAXBK+1, 2)
189 !               write(*,*)
190 !               write(*,*) "blk4 west ghost values ",      blocks(10)%mesh%T(0, 2)
191 !               write(*,*) "blk4 west face values ",      blocks(10)%mesh%T(1, 2)
192 !               write(*,*) "blk4 west interior values ", blocks(10)%mesh%T(2, 2)
193 !           end if
194 !       end if
195
196
197 !       write(*,*) "calc temp ", myid
198
199 !       CALC NEW TEMPERATURE AT ALL POINTS
200 CALL calc_temp(blocks)
201
202 !       write(*,*) "update ghosts ", myid
203
204 !       UPDATE GHOST NODES WITH NEW TEMPERATURE SOLUTION
205 CALL update_ghosts_sameproc(blocks, nbrlists)

```

```

206      CALL update_ghosts_diffproc_send(blocks, mpilists)
207      CALL update_ghosts_diffproc_recv(blocks, mpilists)
208
209      !      write(*,*) "residual ", myid
210
211      ! CALC RESIDUAL FOR LOCAL BLOCKS
212      resmax = 0.D0
213      DO IBLK = 1, MYNBLK
214          ! Find max of each block
215          resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Tmp(2:IMAXBK-1, 2:JMAXBK-1) ) )
216          ! keep biggest residual
217          IF (resloc > resmax) THEN
218              resmax = resloc
219          END IF
220      END DO
221      ! FINAL MAX RESIDUAL (FOR ALL PROCESSORS)
222      CALL MPI_ALLREDUCE(resmax, res, 1, MPI_REAL8, MPI_MAX, &
223          MPI_COMM_WORLD, IERROR)
224
225      ! SWITCH TO NEXT LINK
226      ! (skip first entry)
227      ALLOCATE(hist%next)
228      hist => hist%next
229      NULLIFY(hist%next)
230      ! STORE RESIDUAL HISTORY
231      hist%iter = iter
232      hist%res = res
233
234
235      ! INCREMENT ITERATION COUNT
236      iter = iter + 1
237
238      END DO iter_loop
239
240      ! HOLD UNTIL ALL PROCESSORS HAVE FINISHED ITERATION LOOP
241      CALL MPI_Barrier(MPI_COMM_WORLD, IERROR)
242
243      ! there was an extra increment after final iteration we need to subtract
244      iter = iter - 1
245
246      IF (MYID == 0) THEN
247
248          ! CALC SOLVER WALL CLOCK TIME
249          end_solve = MPI_Wtime()
250          wall_time_solve = end_solve - start_solve
251
252          IF (iter > max_iter) THEN
253              WRITE(*,*) 'DID NOT CONVERGE (NUMBER OF ITERATIONS:', iter, ')'
254          ELSE
255              WRITE(*,*) 'CONVERGED (NUMBER OF ITERATIONS:', iter, ')'
256              WRITE(*,*) '          (MAXIMUM RESIDUAL      :', res, ')'
257          END IF
258
259      END IF
260      END SUBROUTINE solve
261
262      SUBROUTINE output(blocks, iter)
263          ! Save solution performance parameters to file
264
265          TYPE(BLKTYPE), TARGET :: blocks(:)
266          TYPE(BLKTYPE), POINTER :: b
267          REAL(KIND=8), POINTER :: tmpT(:,,:), tempTemperature(:, :)
268          REAL(KIND=8) :: resloc, resmax
269          INTEGER :: iter, I, J, IBLK, IRES, iresmax, jresmax
270
271      !      Temperature => mesh%T(2:IMAX-1, 2:JMAX-1)
272      !      tempTemperature => mesh%Tmp(2:IMAX-1, 2:JMAX-1)
273
274      ! CALC RESIDUAL

```

```

275     resmax = 0.D0
276 DO IBLK = 1, MYNBLK
277     b => blocks(IBLK)
278     DO J = b%JMINLOC, B%JMAXLOC
279         DO I = b%IMINLOC, b%IMAXLOC
280             resloc = ABS( b%mesh%Tmp(I, J) )
281             IF (resloc > resmax) THEN
282                 ! MAX LOCAL RESIDUAL ON PROC
283                 resmax = resloc
284                 ! Local index of block with max residual
285                 IRES = IBLK
286                 ! local indices of max residual
287                 iresmax = I
288                 jresmax = J
289                 ! Global indices of max residual
290                 iresmax = b%imin + iresmax - 2
291                 jresmax = b%jmin + jresmax - 2
292             END IF
293         END DO
294     END DO
295 END DO
296
297
298
299 !           ! Find max of each block
300 !           resloc = MAXVAL( ABS( blocks(IBLK)%mesh%Tmp(2:IMAXBK-1, 2:JMAXBK-1) ) )
301 !           ! keep biggest residual
302 !           IF (resloc > resmax) THEN
303 !               resmax = resloc
304 !               IRES = IBLK
305 !           END IF
306 !       END DO
307
308 CALL MPI_Barrier(MPI_COMM_WORLD, IERROR)
309 CALL MPI_Bcast(wall_time_solve, 1, MPI_REAL8, 0, mpi_comm_world, ierror)
310
311
312 ! Write final maximum residual and location of max residual
313 ! OPEN(UNIT = 1, FILE = casedir // "SteadySoln.dat")
314 ! DO i = 1, IMAX
315 !     DO j = 1, JMAX
316 !         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))
317 !     END DO
318 ! END DO
319 ! CLOSE (1)
320
321 ! Screen output
322 ! tmpT => blocks(IRES)%mesh%Tmp
323 ! WRITE (*,*), "IMAX/JMAX", IMAX, JMAX
324 ! WRITE (*,*), "N/M", N, M
325 ! WRITE (*,*), "iters", iter
326 ! WRITE (*,*), "max residual", MAXVAL(tmpT(2:IMAXBK-1, 2:JMAXBK-1))
327 ! WRITE (*,*), "on block id", IRES
328 ! WRITE (*,*), "residual ij", MAXLOC(tmpT(2:IMAXBK-1, 2:JMAXBK-1))
329
330
331
332
333
334 ! Write to file
335 IF (MYID == 0) THEN
336     OPEN (UNIT = 2, FILE = "SolnInfo.dat")
337     WRITE (2,*), "Running a", IMAX, "by", JMAX, "grid,"
338     WRITE (2,*), "On", NPROCS, "processors, With NxM:", N, "x", M, "blocks,"
339     WRITE (2,*), iter, "iterations"
340 !     WRITE (2,*), wall_time_total, "seconds (Total CPU walltime)"
341     WRITE (2,*), wall_time_solve, "seconds (Solver CPU walltime)"
342 !     WRITE (2,*), wall_time_iter, "seconds (Iteration CPU walltime)"
343     CLOSE (2)

```

```

344     END IF
345
346     ! WRITE RESIDUAL FOR EACH PROC SEQUESENTIALLY
347     DO I = 0, NPROCS-1
348
349         IF (MYID == I) THEN
350             ! WRITE MAX RESIDUAL/LOCATION FOR EACH PROC
351             tmpT => blocks(IRES)%mesh%Tmp
352             WRITE (*,*)
353             ! WRITE (*,*), "MAX RESIDUAL FOR PROCESSOR ", MYID
354             ! WRITE (*,*), "Found max residual of ", MAXVAL(tmpT(2:IMAXBK-1, 2:JMAXBK-1))
355             ! WRITE (*,*), "on block id", blocks(IRES)%ID
356             ! WRITE (*,*), "At ij of ", MAXLOC(tmpT(2:IMAXBK-1, 2:JMAXBK-1))
357             WRITE (*,*), "MAX RESIDUAL FOR PROCESSOR ", MYID
358             WRITE (*,*), "Found max residual of ", resmax
359             WRITE (*,*), "on block id", blocks(IRES)%ID
360             WRITE (*,*), "At ij of ", iresmax, jresmax
361
362         END IF
363
364         ! WAIT FOR CURRENT PROC TO WRITE
365         CALL MPI_Barrier(MPI_COMM_WORLD, IERROR)
366     END DO
367
368
369
370 END SUBROUTINE output
371
372
373
374 END MODULE subroutines

```

Listing 2: Grid decomposition and solution processes are organized in this code

Appendix C: Parallel, Multi-Block Grid Decomposition Code

```

1  ! MAE 267
2  ! PROJECT 5
3  ! LOGAN HALSTROM
4  ! 29 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
81     IMPLICIT NONE
82
83     ! INCLUDE MPI FOR ALL SUBROUTINES THAT USE CONSTANTS
84     INCLUDE "mpif.h"
85     ! MPI PROCESSOR ID, NUMBER OF BLOCKS PER PROCESSOR
86     ! (initialized by each processor in parallel)
87     INTEGER :: MYID, MYNBLK
88     ! MPI ERROR STATUS, NUMBER OF MPI PROCESSORS
89     INTEGER :: IERROR, NPROCS, request
90     INTEGER :: STATUS(MPI_STATUS_SIZE)
91
92     ! CFL number, for convergence (D0 is double-precision, scientific notation)
93     REAL(KIND=8), PARAMETER :: CFL = 0.95D0
94     ! Material constants (steel): thermal conductivity [W/(m*K)],
95     ! density [kg/m^3],
96     ! specific heat ratio [J/(kg*K)]
97     ! initial temperature
98     REAL(KIND=8), PARAMETER :: k = 18.8D0, rho = 8000.D0, cp = 500.D0, T0 = 3.5D0
99     ! Thermal diffusivity [m^2/s]
100    REAL(KIND=8), PARAMETER :: alpha = k / (cp * rho)

```

```

101 ! Pi, grid rotation angle (30 deg)
102 REAL(KIND=8), PARAMETER :: pi = 3.141592654D0, rot = 30.D0*pi/180.D0
103 ! ITERATION PARAMETERS
104 ! Minimum Residual
105 REAL(KIND=8) :: min_res = 0.00001D0
106 ! Maximum number of iterations
107 INTEGER :: max_iter = 250000
108 !   INTEGER :: max_iter = 1+1
109 ! CPU Wall Times
110 REAL(KIND=8) :: wall_time_total, wall_time_solve, wall_time_iter(1:5)
111 ! read square grid size, Total grid size, size of grid on each block (local)
112 INTEGER :: nx, IMAX, JMAX, IMAXBK, JMAXBK
113 ! Dimensions of block layout, Number of Blocks
114 INTEGER :: M, N, NBLK
115 ! Block boundary condition identifiers
116 ! If boundary is on a different proc, multiply bnd type by proc boundary
117 INTEGER :: BND=0, PROCBND = -1
118 ! boundary indicators for each direction, use for mpi sends
119 INTEGER :: NBND = 1, NEBND=2, EBND = 3, SEBND=4, SBND = 5, SWBND = 6, WBND=7, NWBND=8
120 ! Output directory
121 CHARACTER(LEN=18) :: casedir
122 ! RUN MODE: debug = 0, normal =1, optimizd for 10x10 blocks = 2
123 INTEGER :: OPT
124 ! Value for constant temperature BCs for debugging
125 REAL(KIND=8), PARAMETER :: TDEBUG = T0 - T0 * 0.5
126
127 CONTAINS
128
129 SUBROUTINE read_input()
130 ! Reads grid/block size and other simulation parameters from
131 ! "config.in" file. Avoids recompiling for simple input changes
132
133 INTEGER :: I
134 CHARACTER(LEN=3) :: strNX
135 CHARACTER(LEN=1) :: strN, strM
136
137 ! READ INPUTS FROM FILE
138 ! (So I don't have to recompile each time I change an input setting)
139 ! WRITE(*,*) ''
140 ! WRITE(*,*) 'Reading input...'
141 OPEN (UNIT = 1, FILE = 'config.in')
142 DO I = 1, 3
143 ! Skip header lines
144 READ(1,*)
145 END DO
146 ! READ GRIDSIZE (4th line)
147 READ(1,*) nx
148 ! READ BLOCKS (6th and 8th line)
149 READ(1,*)
150 READ(1,*) N
151 READ(1,*)
152 READ(1,*) M
153 ! DEBUG MODE (10th line)
154 READ(1,*)
155 READ(1,*) OPT
156
157 ! SET GRID SIZE
158 IMAX = nx
159 JMAX = nx
160 ! CALC NUMBER OF BLOCKS
161 NBLK = M * N
162 ! SET SIZE OF EACH BLOCK (LOCAL MAXIMUM I, J)
163 IMAXBK = 1 + (IMAX - 1) / N
164 JMAXBK = 1 + (JMAX - 1) / M
165
166 ! OUTPUT DIRECTORIES
167 ! write integers to strings
168 ! WRITE( strNX, '(I3)') nx
169 ! IF ( N - 10 < 0 ) THEN

```


[illegible]

```

239 ! LOCAL/GLOBAL BLOCK INDICIES
240 !
241 ! GLOBAL
242 ! block (IBLK)%IMIN block (IBLK)%IMAX
243 !
244 ! JMAXBLK -|-----| block (IBLK)%JMAX
245 !
246 !
247 !
248 ! L | | G
249 ! O | | L
250 ! C J^ | LOCAL BLOCK INDICES | O
251 ! A | | B
252 ! L | | A
253 ! | | L
254 !
255 !
256 ! 1 -|-----| block (IBLK)%JMIN
257 !
258 ! 1 I -> IMAXBLK
259 ! LOCAL
260 !
261 ! Where block is block data type, IBLK is index of current block
262 !
263 ! Convert from local to global (where I is local index):
264 ! Iglobal = block (IBLK)%IMIN + (I-1)
265 !
266 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
267 !
268 ! LOCAL BLOCK INDICIES WITH GHOST NODES
269 !
270 ! JMAXBLK+1 |---|-----|---|
271 ! |NWG| NORTH GHOST NODES |NEG|
272 ! JMAXBLK |---|-----|---|
273 !
274 ! | W | | E |
275 ! | E | | A |
276 ! | S | | S |
277 ! | T | | T |
278 ! J^ | | LOCAL BLOCK INDICES | |
279 ! | G | | G |
280 ! | H | | H |
281 ! | O | | O |
282 ! | S | | S |
283 ! | T | | T |
284 ! | | | |
285 ! 1 |---|-----|---|
286 ! |SWG| SOUTH GHOST NODES |SEG|
287 ! 0 |---|-----|---|
288 ! | 1 IMAXBLK |
289 ! 0 I -> IMAXBLK+1
290 !
291 ! Where NWG, NEG, etc are corner ghosts
292 !
293 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
294 !
295 ! BLOCK NEIGHBORS
296 !
297 ! | |
298 ! | North |
299 ! NW| (IBLK + N) |NE
300 ! (IBLK + N - 1)| | (IBLK + N + 1)
301 ! -----
302 ! | |
303 ! West | Current | East
304 ! (IBLK - 1) | (IBLK) | (IBLK + 1)
305 ! | |
306 ! -----
307 ! SW| SE

```

