MAE 267 – Project 4
Parallel, Multi-Block, Finite-Volume Methods For Solving 2D Heat Conduction

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

1 Statement of Problem

This analysis 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 will be adapted to solve multi-block decompositions on multiple processors for the 501x501 grid decomposed into 10x10 blocks running on 4 to 8 processors. This stage steps closer to that solution by performing domain decomposition and processor distribution for 5x4 and 10x10 blocks on 4 and 6 processors, and saving the decompositions to restart files to be loaded by 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.

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

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

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.

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

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 refered 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.

$$
\begin{aligned}
&WGEOM = 1 \\
&WCOMM = FACTOR \cdot \frac{(IMAXBLK+2)(JMAXBLK+2)}{(2 \cdot IMAXBLK)+(2 \cdot IMAXBLK)+4} \\
&SIZE = (WGEOM \cdot GEOM) + (WCOMM \cdot COMM)
\end{aligned}
\tag{11}
$$

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 Indentification

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 produces 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 restart files 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.

3 Results and Discussion

Neighbor connectivity files were written for each processor for each case and samples are shown in Appendix A. From these files, neighbor block, processor, neighbor block local index, block dimmension, and block size can be read.

Theoretical load balance was also calculated for each processor based on the load determined by the previously described criteria. Load balance results are presented below in Table 2.

Table 2: Processor Theoretical Load Balances

NPROCS	4	4	6	6
MxN	5x4	10x10	5x4	10x10
Proc0	1.0133	1.0000	0.8802	1.0182
Proc1	1.0133	1.0000	0.8802	1.0182
Proc2	0.98672	1.0000	0.8802	0.9724
Proc3	0.98672	1.0000	1.0908	0.9724
Proc4	N/A	N/A	1.1005	1.0094
Proc5	N/A	N/A	0.8502	1.0094

It can be seen that for these large-block decompositions, the code is well able to balance the loads of the processors. 4 processors proved to be better for load balancing as 4 is a multiple of the number of blocks for both decompositions. Perfect balancing was achieved for the 4 processor 10x10 case, but not for the 4x5 case so I will look into why the distribution algorithm was not as efficient in this case.

4 Conclusion

The core of the work in this project was the development of the processor distributing algorithm. Many factors can go into optimizing processor load balancing, and it was an iteresting problem following the logic behind the process. As my algorithm is currently designed, I believe the weighting of different communication costs will need to be tuned once I am producing results with which to benchmark. I will also look deeper into the algorithm as it does not produce as consistent results as I would have thought.

Appendix A: Sample Connectivity Files

```
NBLK IMAXBLK JMAXBLK
    5  101  126
   ID  IMIN JMIN SIZE   NNB   NNP NLOC   SNB   SNP SLOC   ENB   ENP ELOC   WNB   WNP WLOC NENB NENP   NEL SENB SENP   SEL SWNB S
    7  101  12625910  -12     3    1    -2     2    2    -8     1    1    -6     2    4   13    0     2   -3    3     2   -1
   13  201  25125910   18     0    3    -8     1    1   -14     1    2   -12     3    1  -19    1     3   -9    2     1    7
   18  201  37622872    0    -1    0    13     0    2   -19     1    3   -17     3    3    0   -1     0  -14    1     2  -12
   11    1  25122128   16     0    5    -6     2    4   -12     3    1     0    -1    0  -17    3     3    7    0     1    0
   16    1  37619092    0    -1    0    11     0    4   -17     3    3     0    -1    0    0   -1     0  -12    3     1    0
```

Listing 1: 4 processors 5x4 blocks processor0

```
NBLK IMAXBLK JMAXBLK
   25   51   51
   ID  IMIN JMIN SIZE   NNB   NNP NLOC   SNB   SNP SLOC   ENB   ENP ELOC   WNB   WNP WLOC NENB NENP   NEL SENB SENP   SEL SWNB S
   12   51   51 5410   22     0    3     2     0   17   -13     1    1    11     0   19  -23    1     3   -3    1    17    1
   16  251   51 5410   26     0    4     6     0   18   -17     1    2   -15     3    1  -27    1     4   -7    1    18   -5
   22   51  101 5410   32     0    5    12     0    1   -23     1    3   -21     2   19  -33    1     5  -13    1     1   11
   26  251  101 5410   36     0    6    16     0    2   -27     1    4   -25     3    3  -37    1     6  -17    1     2  -15
   32   51  151 5410   42     0    7    22     0    3   -33     1    5    31     0   20  -43    1     7  -23    1     3  -21
   36  251  151 5410   46     0    8    26     0    4   -37     1    6   -35     3    5  -47    1     8  -27    1     4  -25
   42   51  201 5410   52     0    9    32     0    5   -43     1    7   -41     2   20  -53    1     9  -33    1     5   31
   46  251  201 5410   56     0   10    36     0    6   -47     1    8   -45     3    7  -57    1    10  -37    1     6  -35
   52   51  251 5410   62     0   11    42     0    7   -53     1    9    51     0   21  -63    1    11  -43    1     7  -41
   56  251  251 5410   66     0   12    46     0    8   -57     1   10   -55     3    9  -67    1    12  -47    1     8  -45
   62   51  301 5410   72     0   13    52     0    9   -63     1   11   -61     2   21  -73    1    13  -53    1     9   51
   66  251  301 5410   76     0   14    56     0   10   -67     1   12   -65     3   11  -77    1    14  -57    1    10  -55
   72   51  351 5410   82     0   15    62     0   11   -73     1   13    71     0   22  -83    1    15  -63    1    11  -61
   76  251  351 5410   86     0   16    66     0   12   -77     1   14   -75     3   13  -87    1    16  -67    1    12  -65
   82   51  401 5410   92     0   23    72     0   13   -83     1   15   -81     2   22  -93    1    23  -73    1    13   71
   86  251  401 5410   96     0   24    76     0   14   -87     1   16   -85     3   15  -97    1    24  -77    1    14  -75
    2   51    1 4656   12     0    1     0    -1    0    -3     1   17     1     0   25  -13    1     1    0   -1     0    0
    6  251    1 4656   16     0    2     0    -1    0    -7     1   18    -5     3   17  -17    1     2    0   -1     0    0
   11    1   51 4656  -21     2   19     1     0   25    12     0    1     0    -1    0   22    0     3    2    0    17    0
   31    1  151 4656  -41     2   20   -21     2   19    32     0    5     0    -1    0   42    0     7   22    0     3    0
   51    1  251 4656  -61     2   21   -41     2   20    52     0    9     0    -1    0   62    0    11   42    0     7    0
   71    1  351 4656  -81     2   22   -61     2   21    72     0   13     0    -1    0   82    0    15   62    0    11    0
   92   51  451 4656    0    -1    0    82     0   15   -93     1   23   -91     2   25    0   -1     0  -83    1    15  -81
   96  251  451 4656    0    -1    0    86     0   16   -97     1   24   -95     3   23    0   -1     0  -87    1    16  -85
    1    1    1 3904   11     0   19     0    -1    0     2     0   17     0    -1    0   12    0     1    0   -1     0    0
```

