HPSC Lab 10

Luna McBride, Noki Cheng

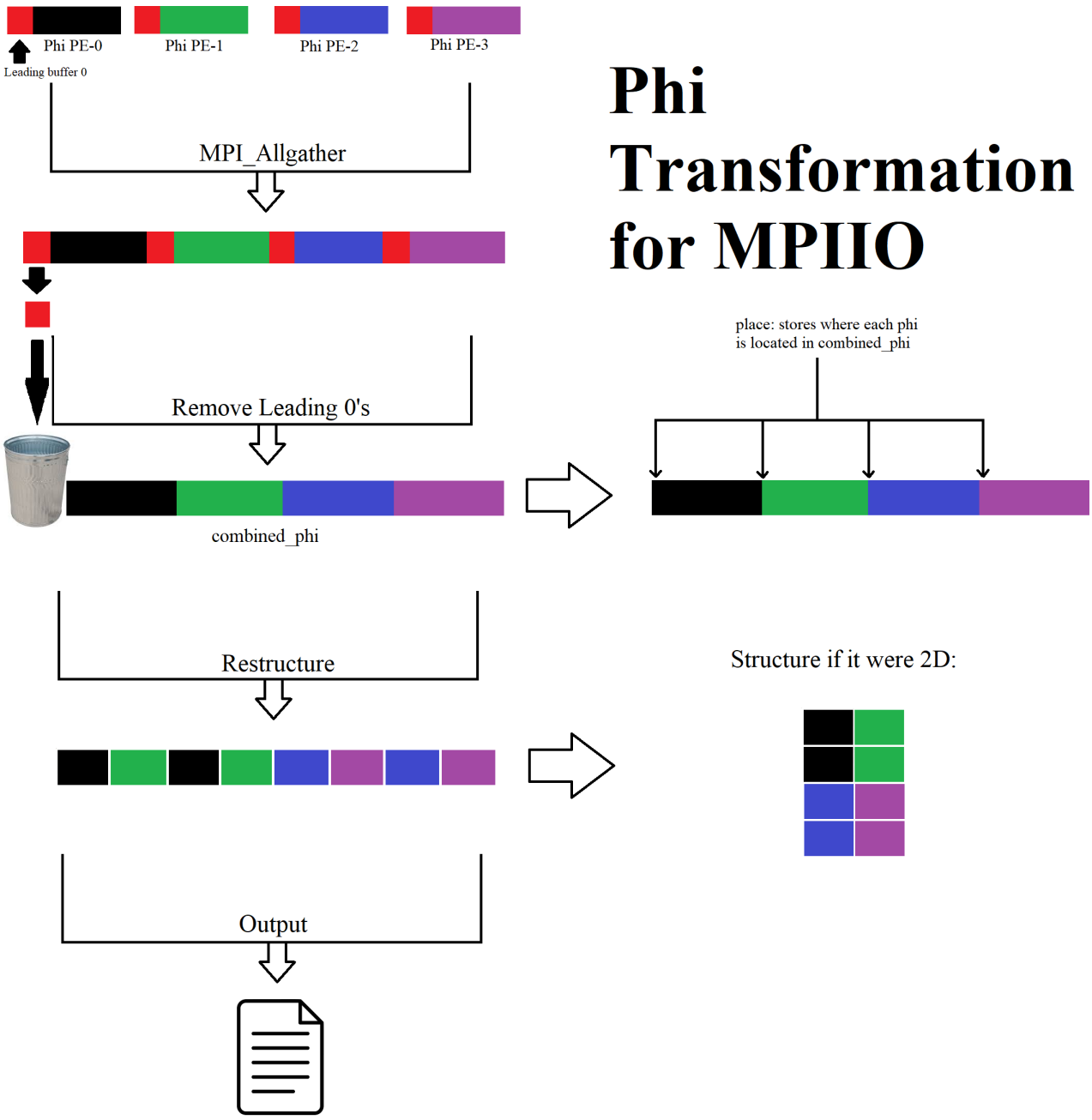
November 17 2023

1 Discussion of Approach

The purpose of this lab is to utilize MPIIO to print the mesh into a single file rather than the many files shown off in other labs. This can be done by utilizing the variable phi, which represents the results of the mesh calculations for each specified PE. Phi, however, is a single dimensional vector that is spread out between each of the PEs. PEsum has already been run to calculate values on shared nodes luckily enough, so the approach comes down to connecting the different phis on the different PEs and sending them to the shared output file.