```

308 ! (IBLK - N - 1) |      South      | (IBLK - N + 1)
309 !               | (IBLK - N)      |
310 !               |                  |
311 !
312 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
313 !
314 ! LOCAL ITERATION BOUNDS (TO INCLUDE GHOSTS/EXCLUDE BC'S)
315 ! -----
316 ! | ~ ~ = BC      |
317 ! | . . = Ghost   |
318 ! -----
319 ! JMAXBLK+1 -|---|-----|---|
320 !           | ~ | . . . . . | ~ |
321 ! JMAXBLK -|---|-----|---| JMAXBLK -|---|-----|---|
322 !           | ~ |          | . | ~ | ~ ~ ~ ~ ~ ~ ~ | ~ |
323 !           | ~ |          | . | JMAXBLK-1-|---|-----|---|
324 ! J^         | ~ | M=1, N=1 | . |          | . |          | ~ |
325 !           | ~ |          | . |          | . |          | ~ |
326 !           | ~ |          | . |          | . | M=M, N=N | ~ |
327 !           | ~ |          | . |          | . |          | ~ |
328 !           | ~ | ~ ~ ~ ~ ~ | ~ |          | . |          | ~ |
329 !           | ~ |          | . |          | . |          | ~ |
330 !           | 2 -|---|-----|---|          | 1 -|---|-----|---|
331 !           | 1 -|---|-----|---|          | 0 -|---|-----|---|
332 !           | 1 -|---|-----|---|          | 1 -|---|-----|---|
333 !           | 0 -|---|-----|---|          | 0 -|---|-----|---|
334 !
335 ! Solver : I = 1 --> IMAXBLK          | Solver : I = 0 --> IMAXBLK-1
336 !       to get: dT: 1 --> IMAXBLK+1  |       to get: dT: 0 --> IMAXBLK
337 ! Update T: I = 2 --> IMAXBLK        | Update T: I = 1 --> IMAXBLK-1
338 !       (avoid updating BC's at I=1)  |       (avoid updating BC's at I=IMAXBLK)
339 !       (IMAXBLK+1 ghost updated later) |       (I=0 ghost updated later)
340 !
341 ! RESULT: Set local iteration bounds IMINLOC, IMAXLOC, etc according to solver limits
342 !       Update temperature starting at IMINLOC+1 to avoid lower BC's
343 !       (upper BC's automatically avoided by explicit scheme solving for i+1)
344 !
345 !
346 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
347
348 ! INITIALIZE VARIABLES/DEPENDANCIES
349 USE CONSTANTS
350
351 IMPLICIT NONE
352 PUBLIC
353
354 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
355 !!!!! DERIVED DATA TYPES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
356 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
357
358 ! DERIVED DATA TYPE FOR GRID INFORMATION
359
360 TYPE MESHTYPE
361 ! Grid points, see coordinate rotaion equations in problem statement
362 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xp, yp, x, y
363 ! Temperature at each point, temporary variable to hold temperature sum
364 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: T, Ttmp
365 ! Iteration Parameters: timestep, cell volume, secondary cell volume,
366 ! equation constant term
367 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: dt, V, V2nd, term
368 ! Areas used in alternative scheme to get fluxes for second-derivative
369 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: Ayi, Axi, Ayj, Axj
370 ! Second-derivative weighting factors for alternative distribution scheme
371 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: yPP, yNP, yNN, yPN
372 REAL(KIND=8), ALLOCATABLE, DIMENSION(:, :) :: xNN, xPN, xPP, xNP
373 END TYPE MESHTYPE
374
375 ! DATA TYPE FOR INFORMATION ABOUT NEIGHBORS

```

```

377 TYPE NBRTYPE
378     ! Information about face neighbors (north, east, south, west)
379     ! And corner neighbors (Northeast, southeast, southwest, northwest)
380     INTEGER :: N, E, S, W, NE, SE, SW, NW
381 END TYPE NBRTYPE
382
383 ! DERIVED DATA TYPE WITH INFORMATION PERTAINING TO SPECIFIC BLOCK
384
385 TYPE BLKTYPE
386     ! DER. DATA TYPE STORES LOCAL MESH INFO
387     TYPE(MESHTYPE) :: mesh
388     ! IDENTIFY FACE AND CORNER NEIGHBOR BLOCKS AND PROCESSORS
389     ! AND LOCAL PROCESSOR BLOCK INDICIES
390     TYPE(NBRTYPE) :: NB, NP, NBLOC
391     ! BLOCK NUMBER, PROCESSOR NUMBER
392     INTEGER :: ID, procID
393     ! GLOBAL INDICIES OF MINIMUM AND MAXIMUM INDICIES OF BLOCK
394     INTEGER :: IMIN, IMAX, JMIN, JMAX
395     ! LOCAL ITERATION BOUNDS TO AVOID UPDATING BC'S + UTILIZE GHOST NODES
396     INTEGER :: IMINLOC, JMINLOC, IMAXLOC, JMAXLOC, IMINUPD, JMINUPD
397     ! BLOCK LOAD PARAMETERS FOR PROCESSOR LOAD BALANCING
398     INTEGER :: SIZE
399     ! BLOCK ORIENTATION
400     INTEGER :: ORIENT
401 END TYPE BLKTYPE
402
403 ! DATA TYPE FOR PROCESSOR INFORMATION
404
405 TYPE PROCTYPE
406     ! Information pertaining to each processor: procID, number of blocks
407     ! on proc
408     INTEGER :: ID, NBLK=0
409     ! processor load, load balance
410     INTEGER :: load=0
411     REAL(KIND=8) :: balance=0
412     ! Blocks contained on processor
413     TYPE(BLKTYPE), ALLOCATABLE :: blocks(:)
414 END TYPE PROCTYPE
415
416 ! LINKED LIST: RECURSIVE POINTER THAT POINTS THE NEXT ELEMENT IN THE LIST
417
418 TYPE LNKLIST
419     ! Next element in linked list
420     TYPE(LNKLIST), POINTER :: next
421     ! Identify what linked list belongs to
422     INTEGER :: ID
423 END TYPE LNKLIST
424
425 ! Collection of linked lists for faces and corners
426
427 TYPE NBRLIST
428     TYPE(LNKLIST), POINTER :: N, E, S, W, NE, SE, SW, NW
429 END TYPE NBRLIST
430
431 CONTAINS
432
433 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
434 !!! INITIALIZE GRID AND WRITE TO FILE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
435 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
436
437 SUBROUTINE init_blocks(b)
438     ! Assign individual block global indicies, neighbor, BCs, and
439     ! orientation information
440
441     ! BLOCK DATA TYPE
442     TYPE(BLKTYPE), TARGET :: b(:)
443     ! Neighbor information pointer
444     TYPE(NBRTYPE), POINTER :: NB
445     ! COUNTER VARIABLES

```

```

446      ! IM, IN COUNT BLOCK INDICIES
447      ! (IBLK COUNTS BLOCK NUMBERS, INBR IS BLOCK NEIGHBOR INDEX)
448      INTEGER :: I, J, IBLK, INBR
449
450      ! STEP THROUGH BLOCKS, ASSIGN IDENTIFYING INFO
451
452      ! START AT BLOCK 1 (INCREMENT IN LOOP)
453      IBLK = 0
454
455      DO J = 1, M
456          DO I = 1, N
457              ! INCREMENT BLOCK NUMBER
458              IBLK = IBLK + 1
459
460              ! Neighbor information pointer
461              NB => b(IBLK)%NB
462
463              ! ASSIGN BLOCK NUMBER
464              b(IBLK)%ID = IBLK
465              ! ASSIGN GLOBAL MIN/MAX INDICIES OF LOCAL GRID
466              b(IBLK)%IMIN = 1 + (IMAXBK - 1) * (I - 1)
467              b(IBLK)%JMIN = 1 + (JMAXBK - 1) * (J - 1)
468              b(IBLK)%IMAX = b(IBLK)%IMIN + (IMAXBK - 1)
469              b(IBLK)%JMAX = b(IBLK)%JMIN + (JMAXBK - 1)
470
471              ! ASSIGN NEIGHBORS
472              ! (Numbers of face and corner neighbor blocks)
473              ! (if boundary face, assign bc later)
474              NB%N = IBLK + N
475              NB%S = IBLK - N
476              NB%E = IBLK + 1
477              NB%W = IBLK - 1
478              NB%NE = IBLK + N + 1
479              NB%NW = IBLK + N - 1
480              NB%SW = IBLK - N - 1
481              NB%SE = IBLK - N + 1
482
483              ! ASSIGN BOUNDARY CONDITIONS
484
485              ! Assign faces and corners on boundary of the actual
486              ! computational grid with number corresponding to which
487              ! boundary they are on.
488              ! Corners on actual corners of the computational grid are
489              ! ambiguously assigned.
490              IF ( b(IBLK)%JMAX == JMAX ) THEN
491                  ! NORTH BLOCK FACE AND CORNERS ARE ON MESH NORTH BOUNDARY
492                  ! AT ACTUAL CORNERS OF MESH, CORNERS ARE AMBIGUOUS
493                  NB%N = BND
494                  NB%NE = BND
495                  NB%NW = BND
496              END IF
497              IF ( b(IBLK)%IMAX == IMAX ) THEN
498                  ! EAST BLOCK FACE IS ON MESH EAST BOUNDARY
499                  NB%E = BND
500                  NB%NE = BND
501                  NB%SE = BND
502
503              END IF
504              IF ( b(IBLK)%JMIN == 1 ) THEN
505                  ! SOUTH BLOCK FACE IS ON MESH SOUTH BOUNDARY
506                  NB%S = BND
507                  NB%SE = BND
508                  NB%SW = BND
509              END IF
510              IF ( b(IBLK)%IMIN == 1 ) THEN
511                  ! WEST BLOCK FACE IS ON MESH WEST BOUNDARY
512                  NB%W = BND
513                  NB%SW = BND
514                  NB%NW = BND

```

```

515         END IF
516
517         ! BLOCK ORIENTATION
518         ! same for all in this project
519         b(IBLK)%ORIENT = 1
520
521     END DO
522 END DO
523 END SUBROUTINE init_blocks
524
525 SUBROUTINE dist_blocks(blocks, procs)
526     ! Distribute blocks to processors. Calculate processor load of each
527     ! block based on geometry and communication costs and weighting factors
528     ! for each.
529     ! Initialize processor list with proc ID's and allocate proc block lists
530     ! Distribute blocks to processors by sorting blocks in decreasing order
531     ! of load, then distributing sequentially to the processor with the
532     ! least load.
533     ! Calculate load balance of all processors.
534
535     ! BLOCK DATA TYPE
536     TYPE(BLKTYPE), TARGET :: blocks(:)
537     TYPE(BLKTYPE), POINTER :: b
538     TYPE(NBRTYPE), POINTER :: NB
539     ! PROCESSOR DATA TYPE
540     TYPE(PROCTYPE), TARGET :: procs(:)
541     TYPE(PROCTYPE), POINTER :: p
542     ! COUNTER VARIABLE
543     INTEGER :: IBLK, I, IPROC, II
544     ! CURRENT BLOCK DIMENSIONS
545     INTEGER :: NXLOC, NYLOC
546     ! COMPUTATIONAL COST PARAMETERS
547     ! (geometric (grid size) and communication weights)
548     INTEGER :: GEOM=0, COMM=0, MAXCOMM, MAXGEOM
549     ! WEIGHTS FOR LOAD BALANCING
550     ! (geometry, communication, fudge factor)
551     REAL(KIND=8) :: WGEOM = 1.0D0, WCOMM, FACTOR=1.0D0
552     ! Perfect load balance
553     INTEGER :: PLB = 0
554     ! VARIABLES FOR SORTING BLOCKS BY LOAD
555     ! maximum block load
556     INTEGER :: MAXSIZE=0, MINLOAD
557     ! 'sorted' is list of IDs of blocks in order of size greatest to least
558     ! 'claimed' indicates if a block has already been sorted (0/1 --> unsorted/sorted)
559     ! initial list is all zeros
560     ! 'IMAXSIZE' is index of remaining block with greatest size
561     INTEGER :: sorted(NBLK), claimed(NBLK), IMAXSIZE, IMINLOAD
562     INTEGER :: locIDs(NBLK), proclist(NBLK), idsSort(Nblk), procSort(NBLK)
563     ! OPTIMIZED DISTRIBUTION
564     INTEGER :: METHOD
565
566     ! INITIALIZE LISTS
567     DO I = 1, NBLK
568         claimed(I) = 0
569         sorted(I) = 0
570     END DO
571
572     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
573     !!! SET WEIGHTING FACTORS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
574     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
575
576     ! SET COMMUNICATION WEIGHT TO BE PROPORTIONAL TO GEOMETRY
577     ! Maximum geometry cost is all cells with ghost nodes at all faces
578     MAXGEOM = ( IMAXBLK + 2 ) * ( JMAXBLK + 2 )
579     ! Maximum communication cost is all face boundaries plus four corners
580     MAXCOMM = ( 2 * IMAXBLK ) + ( 2 * JMAXBLK ) + 4
581     ! Put comm cost on same scale as geom
582     WCOMM = FACTOR * ( DFLOAT(MAXGEOM) / DFLOAT(MAXCOMM) )
583     ! COME UP WITH A BETTER WEIGHTING FACTOR IN PROJECT 5 WHEN YOU CAN BENCHMARK TIMES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

```

10      FORMAT(10A12)
585
586 WRITE(*,*)
587 WRITE(*,*) 'Processor Load Weighting Factors:'
588 WRITE(*,*) 'WGEOM=', WGEOM, 'WCOMM=', WCOMM
589 WRITE(*,*)
590 WRITE(*,*) 'SIZE = WGEOM*GEOM + WCOMM*COMM'
591 WRITE(*,*)
592 WRITE(*,*) 'Block Load Factors:'
593 WRITE(*,10) 'BLKID', 'GEOM', 'COMM', 'SIZE'
594
595 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
596 !!! CALC BLOCK WEIGHTS FOR PROCESSOR LOAD BALANCING !!!!!!!!!!!!!!!
597 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
598
599 ! need local block sizes
600 CALL set_block_bounds(blocks)
601 DO IBLK = 1, NBLK
602     b => blocks(IBLK)
603     NB => b%NB
604
605     ! RESET COST SUMS
606     GEOM = 0
607     COMM = 0
608
609     ! LOCAL BLOCK DIMENSIONS
610     NXLOC = b%IMAXLOC - b%IMINLOC
611     NYLOC = b%JMAXLOC - b%JMINLOC
612
613     ! GEOMETRIC BLOCK WEIGHT ("VOLUME")
614     GEOM = NXLOC * NYLOC
615
616     ! COMMUNICATION BLOCK WEIGHT
617     ! NORTH
618     IF (NB%N > 0) THEN
619         ! Interior faces have communication cost for populating ghosts
620         COMM = COMM + IMAXBK
621     END IF
622     ! EAST
623     IF (NB%E > 0) THEN
624         COMM = COMM + JMAXBK
625     END IF
626     ! SOUTH
627     IF (NB%S > 0) THEN
628         COMM = COMM + IMAXBK
629     END IF
630     ! WEST
631     IF (NB%W > 0) THEN
632         COMM = COMM + JMAXBK
633     END IF
634     ! NORTHEAST
635     IF (NB%N > 0) THEN
636         ! Interior corners have communication cost for populating ghosts
637         COMM = COMM + 1
638     END IF
639     ! SOUTHEAST
640     IF (NB%E > 0) THEN
641         COMM = COMM + 1
642     END IF
643     ! SOUTHWEST
644     IF (NB%S > 0) THEN
645         COMM = COMM + 1
646     END IF
647     ! NORTHWEST
648     IF (NB%W > 0) THEN
649         COMM = COMM + 1
650     END IF
651
652     ! CALCULATE TOTAL LOAD OF BLOCK WITH WEIGHTING FACTORS