Listing 2: 4 processors 10x10 blocks processor0

```
NBLK IMAXBLK JMAXBLK
    3  101  126
   ID IMIN JMIN SIZE  NNB  NNP NLOC  SNB  SNP SLOC  ENB  ENP ELOC  WNB  WNP WLOC NENB NENP  NEL SENB SENP  SEL SWNB S
    9  301  12625910  -14    3    1    4   -1    0   10    0    2    8   -1    0  -15    2    2    5   -1    0   -3
   10  401  12622128  -15    2    2    5   -1    0    0   -1    0    9    0    1    0   -1    0    0   -1    0    4
   20  401  37619092    0   -1    0  -15    2    2    0   -1    0  -19    5    2    0   -1    0    0   -1    0  -14
```

Listing 3: 6 processors 5x4 blocks processor0

```
NBLK IMAXBLK JMAXBLK
   17   51   51
   ID IMIN JMIN SIZE  NNB  NNP NLOC  SNB  SNP SLOC  ENB  ENP ELOC  WNB  WNP WLOC NENB NENP  NEL SENB SENP  SEL SWNB S
   12   51   51 5410  -22    2    2   -2    4   11  -13    1    1  -11    4   13  -23    3    2   -3    5   11   -1
   18  351   51 5410  -28    2    3   -8    2   12  -19    1    2  -17    5    1  -29    3    3   -9    3   12   -7
   26  251  101 5410  -36    2    4  -16    4    1  -27    1    3  -25    5    2  -37    3    4  -17    5    1  -15
   34  151  151 5410  -44    2    5  -24    4    2  -35    1    4  -33    5    3  -45    3    5  -25    5    2  -23
   42   51  201 5410  -52    2    6  -32    4    3  -43    1    5  -41    4   14  -53    3    6  -33    5    3  -31
   48  351  201 5410  -58    2    7  -38    4    4  -49    1    6  -47    5    5  -59    3    7  -39    5    4  -37
   56  251  251 5410  -66    2    8  -46    4    5  -57    1    7  -55    5    6  -67    3    8  -47    5    5  -45
   64  151  301 5410  -74    2    9  -54    4    6  -65    1    8  -63    5    7  -75    3    9  -55    5    6  -53
   72   51  351 5410  -82    2   10  -62    4    7  -73    1    9  -71    4   15  -83    3   10  -63    5    7  -61
   78  351  351 5410  -88    2   11  -68    4    8  -79    1   10  -77    5    9  -89    3   11  -69    5    8  -67
   86  251  401 5410   96    0   16  -76    4    9  -87    1   11  -85    5   10  -97    1   16  -77    5    9  -75
    6  251    1 4656  -16    4    1    0   -1    0   -7    1   12   -5    5   12  -17    5    1    0   -1    0    0
   21    1  101 4656  -31    2   13  -11    4   13  -22    2    2    0   -1    0  -32    4    3   12    0    1    0
   51    1  251 4656  -61    2   14  -41    4   14  -52    2    6    0   -1    0  -62    4    7   42    0    5    0
   81    1  401 4656   91    0   17  -71    4   15  -82    2   10    0   -1    0  -92    2   15   72    0    9    0
   96  251  451 4656    0   -1    0   86    0   11  -97    1   16  -95    5   16    0   -1    0  -87    1   11  -85
   91    1  451 3904    0   -1    0   81    0   15  -92    2   15    0   -1    0    0   -1    0  -82    2   10    0
```