This process starts with gathering the phis into a combined phi by using MPI_Allgather and clipping off the buffer term they all have at the beginning. Phi will be passed to the file as a single dimensional array, so phi must be collected onto a single PE and restructured in a way that the final array outputted resembles the shape of the overall mesh when plotted. The sb python code already handles writing the array into the two-dimensional mesh structure, so this code just needs to weave the phi array so that it can resemble this structure after being sent to sb. This is accomplished using an array called place, while takes note of where each PE's phi starts in the combined phi.

Several for loops are then used to create multiple lines sized by the x axis, with each line connecting as if they were still the contiguous x-PEs shown in a normal mesh. These lines continue until the y-PE boundaries are hit, which then updates place to look at the next set of PEs in the y direction. This results in an array that looks like the mesh, just flipped around. The values of phi in this case go from zero to one from left to right, so the collective phi array needs to be rotated to fit the zero on the bottom and one on the top structure that has existed in the plots from previous labs. This can be done by reversing then transposing the array, utilizing the transpose method discussed here: https://stackoverflow.com/questions/16737298/what-is-the-fastest-way-to-transpose-a-matrix-in-c . The transpose function alone resulted with the zeros on top rather than on bottom, so reversing the array first created the expected ones on the top result. The implementation of this methodology was then tested on all possible meshes made by 4 cores (1x1, 1x2, 1x3, 1x4, 4x1, 3x1, 2x1, 2x2) in order to ensure that the algorithm scaled past two PEs in each direction, thus ensuring scalability. The combined phi was then dumped using MPIIO into the required output file, as shown in the below figure.



2 Results

In this section, we will show the n-to-1 numerical outputs of binary files produced through our use of MPIIO. To avoid redundancy and confusion, some of the configs are fixed

- Solve: jacobi;
- tEnd: 0.15;

- dt: 0.05
- tPlot: 0.05.

We will show three different test cases with different PE and cell sizes, including a special test case 0.

It is also important to note before this, however, that our code uses MPI_Type_contiguous instead of MPI_Type_create_subarray. This does not allow for the Fortran array style to be used, so the sb code needs to have nPEy passed in instead of nPEx due to the array being rotated compared to what the program expects.

2.1 Test Case 0: Special Test Case

```
1
2  ---------------------------------------------------
3
4   S O L V E R S
5   D E M O   C O D E
6
7   Running on 4 processors
8
9  ---------------------------------------------------
10
11
12 Input Summary:
13 ---------------------------
14 No. PE   in  x-direction: 2
15              y-direction: 2
16 No. Cells in x-direction: 3
17              y-direction: 3
18 Linear solver           : jacobi
19 End Time                : 0.15
20 Time Step               : 0.05
21 Plot Interval           : 0.05
22 This is a restart (1/0) : 0
23
24 Plotting at time = 0 Plot ID = 0
25 ************* Writing dump file *****************
26 ************* Writing dump file *****************
27 ************* Writing dump file *****************
28 ************* Writing dump file *****************
29 Plotting at time = 0.05 Plot ID = 1
30 ************* Writing dump file *****************
31 ************* Writing dump file *****************
32 ************* Writing dump file *****************
33 ************* Writing dump file *****************
34 Plotting at time = 0.1 Plot ID = 2
35 ************* Writing dump file *****************
36 ************* Writing dump file *****************
37 ************* Writing dump file *****************
38 ************* Writing dump file *****************
39 Execution Completed Successfully
40
```

```
-----------------------------------
   sb.py
-----------------------------------


Read user specified file: phi_dump.bin
n points x-dir._total   :  7

Matrix Read from Binary File:
--------------------------------------------------

  0.0 10.0 20.0  0.1 10.1 20.1 30.1
  1.0 11.0 21.0  1.1 11.1 21.1 31.1
  2.0 12.0 22.0  2.1 12.1 22.1 32.1
  0.2 10.2 20.2  0.3 10.3 20.3 30.3
  1.2 11.2 21.2  1.3 11.3 21.3 31.3
  2.2 12.2 22.2  2.3 12.3 22.3 32.3
  3.2 13.2 23.2  3.3 13.3 23.3 33.3
```