```

```

b%SIZE = INT( WGEOM * DFLOAT(GEOM) + WCOMM * DFLOAT(COMM) )

! WRITE BLOCK LOADS
WRITE(*,*) IBLK, GEOM, COMM, b%SIZE

! SUM BLOCK LOADS
PLB = PLB + b%SIZE
END DO

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! CALC OPTIMAL LOAD DISTRIBUTION (PERFECT LOAD BALANCE) !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! (total load of all blocks divided by number of processors)
PLB = INT( DFLOAT(PLB) / DFLOAT(NPROCS) )

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! SORT BLOCKS BY LOAD IN DECREASING ORDER !!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO IBLK = 1, NBLK

    ! Reset current max size
    MAXSIZE = 0

    ! FIND MAX SIZE OF REMAINING BLOCKS
    DO I = 1, NBLK
        b => blocks(I)

        ! (all sorted blocks will be excluded by 'claimed')
        IF (claimed(I)==0 .AND. b%SIZE>=MAXSIZE) THEN
            ! CURRENT BLOCK HAS GREATEST LOAD SIZE OF REMAINING BLOCKS
            MAXSIZE = b%SIZE
            ! INDEX OF MAX REMAINING SIZE BLOCK
            IMAXSIZE = I
        END IF
    END DO

    ! MARK LATEST MAX AS SORTED (so it doesn't come up again)
    claimed(IMAXSIZE) = 1
    ! ADD INDEX OF LATEST MAX TO SORTED INDEX LIST
    sorted(IBLK) = IMAXSIZE
END DO

! write block size order
write(*,*) " "
write(*,*) "Blocks ordered by size, greatest to least, with sizes:"
DO I = 1, NBLK
    b => blocks( sorted(I) )
    write(*,*) b%ID, b%SIZE
END DO
write(*,*) " "

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! INITIALIZE PROCS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

DO IPROC = 1, NPROCS
    ! SET EACH PROCESSOR'S ID
    ! (Processor indexing starts at zero)
    procs(IPROC)%ID = IPROC-1
    ! ALLOCATE BLOCK LISTS FOR EACH PROC
    ! (Make them NBLK long even though they will contain less than that
    ! so we dont have to reallocate)
    ALLOCATE( procs(IPROC)%blocks(NBLK) )
END DO

IF (OPT == 2) THEN

```



```

722      !!! HARDCODED OPTIMIZED DIST FOR 10X10 BLOCKS !!!!!!!!!!!!!!!!!!!!!
723      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
724
725      ! DISTRIBUTION OPTIONS
726      IF (NPROCS == 10) THEN
727
728          ! 10X10 BLOCKS, 10 PROCS
729
730          ! ASSIGN EACH ROW TO ONE INDIVIDUAL PROCESSOR
731          DO IPROC = 0, NPROCS-1
732              DO I = 1, 10
733                  CALL assign_block( blocks( I + IPROC*10 ), procs(IPROC) )
734              END DO
735          END DO
736
737      ELSE IF (NPROCS == 8) THEN
738          ! 10X10 BLOCKS, 8 PROCS
739
740          ! TRY TO ASSIGN AS MANY PROCESSORS IN ONE ROW TO ONE PROC
741          ! (assign sequestentially)
742          DO I = 1, 13
743              CALL assign_block( blocks(I), procs(1) )
744          END DO
745          DO I = 14, 25
746              CALL assign_block( blocks(I), procs(2) )
747          END DO
748          DO I = 26, 37
749              CALL assign_block( blocks(I), procs(3) )
750          END DO
751          DO I = 38, 50
752              CALL assign_block( blocks(I), procs(4) )
753          END DO
754          DO I = 51, 63
755              CALL assign_block( blocks(I), procs(5) )
756          END DO
757          DO I = 64, 75
758              CALL assign_block( blocks(I), procs(6) )
759          END DO
760          DO I = 76, 87
761              CALL assign_block( blocks(I), procs(7) )
762          END DO
763          DO I = 88, 100
764              CALL assign_block( blocks(I), procs(8) )
765          END DO
766
767      ELSE IF (NPROCS == 6) THEN
768          ! 10X10 BLOCKS, 6 PROCS
769
770          ! ASSIGN FIRST 3 ROWS TO PROCS 1 & 2,
771          ! DIVIDED IN HALF (PROC1 LEFT, PROC2 RIGHT)
772          DO I = 0, 2
773              DO II = 1, 5
774                  CALL assign_block( blocks( II + I*10 ), procs(1) )
775              END DO
776              DO II = 6, 10
777                  CALL assign_block( blocks( II + I*10 ), procs(2) )
778              END DO
779          END DO
780          ! GIVE CENTER BLOCKS OF 4TH ROW TO PROCS 1 & 2,
781          ! 2 TO EACH PROC ON EACH PROC'S SIDE
782          ! left 2 center blocks to proc 1
783          CALL assign_blocks( blocks, procs(1), [34, 35] )
784          ! right 2 center blocks to proc 2
785          CALL assign_blocks( blocks, procs(2), [36, 37] )
786          ! GIVE EDGE BLOCKS OF 4TH ROW TO PROCS 3 & 4
787          ! leftmost 3 blocks to proc 3
788          CALL assign_blocks( blocks, procs(3), [31, 32, 33] )
789          ! rightmost 3 blocks to proc 4
790          CALL assign_blocks( blocks, procs(4), [38, 39, 40] )

```

```

791      ! ASSIGN 5TH AND 6TH ROWS TO PROCS 3 & 4,
792      ! DIVIDED IN HALF
793      DO I = 4, 5
794          DO II = 1, 5
795              CALL assign_block( blocks( II + I*10 ), procs(3) )
796          END DO
797          DO II = 6, 10
798              CALL assign_block( blocks( II + I*10 ), procs(4) )
799          END DO
800      END DO
801      ! GIVE EDGE BLOCKS OF 7TH ROW TO PROCS 3 & 4
802      ! leftmost 3 blocks to proc 3
803      CALL assign_blocks( blocks, procs(3), [61, 62, 63] )
804      ! rightmost 3 blocks to proc 4
805      CALL assign_blocks( blocks, procs(4), [68, 69, 70] )
806      ! GIVE CENTER BLOCKS OF 7TH ROW TO PROCS 5 & 6,
807      ! left 2 center blocks to proc 5
808      CALL assign_blocks( blocks, procs(5), [64, 65] )
809      ! right 2 center blocks to proc 6
810      CALL assign_blocks( blocks, procs(6), [66, 67] )
811      ! ASSIGN LAST 3 ROWS TO PROCS 5 & 6, DIVIDED IN HALF
812      DO I = 7, 9
813          DO II = 1, 5
814              CALL assign_block( blocks( II + I*10 ), procs(5) )
815          END DO
816          DO II = 6, 10
817              CALL assign_block( blocks( II + I*10 ), procs(6) )
818          END DO
819      END DO
820
821      ELSE IF (NPROCS == 4) THEN
822          ! 10X10 BLOCKS, 4 PROCS
823          ! DIVIDE GRID INTO QUADRANTS
824
825          ! ASSIGN BOTTOM CORNERS TO PROCS 1 AND 2
826          DO I = 0, 4
827              DO II = 1, 5
828                  CALL assign_block( blocks( II + I*10 ), procs(1) )
829              END DO
830              DO II = 6, 10
831                  CALL assign_block( blocks( II + I*10 ), procs(2) )
832              END DO
833          END DO
834          ! ASSIGN TOP CORNERS TO PROCS 3 AND 4
835          DO I = 5, 9
836              DO II = 1, 5
837                  CALL assign_block( blocks( II + I*10 ), procs(3) )
838              END DO
839              DO II = 6, 10
840                  CALL assign_block( blocks( II + I*10 ), procs(4) )
841              END DO
842          END DO
843
844      ELSE IF (NPROCS == 2) THEN
845          ! 10X10 BLOCKS, 2 PROCS
846          ! DIVIDE GRID IN HALF (TOP AND BOTTOM)
847
848          ! ASSIGN BOTTOM BLOCKS TO PROC 1
849          DO I = 0, 4
850              DO II = 1, 10
851                  CALL assign_block( blocks( II + I*10 ), procs(1) )
852              END DO
853          END DO
854          ! ASSIGN TOP BLOCKS TO PROC 2
855          DO I = 5, 9
856              DO II = 1, 10
857                  CALL assign_block( blocks( II + I*10 ), procs(2) )
858              END DO
859          END DO

```

```

860
861     END IF
862
863 ELSE
864     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
865     !!! DISTRIBUTE TO PROCESSOR WITH LEAST LOAD !!!!!!!!!!!!!!!!!!!!!!!!!!!
866     !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
867
868     write(*,*) " "
869     write(*,*) "Block ID assigned to Proc ID:"
870
871     ! LOOP THROUGH BLOCKS IN DECREASING ORDER OF SIZE
872     DO I = 1, NBLK
873         ! sorted gives the indicies of blocks sorted by size
874         b => blocks( sorted(I) )
875
876         ! Reset minimum load
877         MINLOAD = 1000000
878         ! FIND CURRENT PROCESSOR WITH LEAST LOAD
879         DO IPROC = 1, NPROCS
880             p => procs(IPROC)
881
882             IF (p%load<MINLOAD) THEN
883                 MINLOAD = p%load
884                 IMINLOAD = IPROC
885             END IF
886         END DO
887         ! write block processor assignment
888         write(*,*) b%ID, procs(IMINLOAD)%ID
889         proclist(I) = procs(IMINLOAD)%ID
890         locIDs(I) = procs(IMINLOAD)%NBLK+1
891
892         ! ASSIGN BLOCK TO MIN. LOAD PROC
893         CALL assign_block( b, procs(IMINLOAD) )
894     END DO
895
896     ! CALC LOAD BALANCE
897     20      FORMAT(10A13)
898     WRITE(*,*)
899     WRITE(*,*) 'Processor Load Balancing:'
900     WRITE(*,20) 'ID', 'LOAD BALANCE'
901
902     DO IPROC = 1, NPROCS
903         procs(IPROC)%balance = DFLOAT( procs(IPROC)%load ) / DFLOAT( PLB )
904
905         WRITE(*,*) procs(IPROC)%ID, procs(IPROC)%balance
906     END DO
907     WRITE(*,*)
908
909
910
911     do Iblk = 1, Nblk
912         do i = 1, nblk
913             if (sorted(I) == IBLK) then
914                 idsSort(Iblk) = locIDs(I)
915                 procSort(Iblk) = proclist(I)
916             end if
917         end do
918     end do
919
920
921     ! write block amalgamation file
922     OPEN(UNIT=55,FILE = 'blockrebuild.dat',FORM='formatted')
923     write(55,*) "block, processor, local id"
924     do I = 1, NBLK
925         write(55,*) I, procsort(I), IDsSort(I)
926     end do
927     CLOSE(55)
928

```

```

929         OPEN(UNIT=65, FILE = 'procrebuild.dat', FORM='formatted')
930         do I = 1, NProcs
931             write(65,*) procs(I)%NBLK
932         end do
933         CLOSE(65)
934
935     END IF
936
937 END SUBROUTINE dist_blocks
938
939 SUBROUTINE assign_block(b, p)
940     ! Assign block to given processor
941
942     ! Block to assign (not list)
943     TYPE(BLKTYPE), TARGET :: b
944     ! Processor to assign to
945     TYPE(PROCTYPE), TARGET :: p
946
947     ! INCREMENT NUMBER OF BLOCKS ON PROC
948     p%NBLK = p%NBLK + 1
949     ! ADD BLOCK LOAD TO TOTAL PROCESSOR LOAD
950     p%load = p%load + b%SIZE
951     ! ADD BLOCK TO PROC
952     p%blocks(p%NBLK) = b
953     ! ADD BLOCK TO PROC
954     p%blocks(p%NBLK) = b
955
956 END SUBROUTINE assign_block
957
958 SUBROUTINE assign_blocks(blocks, proc, IDs)
959     ! Like assign_blocks, but assign multiple blocks
960
961     ! all blocks
962     TYPE(BLKTYPE), TARGET :: blocks(:)
963     ! Processor to assign to
964     TYPE(PROCTYPE), TARGET :: proc
965     ! IDs of blocks to assign
966     INTEGER :: IDs(:), I
967
968     ! assign each block in IDs to proc
969     DO I = 1, SIZE(IDs)
970         CALL assign_block( blocks( IDS(I) ), proc )
971     END DO
972 END SUBROUTINE assign_blocks
973
974
975 SUBROUTINE init_neighbor_procs(blocks, procs)
976     ! Initialize neighbor processor information for each block
977
978     ! BLOCK DATA TYPE
979     TYPE(BLKTYPE), TARGET :: blocks(:)
980     TYPE(BLKTYPE), POINTER :: bcur, bnbr
981     ! PROCESSOR DATA TYPE
982     TYPE(PROCTYPE), TARGET :: procs(:)
983     TYPE(PROCTYPE), POINTER :: pcur, pnbr
984     ! Neighbor information pointer
985     TYPE(NBRTYPE), POINTER :: NBCUR, NBNBR, NPCUR
986     ! COUNTER VARIABLES
987     ! index of current processor, index of current proc's neighbor proc
988     INTEGER :: IPCUR, IPNBR, IBCUR, IBNBR
989
990     ! ASSIGN PROC INFORMATION TO PROC DATA TYPE LIST
991     DO IPCUR = 1, NPROCS
992         pcur => procs(IPCUR)
993
994         ! ALL BLOCKS ASSIGNED TO CURRENT PROC ARE ON CURRENT PROC
995         pcur%blocks%procID = pcur%ID
996         ! DEFAULT ALL NEIGHBORS TO -1
997         ! (indicates boundary with no neighbor if not reassigned later)