Listing 4: 6 processors 10x10 blocks processor0

Appendix B: Parallel, Multi-Block Grid Decomposition Code

```
! MAE 267
! PROJECT 4
! LOGAN HALSTROM
! 14 NOVEMBER 2015

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

! CONTENTS:

! CONSTANTS --> Module that reads, initializes, and stores constants.
    ! Math and material contants, solver parameters, block sizing
    ! CONTAINS:

    ! read_input:
        ! Reads grid/block size and other simulation parameters from
        ! "config.in" file.  Avoids recompiling for simple input changes

! BLOCKMOD --> Module that contains data types and functions pertaining to
    ! block mesh generation and solution.  Derived data types include;
    ! MESHTYPE containing node information like temperature, and area,
    ! NBRTYPE containing information about cell neighbors
    ! LNKLIST linked list for storing similar neighbor information
    ! CONTAINS:

        ! init_blocks
        ! Assign individual block global indicies, neighbor, BCs, and
        ! orientation information

        ! write_blocks
        ! Write block connectivity file with neighbor and BC info

        ! read_blocks
        ! Read block connectivity file

        ! init_mesh
        ! Create xprime/yprime non-uniform grid, then rotate by angle 'rot'.
        ! Allocate arrays for node parameters (i.e. temperature, cell area, etc)

        ! init_temp
        ! Initialize temperature across mesh with dirichlet BCs
        ! or constant temperature BCs for DEBUG=1

        ! set_block_bounds
        ! Calculate iteration bounds for each block to avoid overwriting BCs.
        ! Call after reading in mesh data from restart file

        ! init_linklists
        ! Calculate iteration bounds for each block to avoid overwriting BCs.
        ! Call after reading in mesh data from restart file

        ! update_ghosts
        ! Update ghost nodes of each block based on neightbor linked lists.
        ! Ghost nodes contain solution from respective block face/corner
        ! neighbor for use in current block solution.

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

        ! calc_cell_params
        ! calculate areas for secondary fluxes and constant terms in heat
        ! treansfer eqn. Call after reading mesh data from restart file

        ! calc_constants
        ! Calculate terms that are constant regardless of iteration
        !(time step, secondary volumes, constant term.)  This way,
```

```fortran
        ! they don’t need to be calculated within the loop at each iteration

        ! calc_temp
        ! Calculate temperature at all points in mesh, excluding BC cells.
        ! Calculate first and second derivatives for finite-volume scheme

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

MODULE CONSTANTS
    ! Initialize constants for simulation.  Set grid size.

    IMPLICIT NONE

    ! INCLUDE MPI FOR ALL SUBROUTINES THAT USE CONSTANTS
    INCLUDE "mpif.h"
    ! MPI PROCESSOR ID
    INTEGER :: MYID
    ! MPI ERROR STATUS, NUMBER OF MPI PROCESSORS
    INTEGER :: IERROR, NPROCS, request
    INTEGER :: STATUS(MPI_STATUS_SIZE)

    ! CFL number, for convergence (D0 is double-precision, scientific notation)
    REAL(KIND=8), PARAMETER :: CFL = 0.95D0
    ! Material constants (steel): thermal conductivity [W/(m*K)],
                                ! density [kg/m^3],
                                ! specific heat ratio [J/(kg*K)]
                                ! initial temperature
    REAL(KIND=8), PARAMETER :: k = 18.8D0, rho = 8000.D0, cp = 500.D0, T0 = 3.5D0
    ! Thermal diffusivity [m^2/s]
    REAL(KIND=8), PARAMETER :: alpha = k / (cp * rho)
    ! Pi, grid rotation angle (30 deg)
    REAL(KIND=8), PARAMETER :: pi = 3.141592654D0, rot = 30.D0*pi/180.D0
    ! ITERATION PARAMETERS
    ! Minimum Residual
    REAL(KIND=8) :: min_res = 0.00001D0
    ! Maximum number of iterations
    INTEGER :: max_iter = 1000000
    ! CPU Wall Times
    REAL(KIND=8) :: wall_time_total, wall_time_solve, wall_time_iter(1:5)
    ! read square grid size, Total grid size, size of grid on each block (local)
    INTEGER :: nx, IMAX, JMAX, IMAXBLK, JMAXBLK
    ! Dimensions of block layout, Number of Blocks
    INTEGER :: M, N, NBLK
    ! Block boundary condition identifiers
        ! If block face is on North,east,south,west of main grid, identify
        ! If boundary is on a different proc, multiply bnd type by proc boundary
    INTEGER :: NBND = -1, EBND = -2, SBND = -3, WBND = -4, BND=0, PROCBND = -1
    ! Output directory
    CHARACTER(LEN=18) :: casedir
    ! Debug mode = 1
    INTEGER :: DEBUG
    ! Value for constant temperature BCs for debugging
    REAL(KIND=8), PARAMETER :: TDEBUG = T0 - T0 * 0.5

CONTAINS

    SUBROUTINE read_input()
        ! Reads grid/block size and other simulation parameters from
        ! "config.in" file.  Avoids recompiling for simple input changes

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

        ! READ INPUTS FROM FILE
            !(So I don’t have to recompile each time I change an input setting)
!         WRITE(*,*) ''
```

```fortran
!         WRITE(*,*) 'Reading input...'
        OPEN (UNIT = 1, FILE = 'config.in')
        DO I = 1, 3
            ! Skip header lines
            READ(1,*)
        END DO
        ! READ GRIDSIZE (4th line)
        READ(1,*) nx
        ! READ BLOCKS (6th and 8th line)
        READ(1,*)
        READ(1,*) M
        READ(1,*)
        READ(1,*) N
        ! DEBUG MODE (10th line)
        READ(1,*)
        READ(1,*) DEBUG

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

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

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

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

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

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
```

```
! BLOCK SUBDOMAIN DIAGRAM WITH BOUNDARY CONDITIONS (FOR MXN = 4X5)
!
!                     NBND = -1
!
!     JMAX -|-----|-----|-----|-----|-----|
!           |     |     |     |     |     |
!           | 16  | 17  | 18  | 19  | 20  |
!           |-----|-----|-----|-----|-----|
!           |     |     |     |     |     |
!     J^    | 11  | 12  | 13  | 14  | 15  |
!    M=4    |-----|-----|-----|-----|-----|  EBND = -2
!  WBND=-4  |     |     |     |     |     |
!           |  6  |  7  |  8  |  9  | 10  |
!           |-----|-----|-----|-----|-----|
!           |     |     |     |     |     |
!           |  1  |  2  |  3  |  4  |  5  |
!        1 -|-----|-----|-----|-----|-----|
!           |                             |
!           1             I  ->          IMAX
!                          N=5
!                       SBND = -3
!
!  Where IMAX, N, NBND, etc are all global variable stored in CONSTANTS
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! LOCAL/GLOBAL BLOCK INDICIES
!
!                        GLOBAL
!    block(IBLK)%IMIN              block(IBLK)%IMAX
!           |                               |
!  JMAXBLK -|-------------------------------|- block(IBLK)%JMAX
!           |                               |
!           |                               |
!           |                               |
!     L     |                               |    G
!     O     |                               |    L
!     C J^  |      LOCAL BLOCK INDICES      |    O
!     A     |                               |    B
!     L     |                               |    A
!           |                               |    L
!           |                               |
!           |                               |
!        1 -|-------------------------------|- block(IBLK)%JMIN
!           |                               |
!           1              I ->         IMAXBLK
!                         LOCAL
!
!  Where block is block data type, IBLK is index of current block
!
!  Convert from local to global (where I is local index):
!  Iglobal = block(IBLK)%IMIN + (I-1)
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! LOCAL BLOCK INDICIES WITH GHOST NODES
!
! JMAXBLK+1 |---|-------------------------------|---|
!           |NWG|       NORTH GHOST NODES       |NEG|
!   JMAXBLK |---|-------------------------------|---|
!           |   |                               |   |
!           | W |                               | E |
!           | E |                               | A |
!           | S |                               | S |
!           | T |                               | T |
!       J^  |   |      LOCAL BLOCK INDICES      |   |
!           | G |                               | G |
!           | H |                               | H |
!           | O |                               | O |
```

```
!           | S |                              | S |
!           | T |                              | T |
!           |   |                              |   |
!         1 |---|------------------------------|---|
!           |SWG|      SOUTH GHOST NODES       |SEG|
!         0 |---|------------------------------|---|
!           |   1                          IMAXBLK |
!           0                I ->              IMAXBLK+1
!
!  Where NWG, NEG, etc are corner ghosts
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! BLOCK NEIGHBORS
!
!               |                |
!               |     North      |
!             NW|  (IBLK + N)    |NE
! (IBLK + N - 1)|                |(IBLK + N + 1)
! ----------------------------------------------
!               |                |
!     West      |   Current      |    East
!   (IBLK - 1)  |     (IBLK)     |  (IBLK + 1)
!               |                |
! ----------------------------------------------
!             SW|                |SE
! (IBLK - N - 1)|     South      |(IBLK - N + 1)
!               |  (IBLK - N)    |
!               |                |
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! LOCAL ITERATION BOUNDS (TO INCLUDE GHOSTS/EXCLUDE BC'S)
!  -------------
! | ~ ~ = BC    |
! | . . = Ghost |
!  -------------
! JMAXBLK+1 -|---|--------------|---|
!            | ~ |. . . . . . . | . |
!   JMAXBLK -|---|--------------|---|  JMAXBLK -|---|--------------|---|
!            | ~ |              | . |           | ~ |~ ~ ~ ~ ~ ~ ~ | ~ |
!            | ~ |              | . | JMAXBLK-1-|---|--------------|---|
!       J^   | ~ |   M=1, N=1   | . |           | . |              | ~ |
!            | ~ |              | . |           | . |              | ~ |
!            | ~ |              | . |      J^   | . |   M=M, N=N   | ~ |
!         2 -|---|--------------|---|           | . |              | ~ |
!            | ~ |~ ~ ~ ~ ~ ~ ~ | ~ |           | . |              | ~ |
!         1 -|---|--------------|---|         1 -|---|--------------|---|
!            |   2          IMAXBLK |           | . |. . . . . . . | ~ |
!            1         I ->     IMAXBLK+1     0 -|---|--------------|---|
!                                               |   1        IMAXBLK-1 |
!                                               0         I ->     IMAXBLK
!
!    Solver  : I = 1 --> IMAXBLK          | Solver  : I = 0 --> IMAXBLK-1
!        to get: dT: 1 --> IMAXBLK+1      |     to get: dT: 0 --> IMAXBLK
!    Update T: I = 2 --> IMAXBLK          | Update T: I = 1 --> IMAXBLK-1
!        (avoid updating BC's at I=1)     |     (avoid updating BC's at I=IMAXBLK)
!        (IMAXBLK+1 ghost updated later)  |     (I=0 ghost updated later)
!
!  RESULT:  Set local iteration bounds IMINLOC, IMAXLOC, etc according to solver limits
!           Update temperature starting at IMINLOC+1 to avoid lower BC's
!               (upper BC's automatically avoided by explicit scheme solving for i+1)
!
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! INITIALIZE VARIABLES/DEPENDANCIES
USE CONSTANTS