2.2 Test Case 1: nPEx=4, nPEy=1, nCellx=nCelly=5

```
[trmc7708@c3cpu-a2-u15-3 src]$ ./sb_py3.py -f phi_dump.bin -c 5 -n 1

-----------------------------------
   sb.py
-----------------------------------


Read user specified file: phi_dump.bin
n points x-dir._total   :  6

Matrix Read from Binary File:
--------------------------------------------------

1.000 1.000 1.000 1.000 1.000 1.000
0.950 0.946 0.943 0.943 0.946 0.950
0.900 0.892 0.887 0.887 0.892 0.900
0.850 0.838 0.831 0.831 0.838 0.850
0.800 0.785 0.775 0.775 0.785 0.800
0.750 0.732 0.720 0.720 0.732 0.750
0.700 0.679 0.666 0.666 0.679 0.700
0.650 0.627 0.613 0.613 0.627 0.650
0.600 0.575 0.560 0.560 0.575 0.600
0.550 0.525 0.509 0.509 0.525 0.550
0.500 0.474 0.458 0.458 0.474 0.500
0.450 0.425 0.409 0.409 0.425 0.450
0.400 0.376 0.361 0.361 0.376 0.400
0.350 0.327 0.313 0.313 0.327 0.350
0.300 0.279 0.267 0.267 0.279 0.300
0.250 0.232 0.221 0.221 0.232 0.250
0.200 0.185 0.176 0.176 0.185 0.200
0.150 0.138 0.131 0.131 0.138 0.150
0.100 0.092 0.087 0.087 0.092 0.100
0.050 0.046 0.044 0.044 0.046 0.050
0.000 0.000 0.000 0.000 0.000 0.000
```

2.3 Test Case 2: nPEx=2, nPEy=2, nCellx=nCelly=3

```
[trmc7708@c3cpu-a2-u15-3 src]$ ./sb_py3.py -f phi_dump.bin -c 3 -n 2

-----------------------------------
   sb.py
-----------------------------------


Read user specified file: phi_dump.bin
n points x-dir._total   :  7

Matrix Read from Binary File:
--------------------------------------------------

1.000 1.000 1.000 1.000 1.000 1.000 1.000
0.833 0.822 0.814 0.811 0.814 0.822 0.833
0.667 0.647 0.633 0.628 0.633 0.647 0.667
0.500 0.478 0.462 0.456 0.462 0.478 0.500
0.333 0.314 0.301 0.296 0.301 0.314 0.333
0.167 0.156 0.148 0.145 0.148 0.156 0.167
0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

3 Self Evaluation

3.1 Luna

This assignment was split in two by who updated phi and who got the output to run. I updated phi, and may have gone a little far with the concept. The updates became a game of questions like "but what if the mesh is expanded?" and "what if the transpose utility does not work on a mesh that is not square?". Allgather also only worked correctly if phi was an array and not a vector, so add a conversion array to go with the flipping array needed for transpose and the array to hold the data once the phis were weaved together. There was very likely another way to update phi that reduced space and time complexity, but I did not want to get in the same loop of "try something that would work great in Python with NumPy/Pandas but end up empty handed in C++" like happened in the previous lab. The code works and accounts for meshes larger than 2x2, so this is plenty.

3.2 Noki

The lab overall went well, and we successfully implemented the n-to-1 parallel output using MPIIO. The process of adapting the MPIIO code to handle a 1-D array was challenging but educational. Understanding and addressing the overlapping nodes at PE boundaries required careful consideration, which is mostly contributed by Luna.

- **Hard Parts:**
  - Adapting MPIIO for 1-D output: Modifying the MPIIO code to work with a 1-D array instead of a 2-D matrix posed a significant challenge. Ensuring correct indexing and avoiding duplication of PE boundary values required careful debugging.

  - Addressing PE Boundary Overlaps: The transientDiffusion_1mat code had overlapping nodes at PE boundaries, necessitating thoughtful modifications to the MPIIO approach. This required a deep understanding of both the finite difference code and MPIIO.

- **What Went Well:**
  - Successful Parallel Output: The implemented solution demonstrated the desired n-to-1 parallel output capability on a 2x2 and 4x1 PE decomposition without any evident artifacts of the number of processes.
  - Clear Test Case Verification: The test case, showcasing the i-j location and PE residence of phi values, verified the correctness of the n-to-1 writer. The results from sb_py3.py aligned with expectations.

In conclusion, while the lab presented challenges, the successful implementation of the parallel output functionality and the clear verification through test cases indicate a satisfactory completion of the task.

4 Appendix A: C++ Source Code