```

```

998      DO IBCUR = 1, pcur%NBLK
999          NPCUR => pcur%blocks(IBCUR)%NP
1000
1001          NPCUR%N = -1
1002          NPCUR%S = -1
1003          NPCUR%E = -1
1004          NPCUR%W = -1
1005          NPCUR%NE = -1
1006          NPCUR%SE = -1
1007          NPCUR%SW = -1
1008          NPCUR%NW = -1
1009      END DO
1010  END DO
1011
1012  ! INITIALIZE LOCAL PROCESSOR INDICIES OF BLOCKS
1013  DO IPCUR = 1, NPROCS
1014      pcur => procs(IPCUR)
1015      DO IBCUR = 1, pcur%NBLK
1016          bcur => pcur%blocks(IBCUR)
1017          bcur%NBLOC%N = 0
1018          bcur%NBLOC%S = 0
1019          bcur%NBLOC%E = 0
1020          bcur%NBLOC%W = 0
1021          bcur%NBLOC%NE = 0
1022          bcur%NBLOC%SE = 0
1023          bcur%NBLOC%SW = 0
1024          bcur%NBLOC%NW = 0
1025      END DO
1026  END DO
1027
1028  ! FIND PROC WITH NEIGHBOR FOR EACH BLOCK
1029  DO IPCUR = 1, NPROCS
1030      pcur => procs(IPCUR)
1031
1032      ! FOR EACH PROC, STEP THROUGH EACH CONTAINED BLOCK AND FIND NEIGHBORS
1033      DO IBCUR = 1, pcur%NBLK
1034          bcur => pcur%blocks(IBCUR)
1035
1036          ! STEP THROUGH EACH NEIGHBOR PROCESSOR TO FIND NEIGHBOR BLOCK
1037          DO IPNBR = 1, NPROCS
1038              pnbr => procs(IPNBR)
1039
1040              ! STEP THROUGH BLOCKS ON NEIGHBOR PROCESSORS
1041              DO IBNBR = 1, pnbr%NBLK
1042                  bnbr => pnbr%blocks(IBNBR)
1043
1044                  ! CHECK EACH FACE/CORNER FOR MATCH AND ASSIGN
1045                  ! (neighbor procID and local index of
1046                  !   neighbor block on neighbor proc)
1047
1048                  ! NORTH
1049                  IF (bcur%NB%N == bnbr%ID) THEN
1050                      ! PROCESSOR CONTAINING NEIGHBOR BLOCK
1051                      bcur%NP%N = pnbr%ID
1052                      ! NEIGHBOR BLOCK LOCAL INDEX ON NEIGHBOR PROCESSOR
1053                      ! (used to access neighbor on neighbor processor)
1054                      bcur%NBLOC%N = IBNBR
1055
1056                      ! IF NEIGHBOR PROC IS DIFFERENT FROM CURRENT PROC,
1057                      ! COMMUNICATION WILL BE REQUIRED
1058                      ! (indicate processor boundary by making the block
1059                      !   neighbor number negative)
1060                      IF (pcur%ID /= pnbr%ID) THEN
1061                          bcur%NB%N = -bcur%NB%N
1062                      END IF
1063                  END IF
1064                  ! SOUTH
1065                  IF (bcur%NB%S == bnbr%ID) THEN
1066                      bcur%NP%S = pnbr%ID

```

```

1067         bcur%NBLOC%S = IBNBR
1068         IF (pcur%ID /= pnbr%ID) THEN
1069             bcur%NB%S = -bcur%NB%S
1070         END IF
1071     END IF
1072     ! EAST
1073     IF (bcur%NB%E == bnbr%ID) THEN
1074         bcur%NP%E = pnbr%ID
1075         bcur%NBLOC%E = IBNBR
1076         IF (pcur%ID /= pnbr%ID) THEN
1077             bcur%NB%E = -bcur%NB%E
1078         END IF
1079     END IF
1080     ! WEST
1081     IF (bcur%NB%W == bnbr%ID) THEN
1082         bcur%NP%W = pnbr%ID
1083         bcur%NBLOC%W = IBNBR
1084         IF (pcur%ID /= pnbr%ID) THEN
1085             bcur%NB%W = -bcur%NB%W
1086         END IF
1087     END IF
1088     ! NORTH EAST
1089     IF (bcur%NB%NE == bnbr%ID) THEN
1090         bcur%NP%NE = pnbr%ID
1091         bcur%NBLOC%NE = IBNBR
1092         IF (pcur%ID /= pnbr%ID) THEN
1093             bcur%NB%NE = -bcur%NB%NE
1094         END IF
1095     END IF
1096     ! SOUTH EAST
1097     IF (bcur%NB%SE == bnbr%ID) THEN
1098         bcur%NP%SE = pnbr%ID
1099         bcur%NBLOC%SE = IBNBR
1100         IF (pcur%ID /= pnbr%ID) THEN
1101             bcur%NB%SE = -bcur%NB%SE
1102         END IF
1103     END IF
1104     ! SOUTH WEST
1105     IF (bcur%NB%SW == bnbr%ID) THEN
1106         bcur%NP%SW = pnbr%ID
1107         bcur%NBLOC%SW = IBNBR
1108         IF (pcur%ID /= pnbr%ID) THEN
1109             bcur%NB%SW = -bcur%NB%SW
1110         END IF
1111     END IF
1112     ! NORTH WEST
1113     IF (bcur%NB%NW == bnbr%ID) THEN
1114         bcur%NP%NW = pnbr%ID
1115         bcur%NBLOC%NW = IBNBR
1116         IF (pcur%ID /= pnbr%ID) THEN
1117             bcur%NB%NW = -bcur%NB%NW
1118         END IF
1119     END IF
1120 END DO
1121 END DO
1122 END DO
1123 END DO
1124
1125 !     10         FORMAT(10A12)
1126 !     WRITE(*,*)
1127 !     WRITE(*,*) 'Check proc neighbors'
1128 !     WRITE(*,10) 'BLKID', 'NB%N', 'NP%N', 'NBLOC%N'
1129 !     DO IPCUR = 1, NPROCS
1130 !         pcur => procs(IPCUR)
1131 !         WRITE(*,*) 'Proc:', pcur%ID
1132 !         DO IBCUR = 1, pcur%NBLK
1133 !             bcur => pcur%blocks(IBCUR)
1134 !             WRITE(*,*) bcur%ID, bcur%NB%N, bcur%NP%N, bcur%NBLOC%N
1135 !         END DO

```

```

1136 !           END DO
1137
1138 END SUBROUTINE init_neighbor_procs
1139
1140 SUBROUTINE init_mesh(b)
1141     ! Create xprime/yprime non-uniform grid, then rotate by angle 'rot'.
1142     ! Allocate arrays for node parameters (i.e. temperature, cell area, etc)
1143
1144     ! BLOCK DATA TYPE
1145     TYPE(BLKTYPE), TARGET :: b(:)
1146     TYPE(MESHTYPE), POINTER :: m
1147     INTEGER :: IBLK, I, J
1148
1149     DO IBLK = 1, NBLK
1150
1151         m => b(IBLK)%mesh
1152
1153         ! ALLOCATE MESH INFORMATION
1154         ! ADD EXTRA INDEX AT BEGINNING AND END FOR GHOST NODES
1155         ALLOCATE( m%xp( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1156         ALLOCATE( m%yp( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1157         ALLOCATE( m%x( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1158         ALLOCATE( m%y( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1159         ALLOCATE( m%T( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1160         ALLOCATE( m%Ttmp(0:IMAXBLK+1, 0:JMAXBLK+1) )
1161         ALLOCATE( m%dt( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1162         ALLOCATE( m%V2nd(0:IMAXBLK+1, 0:JMAXBLK+1) )
1163         ALLOCATE( m%term(0:IMAXBLK+1, 0:JMAXBLK+1) )
1164         ALLOCATE( m%Ayi( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1165         ALLOCATE( m%Axi( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1166         ALLOCATE( m%Ayj( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1167         ALLOCATE( m%Axj( 0:IMAXBLK+1, 0:JMAXBLK+1) )
1168         ALLOCATE( m%V( 0:IMAXBLK, 0:JMAXBLK ) )
1169         ALLOCATE( m%yPP( 0:IMAXBLK, 0:JMAXBLK ) )
1170         ALLOCATE( m%yNP( 0:IMAXBLK, 0:JMAXBLK ) )
1171         ALLOCATE( m%yNN( 0:IMAXBLK, 0:JMAXBLK ) )
1172         ALLOCATE( m%yPN( 0:IMAXBLK, 0:JMAXBLK ) )
1173         ALLOCATE( m%xNN( 0:IMAXBLK, 0:JMAXBLK ) )
1174         ALLOCATE( m%xPN( 0:IMAXBLK, 0:JMAXBLK ) )
1175         ALLOCATE( m%xPP( 0:IMAXBLK, 0:JMAXBLK ) )
1176         ALLOCATE( m%xNP( 0:IMAXBLK, 0:JMAXBLK ) )
1177
1178         ! STEP THROUGH LOCAL INDICIES OF EACH BLOCK
1179         DO J = 0, JMAXBLK+1
1180             DO I = 0, IMAXBLK+1
1181                 ! MAKE SQUARE GRID
1182                 ! CONVERT FROM LOCAL TO GLOBAL INDEX:
1183                 ! Iglobal = Block%IMIN + (Ilocal - 1)
1184                 m%xp(I, J) = COS( 0.5D0 * PI * DFLOAT(IMAX - ( b(IBLK)%IMIN + I - 1) ) / DFLOAT(IMAX - 1) )
1185                 m%yp(I, J) = COS( 0.5D0 * PI * DFLOAT(JMAX - ( b(IBLK)%JMIN + J - 1) ) / DFLOAT(JMAX - 1) )
1186
1187                 ! ROTATE GRID
1188                 m%x(I, J) = m%xp(I, J) * COS(rot) + (1.D0 - m%yp(I, J) ) * SIN(rot)
1189                 m%y(I, J) = m%yp(I, J) * COS(rot) + ( m%xp(I, J) ) * SIN(rot)
1190             END DO
1191         END DO
1192     END DO
1193 END SUBROUTINE init_mesh
1194
1195 SUBROUTINE init_temp(blocks)
1196     ! Initialize temperature across mesh with dirichlet BCs
1197     ! or constant temperature BCs for OPT=0
1198
1199     ! BLOCK DATA TYPE
1200     TYPE(BLKTYPE), TARGET :: blocks(:)
1201     TYPE(BLKTYPE), POINTER :: b
1202     TYPE(MESHTYPE), POINTER :: m
1203     TYPE(NBRTYPE), POINTER :: NB
1204     INTEGER :: IBLK, I, J

```

```

1205
1206 DO IBLK = 1, NBLK
1207     b => blocks(IBLK)
1208     m => blocks(IBLK)%mesh
1209     NB => blocks(IBLK)%NB
1210     ! FIRST, INITIALIZE ALL POINT TO INITIAL TEMPERATURE (T0)
1211     m%T(0:IMAXBK+1, 0:JMAXBK+1) = T0
1212     ! THEN, INITIALIZE BOUNDARIES DIRICHLET B.C.
1213     IF (OPT /= 0) THEN
1214
1215         ! DIRICHLET B.C.
1216         ! face on north boundary
1217         IF (NB%N == BND) THEN
1218             DO I = 1, IMAXBK
1219                 m%T(I, JMAXBK) = 5.D0 * (SIN(PI * m%xp(I, JMAXBK)) + 1.D0)
1220             END DO
1221         END IF
1222         IF (NB%S == BND) THEN
1223             DO I = 1, IMAXBK
1224                 m%T(I, 1) = ABS(COS(PI * m%xp(I, 1))) + 1.D0
1225             END DO
1226         END IF
1227         IF (NB%E == BND) THEN
1228             DO J = 1, JMAXBK
1229                 m%T(IMAXBK, J) = 3.D0 * m%yp(IMAXBK, J) + 2.D0
1230             END DO
1231         END IF
1232         IF (NB%W == BND) THEN
1233             DO J = 1, JMAXBK
1234                 m%T(1, J) = 3.D0 * m%yp(1, J) + 2.D0
1235             END DO
1236         END IF
1237
1238     ELSE
1239
1240         ! DEBUG BCS
1241         IF (NB%N == BND) THEN
1242             DO I = 1, IMAXBK
1243                 m%T(I, JMAXBK) = TDEBUG
1244             END DO
1245         END IF
1246         IF (NB%S == BND) THEN
1247             DO I = 1, IMAXBK
1248                 m%T(I, 1) = TDEBUG
1249             END DO
1250         END IF
1251         IF (NB%E == BND) THEN
1252             DO J = 1, JMAXBK
1253                 m%T(IMAXBK, J) = TDEBUG
1254             END DO
1255         END IF
1256         IF (NB%W == BND) THEN
1257             DO J = 1, JMAXBK
1258                 m%T(1, J) = TDEBUG
1259             END DO
1260         END IF
1261     END IF
1262 END DO
1263 END SUBROUTINE init_temp
1264
1265 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1266 !!! INITIALIZE SOLUTION AFTER RESTART FILE READ IN !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1267 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1268
1269 SUBROUTINE set_block_bounds(blocks)
1270     ! Calculate iteration bounds for each block to avoid overwriting BCs.
1271     ! Call after reading in mesh data from restart file
1272
1273     TYPE(BLKTYPE), TARGET :: blocks(:)

```



```

1274 TYPE(BLKTYPE), POINTER :: b
1275 TYPE(NBRTYPE), POINTER :: NB
1276 INTEGER :: IBLK, I, J
1277
1278 DO IBLK = 1, NBLK
1279     b => blocks(IBLK)
1280     NB => b%NB
1281
1282     ! Set iteration bounds of each block to preserve BCs
1283     ! south and west boundaries:
1284         ! interior: iminloc, jminloc = 0 (use ghost)
1285         ! boundary: iminloc, jminloc = 2 (1st index is BC)
1286     ! north and east boundaries:
1287         ! interior: imaxloc, jmaxloc = maxblk (use ghost)
1288         ! boundary: imaxloc, jmaxloc = maxblk-1 (max index is BC)
1289
1290     ! NORTH
1291     IF (NB%N > 0) THEN
1292         ! Interior faces have positive ID neighbors
1293         b%JMAXLOC = JMAXBLK
1294     ELSE
1295         ! At North Boundary
1296         b%JMAXLOC = JMAXBLK - 1
1297     END IF
1298
1299     ! EAST
1300     IF (NB%E > 0) THEN
1301         ! Interior
1302         b%IMAXLOC = IMAXBLK
1303     ELSE
1304         ! At east Boundary
1305         b%IMAXLOC = IMAXBLK - 1
1306     END IF
1307
1308     ! SOUTH
1309     IF (NB%S > 0) THEN
1310         ! Interior
1311         b%JMINLOC = 0
1312     ELSE
1313         ! At south Boundary
1314         b%JMINLOC = 1
1315         ! boundary for updating temperature (dont update BC)
1316         b%JMINUPD = 2
1317     END IF
1318
1319     ! WEST
1320     IF (NB%W > 0) THEN
1321         ! Interior
1322         b%IMINLOC = 0
1323     ELSE
1324         ! At west Boundary
1325         b%IMINLOC = 1
1326         b%IMINUPD = 2
1327     END IF
1328 END DO
1329
1330 SUBROUTINE set_block_bounds
1331
1332 SUBROUTINE make_link(NB, list, nbrl, ID)
1333     ! make a single link in a linked list
1334     ! NB --> neighbor information (i.e. NB%N)
1335     ! list --> the neighbor linked list (i.e. nbrlists%N)
1336     ! nbrl --> pointer for neighbor linked list (i.e. nbrl%N)
1337     ! needs to be stored throughout the loop
1338     ! ID --> the block id to assign
1339
1340     ! Neighbor information pointer
1341     INTEGER :: NB, ID
1342     ! Linked lists of neighbor communication instructions
1343     TYPE(LNKLIST), POINTER :: list