```

```fortran
    IMPLICIT NONE
    PUBLIC

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

    ! DERIVED DATA TYPE FOR GRID INFORMATION

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

    ! DATA TYPE FOR INFORMATION ABOUT NEIGHBORS

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

    ! DERIVED DATA TYPE WITH INFORMATION PERTAINING TO SPECIFIC BLOCK

    TYPE BLKTYPE
        ! DER. DATA TYPE STORES LOCAL MESH INFO
        TYPE(MESHTYPE) :: mesh
        ! IDENTIFY FACE AND CORNER NEIGHBOR BLOCKS AND PROCESSORS
        ! AND LOCAL PROCESSOR BLOCK INDICIES
        TYPE(NBRTYPE) :: NB, NP, NBLOC
        ! BLOCK NUMBER, PROCESSOR NUMBER
        INTEGER :: ID, procID
        ! GLOBAL INDICIES OF MINIMUM AND MAXIMUM INDICIES OF BLOCK
        INTEGER :: IMIN, IMAX, JMIN, JMAX
        ! LOCAL ITERATION BOUNDS TO AVOID UPDATING BC'S + UTILIZE GHOST NODES
        INTEGER :: IMINLOC, JMINLOC, IMAXLOC, JMAXLOC, IMINUPD, JMINUPD
        ! BLOCK LOAD PARAMETERS FOR PROCESSOR LOAD BALANCING
        REAL(KIND=8) :: SIZE
        ! BLOCK ORIENTATION
        INTEGER :: ORIENT
    END TYPE BLKTYPE

    ! DATA TYPE FOR PROCESSOR INFORMATION

    TYPE PROCTYPE
        ! Information pertaining to each processor: procID, number of blocks
        ! on proc
        INTEGER :: ID, NBLK=0
        ! processor load, load balance
        REAL(KIND=8) :: load=0.D0, balance=0.D0
        ! Blocks contained on processor
        TYPE(BLKTYPE), ALLOCATABLE :: blocks(:)
    END TYPE PROCTYPE

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

    TYPE LNKLIST
        ! Next element in linked list
        TYPE(LNKLIST), POINTER :: next
```

```
        ! Identify what linked list belongs to
        INTEGER :: ID
    END TYPE LNKLIST

    ! Collection of linked lists for faces and corners

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

CONTAINS

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

    SUBROUTINE init_blocks(b)
        ! Assign individual block global indicies, neighbor, BCs, and
        ! orientation information

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

        ! STEP THROUGH BLOCKS, ASSIGN IDENTIFYING INFO

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

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

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

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

                ! ASSIGN NEIGHBORS
                ! (Numbers of face and corner neighbor blocks)
                ! (if boundary face, assign bc later)
                NB%N  = IBLK + N
                NB%S  = IBLK - N
                NB%E  = IBLK + 1
                NB%W  = IBLK - 1
                NB%NE = IBLK + N + 1
                NB%NW = IBLK + N - 1
                NB%SW = IBLK - N - 1
                NB%SE = IBLK - N + 1

                ! ASSIGN BOUNDARY CONDITIONS

                ! Assign faces and corners on boundary of the actual
                ! computational grid with number corresponding to which
                ! boundary they are on.
                    ! Corners on actual corners of the computational grid are
                    ! ambiguously assigned.
```

```fortran
                IF ( b(IBLK)%JMAX == JMAX ) THEN
                    ! NORTH BLOCK FACE AND CORNERS ARE ON MESH NORTH BOUNDARY
                        ! AT ACTUAL CORNERS OF MESH, CORNERS ARE AMBIGUOUS
                    NB%N  = BND
                    NB%NE = BND
                    NB%NW = BND
                END IF
                IF ( b(IBLK)%IMAX == IMAX ) THEN
                    ! EAST BLOCK FACE IS ON MESH EAST BOUNDARY
                    NB%E  = BND
                    NB%NE = BND
                    NB%SE = BND

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

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

            END DO
        END DO
    END SUBROUTINE init_blocks

    SUBROUTINE dist_blocks(blocks, procs)
        ! Distribute blocks to processors.  Calculate processor load of each
        ! block based on geometry and communication costs and weighting factors
        ! for each.
        ! Initialize processor list with proc ID's and allocate proc block lists
        ! Distribute blocks to processors by sorting blocks in decreasing order
        ! of load, then distributing sequentially to the processor with the
        ! least load.
        ! Calculate load balance of all processors.

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(BLKTYPE), POINTER :: b
        TYPE(NBRTYPE), POINTER :: NB
        ! PROCESSOR DATA TYPE
        TYPE(PROCTYPE), TARGET :: procs(:)
        TYPE(PROCTYPE), POINTER :: p
        ! COUNTER VARIABLE
        INTEGER :: IBLK, I, IPROC
        ! CURRENT BLOCK DIMENSIONS
        INTEGER :: NXLOC, NYLOC
        ! COMPUTATIONAL COST PARAMETERS
        ! (geometric (grid size) and communication weights)
        INTEGER :: GEOM=0, COMM=0, MAXCOMM, MAXGEOM
        ! WEIGHTS FOR LOAD BALANCING
        ! (geometry, communication, fudge factor, perfect load balance)
        REAL(KIND=8) :: WGEOM = 1.0D0, WCOMM, FACTOR=1.D0, PLB=0.D0
        ! VARIABLES FOR SORTING BLOCKS BY LOAD
        ! maximum block load
        REAL(KIND=8) :: MAXSIZE=0.D0, MINLOAD
        ! 'sorted' is list of IDs of blocks in order of size greatest to least
        ! 'claimed' idicates if a block has already been sorted (0/1 --> unsorted/sorted)
            ! initial list is all zeros
        ! 'IMAXSIZE' is index of remaining block with greatest size
```

```fortran
        INTEGER :: sorted(NBLK), claimed(NBLK), IMAXSIZE, IMINLOAD

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! SET WEIGHTING FACTORS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! SET COMMUNICATION WEIGHT TO BE PROPORTIONAL TO GEOMETRY
        ! Maximum geometry cost is all cells with ghost nodes at all faces
        MAXGEOM = ( IMAXBLK + 2.D0 ) * ( JMAXBLK + 2.D0 )
        ! Maximum communication cost is all face boundaries plus four corners
        MAXCOMM = ( 2.D0 * IMAXBLK ) + ( 2.D0 * JMAXBLK ) + 4.D0
        ! Put comm cost on same scale as geom
        WCOMM = FACTOR * ( DFLOAT(MAXGEOM) / DFLOAT(MAXCOMM) )
        ! COME UP WITH A BETTER WEIGHTING FACTOR IN PROJECT 5 WHEN YOU CAN BENCHMARK TIMES!!!!!!!!!!!!!!!!!!!!!!!!!!!

        10      FORMAT(10A12)
        WRITE(*,*)
        WRITE(*,*) 'Processor Load Weighting Factors:'
        WRITE(*,*) 'WGEOM=', WGEOM, 'WCOMM=', WCOMM
        WRITE(*,*)
        WRITE(*,*) 'SIZE = WGEOM*GEOM + WCOMM*COMM'
        WRITE(*,*)
        WRITE(*,*) 'Block Load Factors:'
        WRITE(*,10) 'BLKID', 'GEOM', 'COMM', 'SIZE'

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! CALC BLOCK WEIGHTS FOR PROCESSOR LOAD BALANCING !!!!!!!!!!!!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! need local block sizes
        CALL set_block_bounds(blocks)
        DO IBLK = 1, NBLK
            b => blocks(IBLK)
            NB => b%NB

            ! RESET COST SUMS
            GEOM = 0
            COMM = 0

            ! LOCAL BLOCK DIMENSIONS
            NXLOC = b%IMAXLOC - b%IMINLOC
            NYLOC = b%JMAXLOC - b%JMINLOC

            ! GEOMETRIC BLOCK WEIGHT ("VOLUME")
            GEOM = NXLOC * NYLOC

            ! COMMUNICATION BLOCK WEIGHT
            ! NORTH
            IF (NB%N > 0) THEN
                ! Interior faces have communication cost for populating ghosts
                COMM = COMM + IMAXBLK
            END IF
            ! EAST
            IF (NB%E > 0) THEN
                COMM = COMM + JMAXBLK
            END IF
            ! SOUTH
            IF (NB%S > 0) THEN
                COMM = COMM + IMAXBLK
            END IF
            ! WEST
            IF (NB%W > 0) THEN
                COMM = COMM + JMAXBLK
            END IF
            ! NORTHEAST
            IF (NB%N > 0) THEN
                ! Interior corners have communication cost for populating ghosts
                COMM = COMM + 1
            END IF
```

```
            ! SOUTHEAST
            IF (NB%E > 0) THEN
                COMM = COMM + 1
            END IF
            ! SOUTHWEST
            IF (NB%S > 0) THEN
                COMM = COMM + 1
            END IF
            ! NORTHWEST
            IF (NB%W > 0) THEN
                COMM = COMM + 1
            END IF

            ! CALCULATE TOTAL LOAD OF BLOCK WITH WEIGHTING FACTORS
            b%SIZE = 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 = PLB / DFLOAT(NPROCS)

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

        DO IBLK = 1, NBLK

            ! Reset current max size
            MAXSIZE = 0.D0

            ! 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

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! 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) )
```

```fortran
        END DO

        write(*,*) NBLK
        DO I = 1, NBLK
            b => blocks( sorted(I) )
            write(*,*) b%ID, b%SIZE
        END DO

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! DISTRIBUTE TO PROCESSOR WITH LEAST LOAD !!!!!!!!!!!!!!!!!!!!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        ! LOOP THROUGH BLOCKS IN DECREASING ORDER OF SIZE
        DO I = 1, NBLK
            ! sorted gives the indicies of blocks sorted by size
            b => blocks( sorted(I) )

            ! Reset minimum load
            MINLOAD = 1.0E10
            ! FIND CURRENT PROCESSOR WITH LEAST LOAD
            DO IPROC = 1, NPROCS
                p => procs(IPROC)

                IF (p%load<MINLOAD) THEN
                    MINLOAD = p%load
                    IMINLOAD = IPROC
                END IF
            END DO
            ! ASSIGN BLOCK TO MIN. LOAD PROC
            write(*,*) b%ID, procs(IMINLOAD)%ID
            CALL assign_block( b, procs(IMINLOAD) )
        END DO

        ! CALC LOAD BALANCE
        20      FORMAT(10A13)
        WRITE(*,*)
        WRITE(*,*) 'Processor Load Balancing:'
        WRITE(*,20) 'ID', 'LOAD BALANCE'

        DO IPROC = 1, NPROCS
            procs(IPROC)%balance = procs(IPROC)%load / PLB

            WRITE(*,*) procs(IPROC)%ID, procs(IPROC)%balance
        END DO

    END SUBROUTINE dist_blocks

    SUBROUTINE assign_block(b, p)
        ! Assign block to given processor

        ! Block to assign (not list)
        TYPE(BLKTYPE), TARGET :: b
        ! Processor to assign to
        TYPE(PROCTYPE), TARGET :: p

        ! INCREMENT NUMBER OF BLOCKS ON PROC
        p%NBLK = p%NBLK + 1
        ! ADD BLOCK LOAD TO TOTAL PROCESSOR LOAD
        p%load = p%load + b%SIZE
        ! ADD BLOCK TO PROC
        p%blocks(p%NBLK) = b
        ! ADD BLOCK TO PROC
        p%blocks(p%NBLK) = b

    END SUBROUTINE assign_block

    SUBROUTINE init_neighbor_procs(blocks, procs)
        ! Initialize neighbor processor information for each block