```
//
    ================================================================================

//  ||
    ||
//  ||              transientDiffusion
    ||
//  ||              ------------------------------------------------------
    ||
//  ||              T R A N S I E N T D I F F U S I O N
    ||
//  ||
    ||
//  ||              D E M O N S T R A T I O N   C O D E
    ||
//  ||              ------------------------------------------------------
    ||
//  ||
    ||
//  ||       Developed by: Scott R. Runnels, Ph.D.
    ||
//  ||                     University of Colorado Boulder
    ||
//  ||
    ||
//  ||                For: CU Boulder CSCI 4576/5576 and associated labs
    ||
//  ||
    ||
//  ||           Copyright 2020 Scott Runnels
    ||
```

```
16 //  ||
       ||
17 //  ||                    Not for distribution or use outside of the
       ||
18 //  ||                    this course.
       ||
19 //  ||
       ||
20 //
       ================================================================================


21
22 #include<mpi.h>
23 #include <cstdlib>
24
25 //
       ================================================================================

26 //  ||
       ||
27 //  ||  M P I I O   D u m p   W r i t e r
       ||
28 //  ||
       ||
29 //
       ================================================================================


30
31 void write_mpiio_dump(mpiInfo &myMPI)
32 {
33
34   printf("************* Writing dump file *****************\n");
35
36   int nRealx  = myMPI.nRealx;    // Number of real points in the x- and y-
       directions
37   int nRealy  = myMPI.nRealy;    //
38   int myPE, numPE;    // This process' ID and the number of processes total
39   int nPEx,  nPEy;    // Number of processes in the x- and y-directions, in the 2D
       grid of processes
40   int iPE ,   jPE;    // This process' logical ID in the 2D array of processors
41
42   nPEx = myMPI.nPEx;
43   nPEy = myMPI.nPEy;
44   myPE = myMPI.myPE;
45   numPE = myMPI.numPE;
46   jPE = int(myPE/nPEx);
47   iPE = myPE - jPE*nPEx;
48
49   // float *phiVal;
50   // phiVal = new float [nField+1];
51   // for (int i = 1; i <= nField; i++)
52   //     phiVal[i] = i + myPE/10.;
53
54   MPI_Barrier(MPI_COMM_WORLD);
55   int num_send = phi.size();
```