```

```

1343     TYPE(LNKLIST), POINTER :: nbrl
1344
1345     IF ( .NOT. ASSOCIATED(list) ) THEN
1346         ! Allocate linked list if it hasnt been accessed yet
1347         ALLOCATE(list)
1348         ! Pointer linked list that will help iterate through the
1349         ! primary list in this loop
1350         nbrl => list
1351     ELSE
1352         ! linked list already allocated (started). Allocate next
1353         ! link as assign current block to it
1354         ALLOCATE(nbrl%next)
1355         nbrl => nbrl%next
1356     END IF
1357
1358     ! associate this linked list entry with the current block
1359     nbrl%ID = ID
1360     ! break link to pre-existing pointer target. We will
1361     ! allocated this target later as the next item in the linked list
1362     NULLIFY(nbrl%next)
1363
1364 END SUBROUTINE make_link
1365
1366 SUBROUTINE link_type(NB, list, nbrl, mpi, mpil, ID)
1367     ! make a single link in a linked list for a neighbor either on same
1368     ! processor or different processor
1369     ! NB --> neighbor information (i.e. NB%N)
1370     ! list --> the neighbor linked list (i.e. nbrlists%N)
1371     ! nbrl --> pointer for neighbor linked list (i.e. nbrl%N)
1372     ! needs to be stored throughout the loop
1373     ! mpi, mpil --> same as list/nbrl but for faces on other procs
1374     ! ID --> the block id to assign
1375
1376     ! Neighbor information pointer
1377     INTEGER :: NB, ID
1378     ! Linked lists of neighbor communication instructions
1379     TYPE(LNKLIST), POINTER :: list, nbrl, mpi, mpil
1380
1381     ! If block face is internal, add it to appropriate linked list
1382     ! for internal faces of a certian face (i.e. north).
1383     IF (NB > 0) THEN
1384         ! NEIGHBOR IS ON SAME PROCESSOR
1385         CALL make_link(NB, list, nbrl, ID)
1386     ELSE IF (NB < 0) THEN
1387         ! NEIGHBOR IS ON DIFFERENT PROCESSOR
1388         CALL make_link(NB, mpi, mpil, ID)
1389     END IF
1390 END SUBROUTINE link_type
1391
1392
1393 SUBROUTINE init_linklists(blocks, nbrlists, mpilists)
1394     ! Create linked lists governing block boundary communication.
1395     ! Separate list for each neighbor type so we can avoid logic when
1396     ! updating ghost nodes.
1397
1398     ! BLOCK DATA TYPE
1399     TYPE(BLKTYPE), TARGET :: blocks(:)
1400     ! Neighbor information pointer
1401     TYPE(NBRTYPE), POINTER :: NB
1402     ! Linked lists of neighbor communication instructions
1403     TYPE(NBRLIST) :: nbrlists
1404     TYPE(NBRLIST) :: nbrl
1405     TYPE(NBRLIST) :: mpilists
1406     TYPE(NBRLIST) :: mpil
1407     INTEGER :: IBLK
1408
1409     ! INITIALIZE LINKED LISTS (HPC1 REQUIRES THIS)
1410     NULLIFY(nbrlists%N)
1411     NULLIFY(nbrlists%S)

```

```

1412 NULLIFY(nbrlists%E)
1413 NULLIFY(nbrlists%W)
1414 NULLIFY(nbrlists%NW)
1415 NULLIFY(nbrlists%NE)
1416 NULLIFY(nbrlists%SE)
1417 NULLIFY(nbrlists%SW)
1418
1419 NULLIFY(mpilists%N)
1420 NULLIFY(mpilists%S)
1421 NULLIFY(mpilists%E)
1422 NULLIFY(mpilists%W)
1423 NULLIFY(mpilists%NW)
1424 NULLIFY(mpilists%NE)
1425 NULLIFY(mpilists%SE)
1426 NULLIFY(mpilists%SW)
1427
1428 DO IBLK = 1, MYNBLK
1429     NB => blocks(IBLK)%NB
1430
1431     ! NORTH
1432     IF (NB%N > 0) THEN
1433         ! NEIGHBOR IS ON SAME PROCESSOR
1434         CALL make_link(NB%N, nbrlists%N, nbrl%N, ID%N)
1435     ELSE IF (NB%N < 0) THEN
1436         ! NEIGHBOR IS ON DIFFERENT PROCESSOR
1437         CALL make_link(NB%N, mpi%N, mpil%N, ID%N)
1438     END IF
1439     CALL link_type(NB%N, nbrlists%N, nbrl%N, mpilists%N, mpil%N, IBLK)
1440     ! SOUTH
1441     CALL link_type(NB%S, nbrlists%S, nbrl%S, mpilists%S, mpil%S, IBLK)
1442     ! EAST
1443     CALL link_type(NB%E, nbrlists%E, nbrl%E, mpilists%E, mpil%E, IBLK)
1444     ! WEST
1445     CALL link_type(NB%W, nbrlists%W, nbrl%W, mpilists%W, mpil%W, IBLK)
1446     ! NORTH EAST
1447     CALL link_type(NB%NE, nbrlists%NE, nbrl%NE, mpilists%NE, mpil%NE, IBLK)
1448     ! SOUTH EAST
1449     CALL link_type(NB%SE, nbrlists%SE, nbrl%SE, mpilists%SE, mpil%SE, IBLK)
1450     ! SOUTH WEST
1451     CALL link_type(NB%SW, nbrlists%SW, nbrl%SW, mpilists%SW, mpil%SW, IBLK)
1452     ! NORTH WEST
1453     CALL link_type(NB%NW, nbrlists%NW, nbrl%NW, mpilists%NW, mpil%NW, IBLK)
1454 END DO
1455
1456 !
1457 !
1458 !
1459 !
1460 !
1461 !
1462 !
1463 !
1464 !
1465 !
1466 !
1467
1468
1469 END SUBROUTINE init_linklists
1470
1471 SUBROUTINE update_ghosts_sameproc(b, nbrlists)
1472     ! Update ghost nodes of each block based on neighbor linked list for
1473     ! neighbors on same processor as current block.
1474     ! Ghost nodes contain solution from respective block face/corner
1475     ! neighbor for use in current block solution.
1476
1477     ! BLOCK DATA TYPE
1478     TYPE(BLKTYPE), TARGET :: b(:)
1479     ! temperature information pointers for ghost and neighbor nodes
1480     REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb

```

```

1481 ! Linked lists of neighbor communication instructions
1482 TYPE(NBRLIST) :: nbrlists
1483 TYPE(NBRLIST) :: nbrl
1484 ! iteration parameters, index of neighbor
1485 INTEGER :: I, J, INBR
1486
1487 !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1488
1489 ! NORTH FACE GHOST NODES
1490 nbrl%N => nbrlists%N
1491 ! Step through linked list of north faces with ghosts until end of list
1492 DO
1493     ! If next link in list doesnt exist (end of list), stop loop
1494     IF ( .NOT. ASSOCIATED(nbrl%N) ) EXIT
1495
1496     ! Otherwise, assign neighbor values to all ghost nodes:
1497
1498     ! TEMPERATURE OF CURRENT BLOCK (CONTAINS GHOST NODES)
1499     ! (identified by linked list id)
1500     Tgh => b( nbrl%N%ID )%mesh%T
1501
1502     ! index of north neighbor
1503     INBR = b( nbrl%N%ID )%NBLOC%N
1504     ! TEMPERATURE OF NEIGHBOR BLOCK (UPDATE GHOSTS WITH THIS)
1505     Tnb => b( INBR )%mesh%T
1506
1507     DO I = 1, IMAXBLK
1508         ! NORTH FACE GHOST NODE TEMPERATURE IS EQUAL TO TEMPERATURE OF
1509         ! SECOND-FROM-SOUTH FACE OF NORTH NEIGHBOR
1510         ! (Remember face nodes are shared between blocks)
1511         Tgh(I, JMAXBLK+1) = Tnb(I, 2)
1512     END DO
1513     ! switch pointer to next link in list
1514     nbrl%N => nbrl%N%next
1515 END DO
1516
1517 ! SOUTH FACE GHOST NODES
1518 nbrl%S => nbrlists%S
1519 DO
1520     IF ( .NOT. ASSOCIATED(nbrl%S) ) EXIT
1521     Tgh => b( nbrl%S%ID )%mesh%T
1522     INBR = b( nbrl%S%ID )%NBLOC%S
1523     Tnb => b( INBR )%mesh%T
1524
1525     DO I = 1, IMAXBLK
1526         ! ADD NORTH FACE OF SOUTH NEIGHBOR TO CURRENT SOUTH FACE GHOSTS
1527         Tgh(I, 0) = Tnb(I, JMAXBLK-1)
1528     END DO
1529     nbrl%S => nbrl%S%next
1530 END DO
1531
1532 ! EAST FACE GHOST NODES
1533 nbrl%E => nbrlists%E
1534 DO
1535     IF ( .NOT. ASSOCIATED(nbrl%E) ) EXIT
1536     Tgh => b( nbrl%E%ID )%mesh%T
1537     INBR = b( nbrl%E%ID )%NBLOC%E
1538     Tnb => b( INBR )%mesh%T
1539
1540     DO J = 1, JMAXBLK
1541         ! ADD WEST FACE OF EAST NEIGHBOR TO CURRENT WEST FACE GHOSTS
1542         Tgh(IMAXBLK+1, J) = Tnb(2, J)
1543     END DO
1544     nbrl%E => nbrl%E%next
1545 END DO
1546
1547 ! WEST FACE GHOST NODES
1548 nbrl%W => nbrlists%W
1549 DO

```

```

1550     IF ( .NOT. ASSOCIATED(nbrl%W) ) EXIT
1551     Tgh => b( nbrl%W%ID )%mesh%T
1552     INBR = b( nbrl%W%ID )%NBLOC%W
1553     Tnb => b( INBR )%mesh%T
1554
1555     DO J = 1, JMAXBLK
1556         ! ADD EAST FACE OF WEST NEIGHBOR TO CURRENT EAST FACE GHOSTS
1557         Tgh(0, J) = Tnb(IMAXBLK-1, J)
1558     END DO
1559     nbrl%W => nbrl%W%next
1560 END DO
1561
1562 !!! CORNERS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1563
1564 ! NORTH EAST CORNER GHOST NODES
1565 nbrl%NE => nbrlists%NE
1566 DO
1567     IF ( .NOT. ASSOCIATED(nbrl%NE) ) EXIT
1568     Tgh => b( nbrl%NE%ID )%mesh%T
1569     INBR = b( nbrl%NE%ID )%NBLOC%NE
1570     Tnb => b( INBR )%mesh%T
1571     ! ADD SW CORNER OF NE NEIGHBOR TO CURRENT NE CORNER GHOSTS
1572     Tgh(IMAXBLK+1, JMAXBLK+1) = Tnb(2, 2)
1573     nbrl%NE => nbrl%NE%next
1574 END DO
1575
1576 ! SOUTH EAST CORNER GHOST NODES
1577 nbrl%SE => nbrlists%SE
1578 DO
1579     IF ( .NOT. ASSOCIATED(nbrl%SE) ) EXIT
1580     Tgh => b( nbrl%SE%ID )%mesh%T
1581     INBR = b( nbrl%SE%ID )%NBLOC%SE
1582     Tnb => b( INBR )%mesh%T
1583     ! ADD NW CORNER OF SE NEIGHBOR TO CURRENT SE CORNER GHOSTS
1584     Tgh(IMAXBLK+1, 0) = Tnb(2, JMAXBLK-1)
1585     nbrl%SE => nbrl%SE%next
1586 END DO
1587
1588 ! SOUTH WEST CORNER GHOST NODES
1589 nbrl%SW => nbrlists%SW
1590 DO
1591     IF ( .NOT. ASSOCIATED(nbrl%SW) ) EXIT
1592     Tgh => b( nbrl%SW%ID )%mesh%T
1593     INBR = b( nbrl%SW%ID )%NBLOC%SW
1594     Tnb => b( INBR )%mesh%T
1595     ! ADD NE CORNER OF SW NEIGHBOR TO CURRENT SW CORNER GHOSTS
1596     Tgh(0, 0) = Tnb(IMAXBLK-1, JMAXBLK-1)
1597     nbrl%SW => nbrl%SW%next
1598 END DO
1599
1600 ! NORTH WEST CORNER GHOST NODES
1601 nbrl%NW => nbrlists%NW
1602 DO
1603     IF ( .NOT. ASSOCIATED(nbrl%NW) ) EXIT
1604     Tgh => b( nbrl%NW%ID )%mesh%T
1605     INBR = b( nbrl%NW%ID )%NBLOC%NW
1606     Tnb => b( INBR )%mesh%T
1607     ! ADD SE CORNER OF NW NEIGHBOR TO CURRENT NW CORNER GHOSTS
1608     Tgh(0, JMAXBLK+1) = Tnb(IMAXBLK-1, 2)
1609     nbrl%NW => nbrl%NW%next
1610 END DO
1611 END SUBROUTINE update_ghosts_sameproc
1612
1613 SUBROUTINE update_ghosts_diffproc_send(blocks, mpilists)
1614     ! Gather information on different processors to update current ghosts.
1615     ! send ghost info to neighbor processor to buffer its ghost nodes
1616     ! store in buffers and send via MPI. Buffers will be MPI recieved
1617     ! and distributed to corresponding blocks
1618