```

```fortran
        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(BLKTYPE), POINTER :: bcur, bnbr
        ! PROCESSOR DATA TYPE
        TYPE(PROCTYPE), TARGET :: procs(:)
        TYPE(PROCTYPE), POINTER :: pcur, pnbr
        ! Neighbor information pointer
        TYPE(NBRTYPE), POINTER :: NBCUR, NBNBR, NPCUR
        ! COUNTER VARIABLES
            ! index of current processor, index of current proc's neighbor proc
        INTEGER :: IPCUR, IPNBR, IBCUR, IBNBR

        ! ASSIGN PROC INFORMATION TO PROC DATA TYPE LIST
        DO IPCUR = 1, NPROCS
            pcur => procs(IPCUR)

            ! ALL BLOCKS ASSIGNED TO CURRENT PROC ARE ON CURRENT PROC
            pcur%blocks%procID = pcur%ID
            ! DEFAULT ALL NEIGHBORS TO -1
                ! (indicates boundary with no neighbor if not reassigned later)
            DO IBCUR = 1, pcur%NBLK
                NPCUR => pcur%blocks(IBCUR)%NP

                NPCUR%N  = -1
                NPCUR%S  = -1
                NPCUR%E  = -1
                NPCUR%W  = -1
                NPCUR%NE = -1
                NPCUR%SE = -1
                NPCUR%SW = -1
                NPCUR%NW = -1
            END DO
        END DO

        ! FIND PROC WITH NEIGHBOR FOR EACH BLOCK
        DO IPCUR = 1, NPROCS
            pcur => procs(IPCUR)

            ! FOR EACH PROC, STEP THROUGH EACH CONTAINED BLOCK AND FIND NEIGHBORS
            DO IBCUR = 1, pcur%NBLK
                bcur => pcur%blocks(IBCUR)

                ! STEP THROUGH EACH NEIGHBOR PROCESSOR TO FIND NEIGHBOR BLOCK
                DO IPNBR = 1, NPROCS
                    pnbr => procs(IPNBR)

                    ! STEP THROUGH BLOCKS ON NEIGHBOR PROCESSORS
                    DO IBNBR = 1, pnbr%NBLK
                        bnbr => pnbr%blocks(IBNBR)

                        ! CHECK EACH FACE/CORNER FOR MATCH AND ASSIGN
                            ! (neighbor procID and local index of
                            !  neighbor block on neighbor proc)

                        ! NORTH
                        IF (bcur%NB%N == bnbr%ID) THEN
                            ! PROCESSOR CONTAINING NEIGHBOR BLOCK
                            bcur%NP%N = pnbr%ID
                            ! NEIGHBOR BLOCK LOCAL INDEX ON NEIGHBOR PROCESSOR
                                !(used to access neighbor on neighbor processor)
                            bcur%NBLOC%N = IBNBR

                            ! IF NEIGHBOR PROC IS DIFFERENT FROM CURRENT PROC,
                            ! COMMUNICATION WILL BE REQUIRED
                            ! (indicate processor boundary by making the block
                            !  neighbor number negative)
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%N = -bcur%NB%N
                            END IF
```

```
                        END IF
                        ! SOUTH
                        IF (bcur%NB%S == bnbr%ID) THEN
                            bcur%NP%S = pnbr%ID
                            bcur%NBLOC%S = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%S = -bcur%NB%S
                            END IF
                        END IF
                        ! EAST
                        IF (bcur%NB%E == bnbr%ID) THEN
                            bcur%NP%E = pnbr%ID
                            bcur%NBLOC%E = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%E = -bcur%NB%E
                            END IF
                        END IF
                        ! WEST
                        IF (bcur%NB%W == bnbr%ID) THEN
                            bcur%NP%W = pnbr%ID
                            bcur%NBLOC%W = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%W = -bcur%NB%W
                            END IF
                        END IF
                        ! NORTH EAST
                        IF (bcur%NB%NE == bnbr%ID) THEN
                            bcur%NP%NE = pnbr%ID
                            bcur%NBLOC%NE = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%NE = -bcur%NB%NE
                            END IF
                        END IF
                        ! SOUTH EAST
                        IF (bcur%NB%SE == bnbr%ID) THEN
                            bcur%NP%SE = pnbr%ID
                            bcur%NBLOC%SE = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%SE = -bcur%NB%SE
                            END IF
                        END IF
                        ! SOUTH WEST
                        IF (bcur%NB%SW == bnbr%ID) THEN
                            bcur%NP%SW = pnbr%ID
                            bcur%NBLOC%SW = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%SW = -bcur%NB%SW
                            END IF
                        END IF
                        ! NORTH WEST
                        IF (bcur%NB%NW == bnbr%ID) THEN
                            bcur%NP%NW = pnbr%ID
                            bcur%NBLOC%NW = IBNBR
                            IF (pcur%ID /= pnbr%ID) THEN
                                bcur%NB%NW = -bcur%NB%NW
                            END IF
                        END IF
                    END DO
                END DO
            END DO
        END DO

!         10      FORMAT(10A12)
!         WRITE(*,*)
!         WRITE(*,*) 'Check proc neighbors'
!         WRITE(*,10) 'BLKID', 'NB%N', 'NP%N', 'NBLOC%N'
!         DO IPCUR = 1, NPROCS
!             pcur => procs(IPCUR)
!             WRITE(*,*) 'Proc:', pcur%ID
```

```fortran
!              DO IBCUR = 1, pcur%NBLK
!                  bcur => pcur%blocks(IBCUR)
!                  WRITE(*,*) bcur%ID, bcur%NB%N, bcur%NP%N, bcur%NBLOC%N
!              END DO
!          END DO

    END SUBROUTINE init_neighbor_procs

    SUBROUTINE read_config(b)
        ! Read block connectivity file

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE) :: b(:)
        INTEGER :: I, BLKFILE = 99
        ! READ INFOR FOR BLOCK DIMENSIONS
        INTEGER :: NBLKREAD, IMAXBLKREAD, JMAXBLKREAD

        11 FORMAT(3I5)
        33 FORMAT(A)
        22 FORMAT(33I5)
        44 FORMAT(33A5)

        OPEN (UNIT = BLKFILE , FILE = "blockconfig.dat", form='formatted')
        ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
        READ(BLKFILE,*)
        READ(BLKFILE, 11) NBLK, IMAXBLK, JMAXBLK
        READ(BLKFILE,*)
        DO I = 1, NBLK
            ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
            ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
            ! NORTH EAST SOUTH WEST
            READ(BLKFILE, 22) b(I)%ID, &
                b(I)%IMIN, b(I)%JMIN, &
                b(I)%NB%N, &
                b(I)%NB%NE, &
                b(I)%NB%E, &
                b(I)%NB%SE, &
                b(I)%NB%S, &
                b(I)%NB%SW, &
                b(I)%NB%W, &
                b(I)%NB%NW, &
                b(I)%ORIENT
        END DO
        CLOSE(BLKFILE)
    END SUBROUTINE read_config

    SUBROUTINE init_mesh(b)
        ! Create xprime/yprime non-uniform grid, then rotate by angle 'rot'.
        ! Allocate arrays for node parameters (i.e. temperature, cell area, etc)

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: b(:)
        TYPE(MESHTYPE), POINTER :: m
        INTEGER :: IBLK, I, J

        DO IBLK = 1, NBLK

            m => b(IBLK)%mesh

            ! ALLOCATE MESH INFORMATION
                ! ADD EXTRA INDEX AT BEGINNING AND END FOR GHOST NODES
            ALLOCATE( m%xp(  0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%yp(  0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%x(   0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%y(   0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%T(   0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%Ttmp(0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%dt(  0:IMAXBLK+1,    0:JMAXBLK+1) )
            ALLOCATE( m%V2nd(0:IMAXBLK+1,    0:JMAXBLK+1) )
```