```
56  int max_send;
57
58  MPI_Allreduce(&num_send, &max_send, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
59
60  int gather_size = max_send*myMPI.numPE;
61  double* phi_array = new double[phi.size()];
62  int* pe_gather = new int[phi.size()];
63
64  for(int i = 0; i < phi.size(); ++i){
65    phi_array[i] = phi[i];
66    pe_gather[i] = myMPI.myPE;
67  }
68
69  double* phiTest = new double[gather_size];
70  int* phiPE = new int[gather_size];
71
72    for ( int i = 0 ; i < gather_size ; ++i ) {phiTest[i] = 0.; phiPE[i] = -1;}
73
74  MPI_Allgather( pe_gather , max_send , MPI_INT    , phiPE , max_send , MPI_INT    ,
        MPI_COMM_WORLD);
75  MPI_Allgather( phi_array , max_send , MPI_DOUBLE, phiTest , max_send ,
      MPI_DOUBLE,  MPI_COMM_WORLD);
76
77  MPI_Barrier(MPI_COMM_WORLD);
78  //if(myMPI.myPE==0){
79  double* combined_phi = new double[gather_size - myMPI.numPE];
80  int prev_pe = -1;
81  int phiCount = 0;
82
83  //Removes the padding 0 from the front and puts all actual values into a
      combined phi variable
84  for(int i = 0; i < gather_size; ++i){
85    if(prev_pe != phiPE[i]) prev_pe = phiPE[i];
86    else {
87      combined_phi[phiCount] = phiTest[i];
88      phiCount++;
89    }
90  }
91
92  int* place = new int[myMPI.numPE];
93  for (int i = 0; i < myMPI.numPE; ++i){
94    place[i] = 0 + (phi.size()-1)*i;
95  }
96
97  int big_mesh_x = myMPI.nRealx + nCellx*(myMPI.nPEx-1);
98  int big_mesh_y = myMPI.nRealy + nCelly*(myMPI.nPEy-1);
99
100 int current_pe = 0;
101 int pe_change = 0;
102 int current_position = 0;
103 int num_changedy = 0;
104 int num_changedx = 0;
105
106 double* fixed_phi = new double[big_mesh_x*big_mesh_y];
107 double* flipped_phi = new double[big_mesh_x*big_mesh_y];
```

```
  int i = 0;

  for (int y = 0; y < big_mesh_y; ++y){
    if(y%nCelly==0 && y!=0 && (num_changedy+1 < myMPI.nPEy)){
      pe_change += myMPI.nPEx;
      num_changedy++;
    }
    current_pe = current_pe + pe_change;

    for(int x = 0; x < big_mesh_x; x++){
      if(x%nCellx==0 && x!=0 && (num_changedx+1 < myMPI.nPEx)){
   place[current_pe]++;
   current_pe++;
   num_changedx++;
      }

      current_position = place[current_pe];
      fixed_phi[i] = combined_phi[current_position];
      i++;
      place[current_pe]++;
    }
    current_pe = 0;
    num_changedx = 0;
  }
  int c = 0;
  //Reverse the array so that it can transpose to the correct form
  double temp = 0.;
  int mesh_len = (big_mesh_x*big_mesh_y);
  for(int n = 0; n < mesh_len/2; n++) {
    temp = fixed_phi[mesh_len-n-1];
        fixed_phi[mesh_len-n-1] = fixed_phi[n];
        fixed_phi[n] = temp;
    }

  //Transpose the phi so that it can look correct
  //Source for transpose: https://stackoverflow.com/questions/16737298/what-is-the
    -fastest-way-to-transpose-a-matrix-in-c
  for(int n = 0; n < big_mesh_x*big_mesh_y; n++) {
        int i = n/big_mesh_y;
        int j = n%big_mesh_y;
        flipped_phi[n] = fixed_phi[big_mesh_x*j + i];
    }

  //print out phi
<<<<<<< HEAD
  //if (myMPI.myPE==0){
    //cout << "[" << endl;
  //for(int n = 0; n < big_mesh_x*big_mesh_y; n++) {
    //if(n%big_mesh_y==0) cout << endl;
    //cout << flipped_phi[n] << "  ";
    //}
  ///cout << "]" << endl;
  //

  //if(myMPI.myPE == 0){
```

```
MPI_Datatype myRealNodes;

MPI_Type_contiguous(nTotal, MPI_DOUBLE, &myRealNodes);

MPI_Type_commit(&myRealNodes);
//
//
  ------------------------------------------------------------------------------------------------------------


MPI_File_delete( "phi_dump.bin", MPI_INFO_NULL );

MPI_File fh;

MPI_File_open(MPI_COMM_WORLD, "phi_dump.bin", MPI_MODE_CREATE | MPI_MODE_WRONLY,
   MPI_INFO_NULL, &fh);