```

```

1619 ! BLOCK DATA TYPE
1620 TYPE(BLKTYPE), TARGET :: blocks(:)
1621 TYPE(BLKTYPE), POINTER :: b
1622 ! temperature information pointers for ghost and neighbor nodes
1623 REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
1624 ! buffers to store temperature information for I/J faces, and corners
1625 REAL(KIND=8) :: bufferI(IMAXBLK), bufferJ(JMAXBLK), buffer
1626 ! Linked lists of neighbor communication instructions
1627 TYPE(NBRLIST) :: mpilists
1628 TYPE(NBRLIST) :: mpil
1629 ! iteration parameters, index of neighbor, communication tag, destination
1630 INTEGER :: I, J, INBR, tag, dest
1631 ! counts number of sends in certian direction by proc
1632 INTEGER :: sends
1633
1634 !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1635 ! NORTH FACE GHOST INFO
1636 ! Send north face data to north neighbor to put in neighbor's ghost nodes
1637 mpil%N => mpilists%N
1638 DO
1639     IF ( .NOT. ASSOCIATED(mpil%N) ) EXIT
1640
1641     ! CURRENT BLOCK
1642     b => blocks( mpil%N%ID )
1643     ! SEND FACE INFORMATION FOR THIS BLOCK TO NEIGHBORS ON OTHER PROCS
1644     DO I = 1, IMAXBLK
1645         ! FILL BUFFER WITH TEMPERATURE FROM THIS BLOCK FOR GHOSTS OF OTHER BLOCK
1646         ! (indexing is opposite from project 3 as we are now sending
1647         ! ghost info to neighbor rather than recieving it from them here)
1648         bufferI(I) = b%mesh%T(I, JMAXBLK-1)
1649     END DO
1650
1651     ! FIND DESITNATION OF GHOST INFO (proc id of neighbor)
1652     dest = b%NP%N
1653     ! MAKE UNIQUE TAG
1654     CALL make_mpi_tag(NBND, b%ID, tag)
1655     ! SEND INFO TO NEIGHBOR PROC AND CONTINUE OPERATIONS WITHOUT CONFIRMATION
1656     CALL MPI_Isend(bufferI, IMAXBLK, MPI_REAL8, dest, tag, &
1657                   MPI_COMM_WORLD, REQUEST, IERROR)
1658
1659     mpil%N => mpil%N%next
1660 END DO
1661
1662 ! SOUTH FACE GHOST NODES
1663 mpil%S => mpilists%S
1664 DO
1665     IF ( .NOT. ASSOCIATED(mpil%S) ) EXIT
1666
1667     b => blocks( mpil%S%ID )
1668     DO I = 1, IMAXBLK
1669         bufferI(I) = b%mesh%T(I, 2)
1670     END DO
1671
1672     dest = b%NP%S
1673     CALL make_mpi_tag(SBND, b%ID, tag)
1674     CALL MPI_Isend(bufferI, IMAXBLK, MPI_REAL8, dest, tag, &
1675                   MPI_COMM_WORLD, REQUEST, IERROR)
1676
1677     mpil%S => mpil%S%next
1678 END DO
1679
1680 ! EAST FACE GHOST NODES
1681 mpil%E => mpilists%E
1682 DO
1683     IF ( .NOT. ASSOCIATED(mpil%E) ) EXIT
1684
1685     b => blocks( mpil%E%ID )
1686     DO J = 1, JMAXBLK
1687         bufferJ(J) = b%mesh%T(IMAXBLK-1, J)

```

```

1688         END DO
1689
1690         dest = b%NP%E
1691         CALL make_mpi_tag(EBND, b%ID, tag)
1692         CALL MPI_Isend(bufferJ, JMAXBLK, MPI_REAL8, dest, tag, &
1693             MPI_COMM_WORLD, REQUEST, IERROR)
1694
1695         mpil%E => mpil%E%next
1696     END DO
1697
1698     ! WEST FACE GHOST NODES
1699     mpil%W => mpilists%W
1700     DO
1701         IF ( .NOT. ASSOCIATED(mpil%W) ) EXIT
1702
1703         b => blocks( mpil%W%ID )
1704         DO J = 1, JMAXBLK
1705             bufferJ(J) = b%mesh%T(2, J)
1706         END DO
1707
1708         dest = b%NP%W
1709         CALL make_mpi_tag(WBND, b%ID, tag)
1710         CALL MPI_Isend(bufferJ, JMAXBLK, MPI_REAL8, dest, tag, &
1711             MPI_COMM_WORLD, REQUEST, IERROR)
1712
1713         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1714         ! if (myid == 3 .and. b%ID == 4) then
1715         !     write(*,*)
1716         !     write(*,*) "send east ghosts from: ", b%ID
1717         !     write(*,*) "buffer values: ", bufferJ(2)
1718         ! end if
1719
1720         mpil%W => mpil%W%next
1721     END DO
1722
1723     ! NORTH EAST CORNER GHOST NODES
1724     mpil%NE => mpilists%NE
1725     DO
1726         IF ( .NOT. ASSOCIATED(mpil%NE) ) EXIT
1727
1728         b => blocks( mpil%NE%ID )
1729         buffer = b%mesh%T(IMAXBLK-1, JMAXBLK-1)
1730
1731         dest = b%NP%NE
1732         CALL make_mpi_tag(NEBND, b%ID, tag)
1733         CALL MPI_Isend(buffer, 1, MPI_REAL8, dest, tag, &
1734             MPI_COMM_WORLD, REQUEST, IERROR)
1735
1736         mpil%NE => mpil%NE%next
1737     END DO
1738
1739     ! SOUTH EAST CORNER GHOST NODES
1740     mpil%SE => mpilists%SE
1741     DO
1742         IF ( .NOT. ASSOCIATED(mpil%SE) ) EXIT
1743
1744         b => blocks( mpil%SE%ID )
1745         buffer = b%mesh%T(IMAXBLK-1, 2)
1746
1747         dest = b%NP%SE
1748         CALL make_mpi_tag(SEBND, b%ID, tag)
1749         CALL MPI_Isend(buffer, 1, MPI_REAL8, dest, tag, &
1750             MPI_COMM_WORLD, REQUEST, IERROR)
1751
1752         mpil%SE => mpil%SE%next
1753     END DO
1754
1755     ! SOUTH WEST CORNER GHOST NODES
1756     mpil%SW => mpilists%SW

```

```

1757 DO
1758     IF ( .NOT. ASSOCIATED(mpil%SW) ) EXIT
1759
1760     b => blocks( mpil%SW%ID )
1761     buffer = b%mesh%T(2, 2)
1762
1763     dest = b%NP%SW
1764     CALL make_mpi_tag(SWBND, b%ID, tag)
1765     CALL MPI_Isend(buffer, 1, MPI_REAL8, dest, tag, &
1766                   MPI_COMM_WORLD, REQUEST, IERROR)
1767     mpil%SW => mpil%SW%next
1768 END DO
1769
1770 ! NORTH WEST CORNER GHOST NODES
1771 mpil%NW => mpilists%NW
1772 DO
1773     IF ( .NOT. ASSOCIATED(mpil%NW) ) EXIT
1774
1775     b => blocks( mpil%NW%ID )
1776     buffer = b%mesh%T(2, JMAXBLK-1)
1777
1778     dest = b%NP%NW
1779     CALL make_mpi_tag(NWBND, b%ID, tag)
1780     CALL MPI_Isend(buffer, 1, MPI_REAL8, dest, tag, &
1781                   MPI_COMM_WORLD, REQUEST, IERROR)
1782     mpil%NW => mpil%NW%next
1783 END DO
1784
1785 END SUBROUTINE update_ghosts_diffproc_send
1786
1787 SUBROUTINE update_ghosts_diffproc_recv(blocks, mpilists)
1788     ! Recieve information on different processors to update current ghosts.
1789     ! store in buffers and send via MPI. Buffers will be MPI recieved
1790     ! and distributed to corresponding blocks
1791
1792     ! BLOCK DATA TYPE
1793     TYPE(BLKTYPE), TARGET :: blocks(:)
1794     TYPE(BLKTYPE), POINTER :: b
1795     ! temperature information pointers for ghost and neighbor nodes
1796     REAL(KIND=8), POINTER, DIMENSION(:, :) :: Tgh, Tnb
1797     ! buffers to store temperature information for I/J faces, and corners
1798     REAL(KIND=8) :: bufferI(IMAXBLK), bufferJ(JMAXBLK), buffer
1799     ! Linked lists of neighbor communication instructions
1800     TYPE(NBRLIST) :: mpilists
1801     TYPE(NBRLIST) :: mpil
1802     ! iteration parameters, index of neighbor, communication tag, source proc id
1803     INTEGER :: I, J, INBR, tag, src
1804     ! counts number of sends in certian direction by proc
1805     INTEGER :: sends
1806
1807     !!! FACES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1808     ! NORTH FACE GHOST NODES
1809     ! get info sent by north neighbor and add to ghost node
1810     mpil%N => mpilists%N
1811 DO
1812     IF ( .NOT. ASSOCIATED(mpil%N) ) EXIT
1813
1814     ! CURRENT BLOCK
1815     b => blocks( mpil%N%ID )
1816     ! SOURCE PROCESSOR ID
1817     src = b%NP%N
1818     ! TAG OF NEIGHBOR IS OPPOSITE OF THIS FACE
1819     CALL make_mpi_tag(SBND, ABS(b%NB%N), tag)
1820     ! GET INFO FROM SOURCE PROCESSOR
1821     CALL MPI_RECV(bufferI, IMAXBLK, MPI_REAL8, src, tag, &
1822                  MPI_COMM_WORLD, STATUS, IERROR)
1823
1824     ! ASSIGN GHOST INFORMATION
1825     DO I = 1, IMAXBLK

```



```

1826         b%mesh%T(I, JMAXBLK+1) = bufferI(I)
1827     END DO
1828
1829     mpil%N => mpil%N%next
1830 END DO
1831
1832 ! SOUTH FACE GHOST NODES
1833 mpil%S => mpilists%S
1834 DO
1835     IF ( .NOT. ASSOCIATED(mpil%S) ) EXIT
1836
1837     b => blocks( mpil%S%ID )
1838     src = b%NP%S
1839     CALL make_mpi_tag(NBND, ABS(b%NB%S), tag)
1840     CALL MPI_RECV(bufferI, IMAXBLK, MPI_REAL8, src, tag, &
1841         MPI_COMM_WORLD, STATUS, IERROR)
1842     DO I = 1, IMAXBLK
1843         b%mesh%T(I, 0) = bufferI(I)
1844     END DO
1845
1846     mpil%S => mpil%S%next
1847 END DO
1848
1849 ! EAST FACE GHOST NODES
1850 mpil%E => mpilists%E
1851 DO
1852     IF ( .NOT. ASSOCIATED(mpil%E) ) EXIT
1853
1854     b => blocks( mpil%E%ID )
1855     src = b%NP%E
1856     CALL make_mpi_tag(WBND, ABS(b%NB%E), tag)
1857     CALL MPI_RECV(bufferJ, JMAXBLK, MPI_REAL8, src, tag, &
1858         MPI_COMM_WORLD, STATUS, IERROR)
1859
1860 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
1861 ! if (myid == 0 .and. b%ID == 3) then
1862 !     write(*,*)
1863 !     write(*,*) "recieve east ghosts for: ", b%ID
1864 !     write(*,*) "buffer values: ", bufferJ(2)
1865 ! end if
1866
1867     DO J = 1, JMAXBLK
1868         b%mesh%T(IMAXBLK+1, J) = bufferJ(J)
1869     END DO
1870
1871     mpil%E => mpil%E%next
1872 END DO
1873
1874 ! WEST FACE GHOST NODES
1875 mpil%W => mpilists%W
1876 DO
1877     IF ( .NOT. ASSOCIATED(mpil%W) ) EXIT
1878
1879     b => blocks( mpil%W%ID )
1880     src = b%NP%W
1881     CALL make_mpi_tag(EBND, ABS(b%NB%W), tag)
1882     CALL MPI_RECV(bufferJ, JMAXBLK, MPI_REAL8, src, tag, &
1883         MPI_COMM_WORLD, STATUS, IERROR)
1884     DO J = 1, JMAXBLK
1885         b%mesh%T(0, J) = bufferJ(J)
1886     END DO
1887
1888     mpil%W => mpil%W%next
1889 END DO
1890
1891 ! NORTH EAST CORNER GHOST NODES
1892 mpil%NE => mpilists%NE
1893 DO
1894     IF ( .NOT. ASSOCIATED(mpil%NE) ) EXIT

```

```

1895
1896      b => blocks( mpil%NE%ID )
1897      src = b%NP%NE
1898      CALL make_mpi_tag(SWBND, ABS(b%NB%NE), tag)
1899      CALL MPI_RECV(buffer, 1, MPI_REAL8, src, tag, &
1900                   MPI_COMM_WORLD, STATUS, IERROR)
1901      b%mesh%T(IMAXBK+1, JMAXBK+1) = buffer
1902
1903      mpil%NE => mpil%NE%next
1904  END DO
1905
1906  ! SOUTH EAST CORNER GHOST NODES
1907  mpil%SE => mpilists%SE
1908  DO
1909      IF ( .NOT. ASSOCIATED(mpil%SE) ) EXIT
1910
1911      b => blocks( mpil%SE%ID )
1912      src = b%NP%SE
1913      CALL make_mpi_tag(NWBND, ABS(b%NB%SE), tag)
1914      CALL MPI_RECV(buffer, 1, MPI_REAL8, src, tag, &
1915                   MPI_COMM_WORLD, STATUS, IERROR)
1916      b%mesh%T(IMAXBK+1, 0) = buffer
1917
1918      mpil%SE => mpil%SE%next
1919  END DO
1920
1921  ! SOUTH WEST CORNER GHOST NODES
1922  mpil%SW => mpilists%SW
1923  DO
1924      IF ( .NOT. ASSOCIATED(mpil%SW) ) EXIT
1925
1926      b => blocks( mpil%SW%ID )
1927      src = b%NP%SW
1928      CALL make_mpi_tag(NEBND, ABS(b%NB%SW), tag)
1929      CALL MPI_RECV(buffer, 1, MPI_REAL8, src, tag, &
1930                   MPI_COMM_WORLD, STATUS, IERROR)
1931      b%mesh%T(0, 0) = buffer
1932
1933      mpil%SW => mpil%SW%next
1934  END DO
1935
1936  ! NORTH WEST CORNER GHOST NODES
1937  mpil%NW => mpilists%NW
1938  DO
1939      IF ( .NOT. ASSOCIATED(mpil%NW) ) EXIT
1940
1941      b => blocks( mpil%NW%ID )
1942      src = b%NP%NW
1943      CALL make_mpi_tag(SEBND, ABS(b%NB%NW), tag)
1944
1945      CALL MPI_RECV(buffer, 1, MPI_REAL8, src, tag, &
1946                   MPI_COMM_WORLD, STATUS, IERROR)
1947      b%mesh%T(0, JMAXBK+1) = buffer
1948
1949      mpil%NW => mpil%NW%next
1950  END DO
1951
1952  END SUBROUTINE update_ghosts_diffproc_recv
1953
1954  SUBROUTINE make_mpi_tag(dir, srcblk, tag)
1955      ! Make unique tag for mpi send/receive. Sends are always from one
1956      ! proc to another, so we just need unique tags for each block in each
1957      ! direction.
1958      ! Accomplish this by making a unique number for each direction and
1959      ! concatenating the global block number of the sending block to it
1960
1961      ! dir --> direction of send
1962      ! srcblk --> global id of block sending information
1963