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

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

    SUBROUTINE init_temp(blocks)
        ! Initialize temperature across mesh with dirichlet BCs
        ! or constant temperature BCs for DEBUG=1

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET  :: blocks(:)
        TYPE(BLKTYPE), POINTER :: b
        TYPE(MESHTYPE), POINTER :: m
        TYPE(NBRTYPE), POINTER :: NB
        INTEGER :: IBLK, I, J

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

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

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

        ELSE

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

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

SUBROUTINE set_block_bounds(blocks)
    ! Calculate iteration bounds for each block to avoid overwriting BCs.
    ! Call after reading in mesh data from restart file

    TYPE(BLKTYPE), TARGET :: blocks(:)
    TYPE(BLKTYPE), POINTER :: b
    TYPE(NBRTYPE), POINTER :: NB
    INTEGER :: IBLK, I, J

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

        ! Set iteration bounds of each block to preserve BCs
            ! south and west boundaries:
                ! interior: iminloc, jminloc = 0 (use ghost)
                ! boundary: iminloc, jminloc = 2 (1st index is BC)
            ! north and east boundaries:
                ! interior: imaxloc, jmaxloc = maxblk (use ghost)
                ! boundary: imaxloc, jmaxloc = maxblk-1 (max index is BC)

        ! NORTH
        IF (NB%N > 0) THEN
            ! Interior faces have positive ID neighbors
            b%JMAXLOC = JMAXBLK
        ELSE
            ! At North Boundary
            b%JMAXLOC = JMAXBLK - 1
        END IF

        ! EAST
        IF (NB%E > 0) THEN
            ! Interior
```

```
                b%IMAXLOC = IMAXBLK
            ELSE
                ! At east Boundary
                b%IMAXLOC = IMAXBLK - 1
            END IF

            ! SOUTH
            IF (NB%S > 0) THEN
                ! Interior
                b%JMINLOC = 0
            ELSE
                ! At south Boundary
                b%JMINLOC = 1
                ! boundary for updating temperature (dont update BC)
                b%JMINUPD = 2
            END IF

            ! WEST
            IF (NB%W > 0) THEN
                ! Interior
                b%IMINLOC = 0
            ELSE
                ! At west Boundary
                b%IMINLOC = 1
                b%IMINUPD = 2
            END IF
        END DO
    END SUBROUTINE set_block_bounds

    SUBROUTINE init_linklists(blocks, nbrlists)
        ! Create linked lists governing block boundary communication.
        ! Separate list for each neighbor type so we can avoid logic when
        ! updating ghost nodes.