MPI_File_set_view (fh, 0, MPI_DOUBLE, myRealNodes, "native", MPI_INFO_NULL);
//                          |                                       |
//                          |                                       |
//                      The data type of each element     Could contain
  optimization hints,
//                                                        e.g., striping
  information
//
  ------------------------------------------------------------------------------------------------------------


//  (8) Perform the collective write operation


//
  ------------------------------------------------------------------------------------------------------------

//                          The sub_array describing this
//        File pointer      PE's real-nodes part of A
//                  |                   |
//                  |                   |
if(myMPI.myPE==0){
MPI_File_write(fh, flipped_phi, 1, myRealNodes, MPI_STATUS_IGNORE);
}
//                      |
//                      |
//                  Memory location of original
//                  data set for which the
//                  sub arrays were created.
//
  ------------------------------------------------------------------------------------------------------------


MPI_File_close(&fh);
MPI_Type_free(&myRealNodes);
//}
free(pe_gather);
free(phiPE);
```

```
  free(phi_array);
  free(phiTest);
  free(combined_phi);
  free(fixed_phi);
  free(flipped_phi);

  //}
  MPI_Barrier(MPI_COMM_WORLD);
}
```

5 Appendix B: sb_py3.py Script

```
#!/usr/bin/python3

import os

import sys
import getopt
import glob
import struct


# ==
# ||
# || Main Program
# ||
# ==

def sb(argv):

#    os.system('clear')

    print()
    print( '-------------------------------------')
    print( '   sb.py                             ')
    print( '-------------------------------------')
    print()

    # -
    # |
    # | Get user arguments
    # |
    # -

    readFile = 'NULL'
    nCellx   = 0
    nPEx     = 0

    try:
        opts, args = getopt.getopt(argv,"h f: c: n:")

    except:
       print( 'Error in command-line arguments.')
       exit(0)

```

```
44      for opt, arg in opts:
45
46          if opt == "-h":
47              print( './sb_py3.py -f <binary file to be read> -c <no. cells in x/y
    dir *per PE*> -n <nPEx>')
48              exit(0)
49
50          if opt == "-c":
51              nCellx = int(arg)
52
53          if opt == "-n":
54              nPEx  = int(arg)
55
56          elif opt == "-f":
57              readFile = arg
58
59      # -
60      # |
61      # | Check user arguments
62      # |
63      # -
64
65      if readFile == 'NULL':
66          print("Fatal Error: Specify binary file with -f")
67          exit(0)
68
69      if nCellx == 0:
70          print("Fatal Error: Specify no. cells in x/y direction with -c")
71          exit(0)
72
73      if nPEx == 0:
74          print("Fatal Error: Specify no. PE in x direction with -n")
75          exit(0)
76
77      # -
78      # |
79      # | Specifications for reading the binary and writing to tty
80      # |
81      # -
82
83      nx_total      = nPEx*nCellx + 1
84      DecimalSize   = 8        # 4 for single precision (float) 8 for double
    precisioni (double)
85      DecimalFormat = 'd'    # 'f' for single, 'd' for double
86
87      # -
88      # |
89      # | Read binary file and print it to screen
90      # |
91      # -
92
93      print()
94      print( 'Read user specified file: ' + readFile)
95      print( 'n points x-dir._total   : ' , nx_total)
96
```

```
    file = open(readFile, "rb")

    print()
    print( "Matrix Read from Binary File:")
    print( "-----------------------------------------------------")
    print()

    # Read first data

    Num  = file.read(DecimalSize)

    count_x_total = 0

    # Continue reading data until end of file

    while Num:

        # Read and print data with no line feed

        floatValue = struct.unpack(DecimalFormat,Num)[0]    # 'f' for single, 'd' for double
        print( "{:_.3f}".format(floatValue),end=' ')

        # Put line feed at the end of the mesh

        count_x_total += 1
        if count_x_total == nx_total:
            print()
            count_x_total = 0

        Num  = file.read(DecimalSize)


    file.close()




    print()
    print()



if __name__ == "__main__":
    sb(sys.argv[1:])
```