```

```

1964     INTEGER :: dir, srcblk, tag
1965     CHARACTER(len=1) :: dirstr
1966     CHARACTER(len=25) :: srcstr
1967     CHARACTER(len=50) :: tagstr
1968
1969     ! CONVERT INTEGERS TO STRINGS
1970     WRITE(dirstr, '(I1)') dir
1971     WRITE(srcstr, *) srcblk
1972
1973     ! CONCATENATE STRINGS INTO INTEGER VALUE
1974     ! (direction, then block number)
1975     ! adjust right and left so numbers line up
1976     tagstr = ADJUSTR(TRIM(dirstr)) // ADJUSTL(TRIM(srcstr))
1977     ! CONVERT TO INTEGER
1978     READ(tagstr, *) tag
1979
1980 END SUBROUTINE make_mpi_tag
1981
1982
1983 SUBROUTINE calc_cell_params(blocks)
1984     ! calculate areas for secondary fluxes and constant terms in heat
1985     ! transfer eqn. Call after reading mesh data from restart file
1986
1987     ! BLOCK DATA TYPE
1988     TYPE(BLKTYPE), TARGET :: blocks(:)
1989     TYPE(MESHTYPE), POINTER :: m
1990     INTEGER :: IBLK, I, J
1991     ! Areas used in counter-clockwise trapezoidal integration to get
1992     ! x and y first-derivatives for center of each cell (Green's thm)
1993     REAL(KIND=8) :: Ayi_half, Axi_half, Ayj_half, Axj_half
1994
1995     DO IBLK = 1, MYNBLK
1996         m => blocks(IBLK)%mesh
1997
1998         DO J = 0, JMAXBLK+1
1999             DO I = 0, IMAXBLK+1
2000                 ! CALC CELL VOLUME
2001                 ! cross product of cell diagonals p, q
2002                 ! where p has x,y components px, py and q likewise.
2003                 ! Thus, p cross q = px*qy - qx*py
2004                 ! where, px = x(i+1,j+1) - x(i,j), py = y(i+1,j+1) - y(i,j)
2005                 ! and    qx = x(i,j+1) - x(i+1,j), qy = y(i,j+1) - y(i+1,j)
2006                 m%V(I,J) = ABS( ( m%x(I+1,J+1) - m%x(I, J) ) &
2007                               * ( m%y(I, J+1) - m%y(I+1,J) ) &
2008                               - ( m%x(I, J+1) - m%x(I+1,J) ) &
2009                               * ( m%y(I+1,J+1) - m%y(I, J) ) )
2010             END DO
2011         END DO
2012
2013         ! CALC CELL AREAS (FLUXES) IN J-DIRECTION
2014         DO J = 0, JMAXBLK+1
2015             DO I = 0, IMAXBLK
2016                 m%Axi(I,J) = m%x(I+1,J) - m%x(I,J)
2017                 m%Ayj(I,J) = m%y(I+1,J) - m%y(I,J)
2018             END DO
2019         END DO
2020
2021         ! CALC CELL AREAS (FLUXES) IN I-DIRECTION
2022         DO J = 0, JMAXBLK
2023             DO I = 0, IMAXBLK+1
2024                 ! CALC CELL AREAS (FLUXES)
2025                 m%Axi(I,J) = m%x(I,J+1) - m%x(I,J)
2026                 m%Ayi(I,J) = m%y(I,J+1) - m%y(I,J)
2027             END DO
2028         END DO
2029
2030         ! Actual finite-volume scheme equation parameters
2031         DO J = 0, JMAXBLK
2032             DO I = 0, IMAXBLK

```

[illegible]

```

2102 SUBROUTINE calc_temp(b)
2103     ! Calculate temperature at all points in mesh, excluding BC cells.
2104     ! Calculate first and second derivatives for finite-volume scheme
2105
2106     TYPE(BLKTYPE), TARGET :: b(:)
2107     TYPE(MESHTYPE), POINTER :: m
2108     ! First partial derivatives of temperature in x and y directions
2109     REAL(KIND=8) :: dTdx, dTdy
2110     INTEGER :: IBLK, I, J
2111
2112     DO IBLK = 1, MYNBLK
2113         m => b(IBLK)%mesh
2114
2115         ! RESET SUMMATION
2116         m%Ttmp = 0.D0
2117
2118         ! PREVIOUSLY SET ITERATION LIMITS TO UTILIZE GHOST NODES ONLY
2119         !ON INTERIOR FACES
2120         DO J = b(IBLK)%JMINLOC, b(IBLK)%JMAXLOC
2121             DO I = b(IBLK)%IMINLOC, b(IBLK)%IMAXLOC
2122                 ! CALC FIRST DERIVATIVES
2123                 dTdx = + 0.5d0 &
2124                     * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Ayi(I+1,J) &
2125                       - ( m%T(I, J) + m%T(I, J+1) ) * m%Ayi(I, J) &
2126                       - ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Ayj(I,J+1) &
2127                       + ( m%T(I, J) + m%T(I+1, J) ) * m%Ayj(I, J) &
2128                       ) / m%V(I,J)
2129                 dTdy = - 0.5d0 &
2130                     * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Axi(I+1,J) &
2131                       - ( m%T(I, J) + m%T(I, J+1) ) * m%Axi(I, J) &
2132                       - ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Axi(I,J+1) &
2133                       + ( m%T(I, J) + m%T(I+1, J) ) * m%Axi(I, J) &
2134                       ) / m%V(I,J)
2135
2136                 ! Alternate distributive scheme second-derivative operator.
2137                 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 )
2138                 m%Ttmp(I, J) = m%Ttmp(I, J) + m%term(I, J) * ( m%yPN(I,J) * dTdx + m%xNP(I,J) * dTdy )
2139                 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 )
2140                 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 )
2141             END DO
2142         END DO
2143         ! SAVE NEW TEMPERATURE DISTRIBUTION
2144         ! (preserve Ttmp for residual calculation in solver loop)
2145
2146         ! Previously set bounds, add one to lower limit so as not to
2147         ! update BC. (dont need to for upper limit because explicit scheme)
2148         DO J = b(IBLK)%JMINLOC + 1, b(IBLK)%JMAXLOC
2149             DO I = b(IBLK)%IMINLOC + 1, b(IBLK)%IMAXLOC
2150                 m%T(I,J) = m%T(I,J) + m%Ttmp(I,J)
2151             END DO
2152         END DO
2153     END DO
2154 END SUBROUTINE calc_temp
2155
2156 END MODULE BLOCKMOD

```

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

Appendix D: Multi-Block Plot3D Reader-Writer

```
1  ! MAE 267
2  ! PROJECT 5
3  ! LOGAN HALSTROM
4  ! 29 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
38     SUBROUTINE write_config(procs)
39         ! Write block connectivity file with neighbor and BC info
40         ! for each processor.
41         ! Also write PLOT3D restart files for each processor.
42
43         TYPE(PROCTYPE), TARGET :: procs(:)
44         TYPE(PROCTYPE), POINTER :: p
45         ! BLOCK DATA TYPE
46         TYPE(BLKTYPE), POINTER :: b
47         INTEGER :: IP, IB, BLKFILE = 99
48         CHARACTER(2) :: procname
49         CHARACTER(20) :: xfile, qfile
50
51         33 FORMAT(A)
52         11 FORMAT( 3I7)
53         22 FORMAT(33I7)
54         44 FORMAT(33A7)
55
56         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
57         !!! WRITE CONFIG FILE FOR EACH PROCESSOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
58         !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
59
60         DO IP = 1, NPROCS
61             p => procs(IP)
62
63             ! MAKE FILE NAME (i.e. 'p01.config')
64             IF (p%ID<10) THEN
65                 ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
66                 WRITE(procname, '(A,I1)') '0', p%ID
67             ELSE
```

```

68      WRITE(procname, '(I2)') p%ID
69  END IF

70
71  OPEN (UNIT = BLKFILE , FILE = TRIM("p" // procname // ".config"), form='formatted')
72
73  ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
74  WRITE(BLKFILE, 33) 'NBLK' // ' IMAXBLK' // ' JMAXBLK'
75  WRITE(BLKFILE, 11) p%NBLK, IMAXBLK, JMAXBLK
76
77  ! HEADER
78  WRITE(BLKFILE, 44) 'ID', 'IMIN', 'JMIN', 'SIZE', &
79                    'IMNL', 'IMXL', 'JMN', 'JMXL', &
80                    'NNB', 'NNP', 'NLOC', &
81                    'SNB', 'SNP', 'SLOC', &
82                    'ENB', 'ENP', 'ELOC', &
83                    'WNB', 'WNP', 'WLOC', &
84                    'NENB', 'NENP', 'NEL', &
85                    'SENB', 'SENP', 'SEL', &
86                    'SWNB', 'SWNP', 'SWL', &
87                    'NWNB', 'NWNP', 'NWL', &
88                    'ORI'
89
90  DO IB = 1, p%NBLK
91      b => p%blocks(IB)
92      ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
93      ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
94      ! NORTH EAST SOUTH WEST
95      WRITE(BLKFILE, 22) b%ID, b%IMIN, b%JMIN, INT(b%SIZE), &
96                        b%IMINLOC, b%IMAXLOC, b%JMINLOC, b%JMAXLOC, &
97                        b%NB%N, b%NP%N, b%NBLOC%N, &
98                        b%NB%S, b%NP%S, b%NBLOC%S, &
99                        b%NB%E, b%NP%E, b%NBLOC%E, &
100                       b%NB%W, b%NP%W, b%NBLOC%W, &
101                       b%NB%NE, b%NP%NE, b%NBLOC%NE, &
102                       b%NB%SE, b%NP%SE, b%NBLOC%SE, &
103                       b%NB%SW, b%NP%SW, b%NBLOC%SW, &
104                       b%NB%NW, b%NP%NW, b%NBLOC%NW, &
105                       b%ORIENT
106
107  END DO
108
109  CLOSE(BLKFILE)
110
111  END DO
112
113  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
114  !!! WRITE SOLUTION RESTART FILES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
115  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
116
117  DO IP = 1, NPROCS
118      p => procs(IP)
119      ! MAKE FILE NAME
120      IF (p%ID<10) THEN
121          ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
122          WRITE(procname, '(A,I1)') '0', p%ID
123      ELSE
124          WRITE(procname, '(I2)') p%ID
125      END IF
126      xfile = "p" // procname // ".grid"
127      qfile = "p" // procname // ".T"
128      CALL plot3D(p%blocks, p%NBLK, xfile, qfile)
129  END DO
130
131  SUBROUTINE write_config
132
133  SUBROUTINE read_config(blocks)
134      ! Called by each processor individually for its own blocks
135      ! For given processor, read corresponding configuration file.
136      ! Get neighbor connectivity info
137      ! Also read PLOT3D restart files for grids for given processor
138
139      ! BLOCK DATA TYPE
140      TYPE(BLKTYPE), POINTER :: blocks(:)
141      TYPE(BLKTYPE), POINTER :: b

```

```

137 TYPE(MESHTYPE), POINTER :: m
138 INTEGER :: IP, IB, BLKFILE = 99
139 CHARACTER(2) :: procname
140 CHARACTER(20) :: xfile, qfile
141
142 33 FORMAT(A)
143 11 FORMAT( 3I7)
144 22 FORMAT(33I7)
145 44 FORMAT(33A7)
146
147 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
148 !!! READ CONFIG FILE FOR GIVEN PRO!SSOR !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
149 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
150
151
152 ! FILE NAME (i.e. 'p01.config')
153 IF (MYID<10) THEN
154     ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
155     WRITE(procname, '(A,I1)') '0', MYID
156 ELSE
157     WRITE(procname, '(I2)') MYID
158 END IF
159
160 OPEN (UNIT = BLKFILE , FILE = TRIM("p" // procname // ".config"), form='formatted')
161
162 ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
163 READ(BLKFILE, *)
164 READ(BLKFILE, 11) MYNBLK, IMAXBK, JMAXBK
165 ! (MYNBLK is global variable on this processor that is the number of
166 !   blocks allocated to this processor)
167
168 ! ALLOCATE BLOCKS FOR THIS PROCESSOR
169 ! ('blocks' stores just the blocks for this processor. It is
170 ! in parallel for each processor)
171 ALLOCATE( blocks(1:MYNBLK) )
172
173 ! ALLOCATE MESH STUFF TO BE READ IN
174 DO IB = 1, MYNBLK
175     m => blocks(IB)%mesh
176
177     ! ALLOCATE MESH INFORMATION
178     ! ADD EXTRA INDEX AT BEGINNING AND END FOR GHOST NODES
179     ALLOCATE( m%xp( 0:IMAXBK+1, 0:JMAXBK+1) )
180     ALLOCATE( m%yp( 0:IMAXBK+1, 0:JMAXBK+1) )
181     ALLOCATE( m%x( 0:IMAXBK+1, 0:JMAXBK+1) )
182     ALLOCATE( m%y( 0:IMAXBK+1, 0:JMAXBK+1) )
183     ALLOCATE( m%T( 0:IMAXBK+1, 0:JMAXBK+1) )
184     ALLOCATE( m%Ttmp(0:IMAXBK+1, 0:JMAXBK+1) )
185     ALLOCATE( m%dt( 0:IMAXBK+1, 0:JMAXBK+1) )
186     ALLOCATE( m%V2nd(0:IMAXBK+1, 0:JMAXBK+1) )
187     ALLOCATE( m%term(0:IMAXBK+1, 0:JMAXBK+1) )
188     ALLOCATE( m%Ayi( 0:IMAXBK+1, 0:JMAXBK+1) )
189     ALLOCATE( m%Axi( 0:IMAXBK+1, 0:JMAXBK+1) )
190     ALLOCATE( m%Ayj( 0:IMAXBK+1, 0:JMAXBK+1) )
191     ALLOCATE( m%Axj( 0:IMAXBK+1, 0:JMAXBK+1) )
192     ALLOCATE( m%V( 0:IMAXBK, 0:JMAXBK ) )
193     ALLOCATE( m%yPP( 0:IMAXBK, 0:JMAXBK ) )
194     ALLOCATE( m%yNP( 0:IMAXBK, 0:JMAXBK ) )
195     ALLOCATE( m%yNN( 0:IMAXBK, 0:JMAXBK ) )
196     ALLOCATE( m%yPN( 0:IMAXBK, 0:JMAXBK ) )
197     ALLOCATE( m%xNN( 0:IMAXBK, 0:JMAXBK ) )
198     ALLOCATE( m%xPN( 0:IMAXBK, 0:JMAXBK ) )
199     ALLOCATE( m%xPP( 0:IMAXBK, 0:JMAXBK ) )
200     ALLOCATE( m%xNP( 0:IMAXBK, 0:JMAXBK ) )
201 END DO
202
203 ! HEADER
204 READ(BLKFILE, *)
205 DO IB = 1, MYNBLK