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: blocks(:)
        ! Neighbor information pointer
        TYPE(NBRTYPE), POINTER :: NB
        ! Linked lists of neighbor communication instructions
        TYPE(NBRLIST) :: nbrlists
        TYPE(NBRLIST) :: nbrl
        INTEGER :: IBLK

        ! INITIALIZE LINKED LISTS (HPC1 REQUIRES THIS)
        NULLIFY(nbrlists%N)
        NULLIFY(nbrlists%S)
        NULLIFY(nbrlists%E)
        NULLIFY(nbrlists%W)
        NULLIFY(nbrlists%NW)
        NULLIFY(nbrlists%NE)
        NULLIFY(nbrlists%SE)
        NULLIFY(nbrlists%SW)

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

            ! NORTH
            ! If block north face is internal, add it to appropriate linked list
            ! for north internal faces.
            IF (NB%N > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%N) ) THEN
                    ! Allocate linked list if it hasnt been accessed yet
                    ALLOCATE(nbrlists%N)
                    ! Pointer linked list that will help iterate through the
                    ! primary list in this loop
                    nbrl%N => nbrlists%N
                ELSE
                    ! linked list already allocated (started).  Allocate next
                    ! link as assign current block to it
```

```
                    ALLOCATE(nbrl%N%next)
                    nbrl%N => nbrl%N%next
                END IF

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

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

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

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

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

            ! SOUTH EAST
            IF (NB%SE > 0) THEN
                IF ( .NOT. ASSOCIATED(nbrlists%SE) ) THEN
                    ALLOCATE(nbrlists%SE)
                    nbrl%SE => nbrlists%SE
                ELSE
```

```fortran
                    ALLOCATE(nbrl%SE%next)
                    nbrl%SE => nbrl%SE%next
                END IF
                nbrl%SE%ID = IBLK
                NULLIFY(nbrl%SE%next)
            END IF

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

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

    SUBROUTINE update_ghosts(b, nbrlists)
        ! Update ghost nodes of each block based on neightbor linked lists.
        ! Ghost nodes contain solution from respective block face/corner
        ! neighbor for use in current block solution.

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

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

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

            ! Otherwise, assign neighbor values to all ghost nodes:

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

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

```

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

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

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

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

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

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

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

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

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

```fortran

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

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

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

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

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


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


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

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

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

```
            END IF

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

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

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

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

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

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

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

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

            ! NORTH WEST CORNER GHOST NODES
            IF ( NB%NW > 0 ) THEN
                Tgh => b( IBLK )%mesh%T
                INBR = b( IBLK )%NB%NW
```

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

    SUBROUTINE calc_cell_params(blocks)
        ! calculate areas for secondary fluxes and constant terms in heat
        ! treansfer eqn. Call after reading mesh data from restart file

        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(MESHTYPE), POINTER :: m
        INTEGER :: IBLK, I, J
        ! Areas used in counter-clockwise trapezoidal integration to get
        ! x and y first-derivatives for center of each cell (Green's thm)
        REAL(KIND=8) :: Ayi_half, Axi_half, Ayj_half, Axj_half

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

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

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

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

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

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

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

    SUBROUTINE calc_constants(blocks)
        ! Calculate terms that are constant regardless of iteration
        !(time step, secondary volumes, constant term.)  This way,
        ! they don't need to be calculated within the loop at each iteration

        TYPE(BLKTYPE), TARGET :: blocks(:)
        TYPE(MESHTYPE), POINTER :: m
        INTEGER :: IBLK, I, J
        DO IBLK = 1, NBLK
            m => blocks(IBLK)%mesh
            DO J = 0, JMAXBLK + 1
                DO I = 0, IMAXBLK + 1
                    ! CALC TIMESTEP FROM CFL
                    m%dt(I,J) = ((CFL * 0.5D0) / alpha) * m%V(I,J) ** 2 &
                                    / ( (m%xp(I+1,J) - m%xp(I,J))**2 &
                                      + (m%yp(I,J+1) - m%yp(I,J))**2 )
                    ! CALC SECONDARY VOLUMES
                    ! (for rectangular mesh, just average volumes of the 4 cells
                    !  surrounding the point)
                    m%V2nd(I,J) = ( m%V(I,  J) + m%V(I-1,  J) &
                                  + m%V(I,J-1) + m%V(I-1,J-1) ) * 0.25D0
                    ! CALC CONSTANT TERM
                    ! (this term remains constant in the equation regardless of
                    !  iteration number, so only calculate once here,
                    !  instead of in loop)
                    m%term(I,J) = m%dt(I,J) * alpha / m%V2nd(I,J)
                END DO
            END DO
        END DO
    END SUBROUTINE calc_constants

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

    SUBROUTINE calc_temp(b)
        ! Calculate temperature at all points in mesh, excluding BC cells.
        ! Calculate first and second derivatives for finite-volume scheme

        TYPE(BLKTYPE), TARGET :: b(:)
        TYPE(MESHTYPE), POINTER :: m
        ! First partial derivatives of temperature in x and y directions
        REAL(KIND=8) :: dTdx, dTdy
        INTEGER :: IBLK, I, J

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

            ! RESET SUMMATION
            m%Ttmp = 0.D0

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

```
                                   ) / m%V(I,J)
                    dTdy = - 0.5d0 &
                                * (( m%T(I+1,J) + m%T(I+1,J+1) ) * m%Axi(I+1,J) &
                                -  ( m%T(I,  J) + m%T(I,  J+1) ) * m%Axi(I,  J) &
                                -  ( m%T(I,J+1) + m%T(I+1,J+1) ) * m%Axj(I,J+1) &
                                +  ( m%T(I,  J) + m%T(I+1,  J) ) * m%Axj(I,  J) &
                                   ) / m%V(I,J)

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

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

END MODULE BLOCKMOD
```

Listing 5: Grids are decomposed into blocks and mapped onto NPROCS processors and information pertaining to neighbors is stored using the GRIDMOD module

Appendix C: Multi-Block Plot3D Reader-Writer

```
! MAE 267
! PROJECT 4
! LOGAN HALSTROM
! 14 NOVEMBER 2015

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

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

MODULE IO
    USE CONSTANTS
    USE BLOCKMOD
    IMPLICIT NONE

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

    ! LINKED LIST OF RESIDUAL HISTORY

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

    CONTAINS

        SUBROUTINE write_config(procs)
        ! Write block connectivity file with neighbor and BC info
        ! for each processor.
        ! Also write PLOT3D restart files for each processor.

        TYPE(PROCTYPE), TARGET :: procs(:)
        TYPE(PROCTYPE), POINTER :: p
        ! BLOCK DATA TYPE
        TYPE(BLKTYPE), POINTER :: b
        INTEGER :: IP, IB, BLKFILE = 99
        CHARACTER(2) :: procname
        CHARACTER(20) :: xfile, qfile

        11 FORMAT(3I5)
        33 FORMAT(A)
        22 FORMAT(33I5)
        44 FORMAT(33A5)

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! WRITE CONFIG FILE FOR EACH PROCESSOR !!!!!!!!!!!!!!!!!!!!!!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        DO IP = 1, NPROCS
            p => procs(IP)

            ! MAKE FILE NAME (i.e. 'p01.config')
            IF (p%ID<10) THEN
                ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
                WRITE(procname, '(A,I1)') '0', p%ID
            ELSE
```

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

            OPEN (UNIT = BLKFILE , FILE = TRIM("p" // procname // ".config"), form='formatted')

            ! WRITE AMOUNT OF BLOCKS AND DIMENSIONS
            WRITE(BLKFILE, 33) 'NBLK' // ' IMAXBLK' // ' JMAXBLK'
            WRITE(BLKFILE, 11) p%NBLK, IMAXBLK, JMAXBLK

            ! HEADER
            WRITE(BLKFILE, 44) 'ID', 'IMIN', 'JMIN', 'SIZE', &
                               'NNB', 'NNP', 'NLOC', &
                               'SNB', 'SNP', 'SLOC', &
                               'ENB', 'ENP', 'ELOC', &
                               'WNB', 'WNP', 'WLOC', &
                               'NENB', 'NENP', 'NEL', &
                               'SENB', 'SENP', 'SEL', &
                               'SWNB', 'SWNP', 'SWL', &
                               'NWNB', 'NWNP', 'NWL', &
                               'ORI'
            DO IB = 1, p%NBLK
                b => p%blocks(IB)
                ! FOR EACH BLOCK, WRITE BLOCK NUMBER, STARTING/ENDING GLOBAL INDICES.
                ! THEN BOUNDARY CONDITION AND NEIGHBOR NUMBER FOR EACH FACE:
                ! NORTH EAST SOUTH WEST
                WRITE(BLKFILE, 22) b%ID, b%IMIN, b%JMIN, INT(b%SIZE), &
                                   b%NB%N,  b%NP%N,  b%NBLOC%N, &
                                   b%NB%S,  b%NP%S,  b%NBLOC%S, &
                                   b%NB%E,  b%NP%E,  b%NBLOC%E, &
                                   b%NB%W,  b%NP%W,  b%NBLOC%W, &
                                   b%NB%NE, b%NP%NE, b%NBLOC%NE, &
                                   b%NB%SE, b%NP%SE, b%NBLOC%SE, &
                                   b%NB%SW, b%NP%SW, b%NBLOC%SW, &
                                   b%NB%NW, b%NP%NW, b%NBLOC%NW, &
                                   b%ORIENT
            END DO
            CLOSE(BLKFILE)
        END DO

        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!! WRITE SOLUTION RESTART FILES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        DO IP = 1, NPROCS
            p => procs(IP)
            ! MAKE FILE NAME
            IF (p%ID<10) THEN
                ! IF SINGLE DIGIT, PAD WITH 0 IN FRONT
                WRITE(procname, '(A,I1)') '0', p%ID
            ELSE
                WRITE(procname, '(I2)') p%ID
            END IF
            xfile = "p" // procname // ".grid"
            qfile = "p" // procname // ".T"
            CALL plot3D(p%blocks, p%NBLK, xfile, qfile)
        END DO
    END SUBROUTINE write_config

    SUBROUTINE plot3D(blocks, NBLKS, xfile, qfile)
        IMPLICIT NONE

        TYPE(BLKTYPE) :: blocks(:)
        INTEGER :: IBLK, I, J, NBLKS
        ! OUTPUT FILES (without file exension)
        CHARACTER(20) :: xfile, qfile

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

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

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

         ! OPEN FILES
         OPEN(UNIT=gridUnit,FILE = TRIM(xfile) // '.form.xyz',FORM='formatted')
         OPEN(UNIT=tempUnit,FILE = TRIM(qfile) // '.form.dat',FORM='formatted')

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


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

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

         ! CLOSE FILES
         CLOSE(gridUnit)
         CLOSE(tempUnit)

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

         ! OPEN FILES
         OPEN(UNIT=gridUnit,FILE = TRIM(xfile) // '.xyz',FORM='unformatted')
         OPEN(UNIT=tempUnit,FILE = TRIM(qfile) // '.dat',FORM='unformatted')

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


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

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

```fortran
        END DO

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

    SUBROUTINE readPlot3D(blocks)
        IMPLICIT NONE

        TYPE(BLKTYPE) :: blocks(:)
        INTEGER :: IBLK, I, J
        ! READ INFO FOR BLOCK DIMENSIONS
        INTEGER :: NBLKREAD, IMAXBLKREAD, JMAXBLKREAD
        ! OUTPUT FILES
        CHARACTER(20) :: xfile, qfile

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

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

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

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


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

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

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



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

```
        TYPE(RESLIST), POINTER :: hist

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

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

        CLOSE(resUnit)
    END SUBROUTINE write_res


END MODULE IO
```

Listing 6: Code for saving formatted multiblock PLOT3D solution files and reading restart files

Appendix D: Other Relevant Codes

```
! MAE 267
! PROJECT 4
! LOGAN HALSTROM
! 14 NOVEMBER 2015


! DESCRIPTION:  Solve heat conduction equation for single block of steel.

! INPUTS: Set grid size, block decomposition, debug in 'config.in'
!         Set number of processors in 'run.sh'

! TO COMPILE:
    ! mpif90 -o main -O3 modules.f90 inout.f90 subroutines.f90 main.f90
        ! makes executable file 'main'
        ! 'rm *.mod' afterward to clean up unneeded compiled files
! TO RUN:
    ! on hpc1 nodes: sbatch run.sh
    ! on hpc1 front end: ./main or ./run.sh



PROGRAM heatTrans
!     USE CLOCK
    USE CONSTANTS
    USE subroutines
    USE IO

    IMPLICIT NONE

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! INITIALIZE VARIABLES !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    ! BLOCKS
    TYPE(BLKTYPE), ALLOCATABLE :: blocks(:)
    ! LINKED LISTS STORING NEIGHBOR INFO
```

```
    TYPE(NBRLIST) :: nbrlists
    ! PROCESSORS
    TYPE(PROCTYPE), ALLOCATABLE :: procs(:)
    ! ITERATION PARAMETERS
    ! Residual history linked list
    TYPE(RESLIST), POINTER :: res_hist
    ! Maximum number of iterations
    INTEGER :: iter = 1, IBLK
    REAL(KIND=8) :: start_total, end_total
    REAL(KIND=8) :: start_solve, end_solve
    ! CLOCK TOTAL TIME OF RUN
    start_total = MPI_Wtime()


    write(*,*) 'starting mpi'

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! START MPI !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    ! INITIALIZE MPI
    CALL MPI_Init(IERROR)
    ! DETERMINE MY PROCESSOR ID
    ! ARGUMENTS: COMM, MYID, IERROR
    CALL MPI_Comm_rank(MPI_COMM_WORLD, MYID, IERROR)
    ! FIND OUT HOW MANY PROCESSORS ARE USED
    ! ARGUMENTS: COMM, NPROCS, IERROR
    CALL MPI_Comm_size(MPI_COMM_WORLD, NPROCS, IERROR)

!     ! SET NPROCS MANUALLY NOW FOR DEBUG!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!     NPROCS = 4

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! INITIALIZE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    ! have the first processor only set up problem
    IF(MYID == 0) THEN

        write(*,*) 'initializing'



        ! READ INPUTS FROM FILE
        CALL read_input()
        ALLOCATE( blocks(NBLK) )
        ALLOCATE( procs(NPROCS) )
        ! INIITIALIZE GRID SYSTEM
        WRITE(*,*) 'Making mesh...'
        CALL init_gridsystem(blocks, procs)

        !TURN THIS ON FOR PJ5!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        ! CLEAN UP INITIALIZATION
        DEALLOCATE(blocks, procs)
    END IF



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

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

    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !!! SAVE RESULTS !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    WRITE(*,*) 'Writing results...'

    !TURN THIS ON FOR PJ5!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!     ! SAVE SOLUTION AS PLOT3D FILES
!     CALL plot3D(blocks)
    ! CALC TOTAL WALL TIME
    end_total = MPI_Wtime()
    wall_time_total = end_total - start_total

    !TURN THIS ON FOR PJ5!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

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

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

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

    WRITE(*,*) 'Done!'

    CALL MPI_Finalize(ierror)
END PROGRAM heatTrans
```

Listing 7: Wrapper program