```



```

206      b => blocks(IB)
207      ! FOR EACH BLOCK, READ BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
208      ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
209      ! NORTH EAST SOUTH WEST
210      READ(BLKFILE, 22) b%ID, b%IMIN, b%JMIN, b%SIZE, &
211      b%IMINLOC, b%IMAXLOC, b%JMINLOC, b%JMAXLOC, &
212      b%NB%N, b%NP%N, b%NBLOC%N, &
213      b%NB%S, b%NP%S, b%NBLOC%S, &
214      b%NB%E, b%NP%E, b%NBLOC%E, &
215      b%NB%W, b%NP%W, b%NBLOC%W, &
216      b%NB%NE, b%NP%NE, b%NBLOC%NE, &
217      b%NB%SE, b%NP%SE, b%NBLOC%SE, &
218      b%NB%SW, b%NP%SW, b%NBLOC%SW, &
219      b%NB%NW, b%NP%NW, b%NBLOC%NW, &
220      b%ORIENT
221
222      END DO
223      CLOSE(BLKFILE)
224
225      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
226      !!! READ SOLUTION RESTART FILES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
227      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
228
229      ! MAKE FILE NAME
230      IF (MYID<10) THEN
231        ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
232        WRITE(procname, '(A,I1)') '0', MYID
233      ELSE
234        WRITE(procname, '(I2)') MYID
235      END IF
236      xfile = "p" // procname // ".grid"
237      qfile = "p" // procname // ".T"
238      CALL readPlot3D(blocks, xfile, qfile)
239
240      END SUBROUTINE read_config
241
242      SUBROUTINE plot3D(blocks, NBLKS, xfile, qfile)
243        ! write plt 2d file given blocks, number of blocks,
244        ! x and q file names (no file extension), and the bounds for writing
245        ! (0 for real grid, 1 to include ghosts)
246        IMPLICIT NONE
247
248        TYPE(BLKTYPE) :: blocks(:)
249        INTEGER :: IBLK, I, J, NBLKS, bound = 1
250        ! OUTPUT FILES (without file extension)
251        CHARACTER(20) :: xfile, qfile
252
253        ! FORMAT STATEMENTS
254        ! I --> Integer, number following is number of sig figs
255        ! E --> scientific notation,
256        ! before decimal is sig figs of exponent?
257        ! after decimal is sig figs of value
258        ! number before letter is how many entries on single line
259        ! before newline (number of columns)
260
261        10      FORMAT(I10)
262        20      FORMAT(10I10)
263        30      FORMAT(10E20.8)
264
265        !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
266
267        ! OPEN FILES
268        OPEN(UNIT=gridUnit, FILE = TRIM(xfile) // '.form.xyz', FORM='formatted')
269        OPEN(UNIT=tempUnit, FILE = TRIM(qfile) // '.form.dat', FORM='formatted')
270
271        ! WRITE TO GRID FILE
272        WRITE(gridUnit, 10) NBLKS
273        WRITE(gridUnit, 20) ( IMAXBLOCK, JMAXBLOCK, IBLK=1, NBLKS)
274        ! WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
275        DO IBLK = 1, NBLKS
276          WRITE(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1-bound, IMAXBLOCK+bound), J=1-bound, JMAXBLOCK+bound), &

```

```

275          ( (blocks(IBLK)%mesh%y(I,J), I=1-bound, IMAXBK+bound), J=1-bound, JMAXBK+bound)
276      END DO
277
278
279      ! WRITE TO TEMPERATURE FILE
280      ! When read in paraview, 'density' will be equivalent to temperature
281      WRITE(tempUnit, 10) NBLKS
282      WRITE(tempUnit, 20) ( IMAXBK, JMAXBK, IBLK=1, NBLKS)
283      DO IBLK = 1, NBLKS
284
285          WRITE(tempUnit, 30) tRef,dum,dum,dum
286          WRITE(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1-bound, IMAXBK+bound), J=1-bound, JMAXBK+bound), &
287              ( (blocks(IBLK)%mesh%T(I,J), I=1-bound, IMAXBK+bound), J=1-bound, JMAXBK+bound), &
288              ( (blocks(IBLK)%mesh%T(I,J), I=1-bound, IMAXBK+bound), J=1-bound, JMAXBK+bound), &
289              ( (blocks(IBLK)%mesh%T(I,J), I=1-bound, IMAXBK+bound), J=1-bound, JMAXBK+bound)
290      END DO
291
292      ! CLOSE FILES
293      CLOSE(gridUnit)
294      CLOSE(tempUnit)
295
296      !!! UNFORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
297
298      ! OPEN FILES
299      OPEN(UNIT=gridUnit,FILE = TRIM(xfile) // '.xyz',FORM='unformatted')
300      OPEN(UNIT=tempUnit,FILE = TRIM(qfile) // '.dat',FORM='unformatted')
301
302      ! WRITE TO GRID FILE (UNFORMATTED)
303      ! (Paraview likes unformatted better)
304      WRITE(gridUnit) NBLKS
305      WRITE(gridUnit) ( IMAXBK, JMAXBK, IBLK=1, NBLKS)
306      ! WRITE(gridUnit) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
307      DO IBLK = 1, NBLKS
308          WRITE(gridUnit) ( (blocks(IBLK)%mesh%x(I,J), I=1, IMAXBK), J=1, JMAXBK), &
309              ( (blocks(IBLK)%mesh%y(I,J), I=1, IMAXBK), J=1, JMAXBK)
310      END DO
311
312
313      ! WRITE TO TEMPERATURE FILE
314      ! When read in paraview, 'density' will be equivalent to temperature
315      WRITE(tempUnit) NBLKS
316      WRITE(tempUnit) ( IMAXBK, JMAXBK, IBLK=1, NBLKS)
317      DO IBLK = 1, NBLKS
318
319          WRITE(tempUnit) tRef,dum,dum,dum
320          WRITE(tempUnit) ( (blocks(IBLK)%mesh%T(I,J), I=1, IMAXBK), J=1, JMAXBK), &
321              ( (blocks(IBLK)%mesh%T(I,J), I=1, IMAXBK), J=1, JMAXBK), &
322              ( (blocks(IBLK)%mesh%T(I,J), I=1, IMAXBK), J=1, JMAXBK), &
323              ( (blocks(IBLK)%mesh%T(I,J), I=1, IMAXBK), J=1, JMAXBK)
324      END DO
325
326      ! CLOSE FILES
327      CLOSE(gridUnit)
328      CLOSE(tempUnit)
329  END SUBROUTINE plot3D
330
331  SUBROUTINE readPlot3D(blocks, xfile, qfile)
332      IMPLICIT NONE
333
334      TYPE(BLKTYPE) :: blocks(:)
335      INTEGER :: IBLK, I, J, NBLKS
336      INTEGER :: NBLKREAD, IMAXBKREAD, JMAXBKREAD, bound = 1
337      REAL(KIND=8) :: dum1, dum2, dum3, dum4
338      ! OUTPUT FILES (without file extension)
339      CHARACTER(20) :: xfile, qfile
340
341      ! FORMAT STATEMENTS
342      ! I --> Integer, number following is number of sig figs
343      ! E --> scientific notation,

```

```

344         ! before decimal is sig figs of exponent?
345         ! after decimal is sig figs of value
346         ! number before letter is how many entries on single line
347         ! before newline (number of columns)
348     10     FORMAT(I10)
349     20     FORMAT(10I10)
350     30     FORMAT(10E20.8)
351
352     !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
353
354     ! OPEN FILES
355     OPEN(UNIT=gridUnit,FILE = TRIM(xfile) // '.form.xyz',FORM='formatted')
356     OPEN(UNIT=tempUnit,FILE = TRIM(qfile) // '.form.dat',FORM='formatted')
357
358     ! READ GRID FILE
359     READ(gridUnit, 10) NBLKREAD
360     READ(gridUnit, 20) ( IMAXBKREAD, JMAXBKREAD, IBLK=1, NBLKREAD)
361     DO IBLK = 1, NBLKREAD
362         READ(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound), &
363                             ( (blocks(IBLK)%mesh%y(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound)
364     END DO
365
366     ! READ TEMPERATURE FILE
367     READ(tempUnit, 10) NBLKREAD
368     READ(tempUnit, 20) ( IMAXBKREAD, JMAXBKREAD, IBLK=1, NBLKREAD)
369     DO IBLK = 1, NBLKREAD
370
371         READ(tempUnit, 30) tRef,dum,dum,dum
372         READ(tempUnit, 30) dum1, dum2, dum3, dum4
373         READ(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound), &
374                             ( (blocks(IBLK)%mesh%T(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound), &
375                             ( (blocks(IBLK)%mesh%T(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound), &
376                             ( (blocks(IBLK)%mesh%T(I,J), I=1-bound,IMAXBK+bound), J=1-bound,JMAXBK+bound)
377     END DO
378
379     ! CLOSE FILES
380     CLOSE(gridUnit)
381     CLOSE(tempUnit)
382
383
384     END SUBROUTINE readPlot3D
385
386     ! SUBROUTINE readPlot3D(blocks)
387     ! IMPLICIT NONE
388
389     ! TYPE(BLKTYPE) :: blocks(:)
390     ! INTEGER :: IBLK, I, J
391     ! ! READ INFO FOR BLOCK DIMENSIONS
392     ! INTEGER :: NBLKREAD, IMAXBKREAD, JMAXBKREAD
393     ! ! OUTPUT FILES
394     ! CHARACTER(20) :: xfile, qfile
395
396     ! FORMAT STATEMENTS
397     ! ! I --> Integer, number following is number of sig figs
398     ! ! E --> scientific notation,
399     ! ! before decimal is sig figs of exponent?
400     ! ! after decimal is sig figs of value
401     ! ! number before letter is how many entries on single line
402     ! ! before newline (number of columns)
403     ! 10     FORMAT(I10)
404     ! 20     FORMAT(10I10)
405     ! 30     FORMAT(10E20.8)
406
407     ! !!! FORMATTED !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
408
409     ! OPEN FILES
410     ! ! OPEN(UNIT=gridUnit,FILE= TRIM(casedir) // 'grid_form.xyz',FORM='formatted')
411     ! ! OPEN(UNIT=tempUnit,FILE= TRIM(casedir) // 'T_form.dat',FORM='formatted')
412     ! OPEN(UNIT=gridUnit,FILE= 'grid_form.xyz',FORM='formatted')

```

```

413 !      OPEN(UNIT=tempUnit,FILE= 'T_form.dat',FORM='formatted')
414
415 !      ! READ GRID FILE
416 !      READ(gridUnit, 10) NBLKREAD
417 !      READ(gridUnit, 20) ( IMAXBLKREAD, JMAXBLKREAD, IBLK=1, NBLKREAD)
418 !      !      WRITE(gridUnit, 20) ( blocks(IBLK)%IMAX, blocks(IBLK)%JMAX, IBLK=1, NBLK)
419 !      DO IBLK = 1, NBLKREAD
420 !          READ(gridUnit, 30) ( (blocks(IBLK)%mesh%x(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
421 !              ( (blocks(IBLK)%mesh%y(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
422 !      END DO
423
424
425 !      ! READ TEMPERATURE FILE
426 !      ! When read in paraview, 'density' will be equivalent to temperature
427 !      READ(tempUnit, 10) NBLKREAD
428 !      READ(tempUnit, 20) ( IMAXBLKREAD, JMAXBLKREAD, IBLK=1, NBLKREAD)
429 !      DO IBLK = 1, NBLKREAD
430
431 !          READ(tempUnit, 30) tRef,dum,dum,dum
432 !          READ(tempUnit, 30) ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
433 !              ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
434 !              ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK), &
435 !              ( (blocks(IBLK)%mesh%T(I,J), I=1,IMAXBLK), J=1,JMAXBLK)
436 !      END DO
437
438 !      ! CLOSE FILES
439 !      CLOSE(gridUnit)
440 !      CLOSE(tempUnit)
441 !      END SUBROUTINE readPlot3D
442
443 !      SUBROUTINE compositePlot3D()
444 !          type(blktype), ALLOCATABLE :: blocks(:)
445 !          type(proctype), target :: procs(nprocs)
446 !          type(proctype), pointer :: p
447 !          CHARACTER(2) :: procname
448 !          CHARACTER(20) :: xfile, qfile
449
450 !          integer :: procsort(NBLK), IDSort(NBLK), I, ii
451 !          allocate(blocks(NBLK))
452 !          ! read block amalgamation file
453 !          OPEN(UNIT=55,FILE = 'blockrebuild.dat',FORM='formatted')
454 !          read(55,*)
455 !          do I = 1, NBLK
456 !              read(55,*) Ii, procsort(I), IDSort(I)
457 !          end do
458 !          CLOSE(55)
459
460 !          OPEN(UNIT=65,FILE = 'procrebuild.dat',FORM='formatted')
461
462 !          do i = 1, NPROCS
463 !              p => procs(I)
464 !              READ(65,*) p%NBLK
465 !              allocate(p%blocks(p%NBLK))
466
467 !              IF (p%ID<10) THEN
468 !                  ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
469 !                  WRITE(procname, '(A,I1)') '0', p%ID
470 !              ELSE
471 !                  WRITE(procname, '(I2)') p%ID
472 !              END IF
473 !              xfile = "p" // procname // ".grid"
474 !              qfile = "p" // procname // ".T"
475 !              call readplot3d(p%blocks, xfile, qfile)
476
477 !          end do
478 !          CLOSE(65)
479
480 !          do i = 1, nblk
481 !              blocks(I) = procs( procsort(i) )%blocks( idsSort(i) )

```

```

482 !           end do
483
484
485 !           call plot3d(blocks, nblk, 'grid', 'T')
486
487
488
489 !       END SUBROUTINE compositePlot3D
490
491
492 SUBROUTINE write_res(res_hist)
493     TYPE(RESLIST), POINTER :: res_hist
494     ! pointer to iterate linked list
495     TYPE(RESLIST), POINTER :: hist
496
497     ! open residual file
498 !       OPEN(UNIT=resUnit,FILE= TRIM(casedir) // 'res_hist.dat')
499     OPEN(UNIT=resUnit,FILE = 'res_hist.dat')
500     ! column headers
501     WRITE(resUnit,*) 'ITER      RESID'
502
503     ! point to residual linked list
504     hist => res_hist
505     ! skip first link, empty from iteration loop design
506     hist => hist%next
507     ! write residual history to file until list ends
508     DO
509         IF ( .NOT. ASSOCIATED(hist) ) EXIT
510         ! write iteration and residual in two columns
511         WRITE(resUnit,*) hist%iter, hist%res
512         hist => hist%next
513     END DO
514
515     CLOSE(resUnit)
516 END SUBROUTINE write_res
517
518
519 END MODULE IO

```

Listing 4: Code for saving formatted multiblock PLOT3D solution files and reading restart files