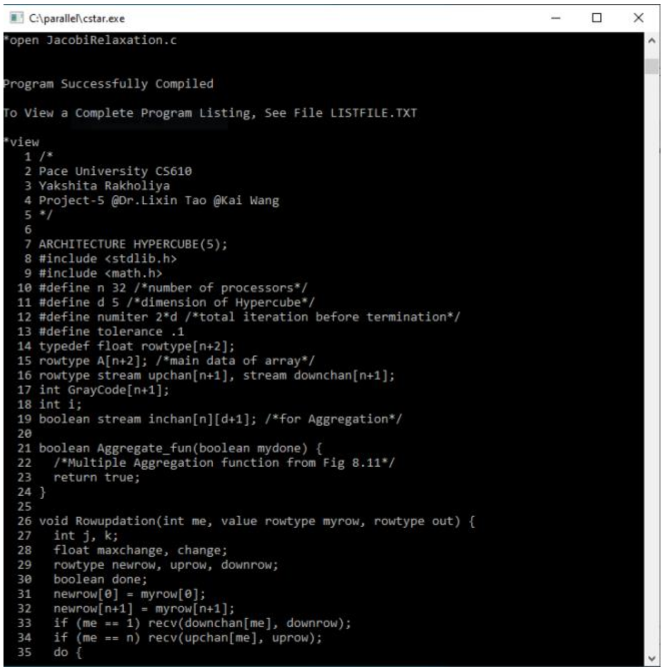
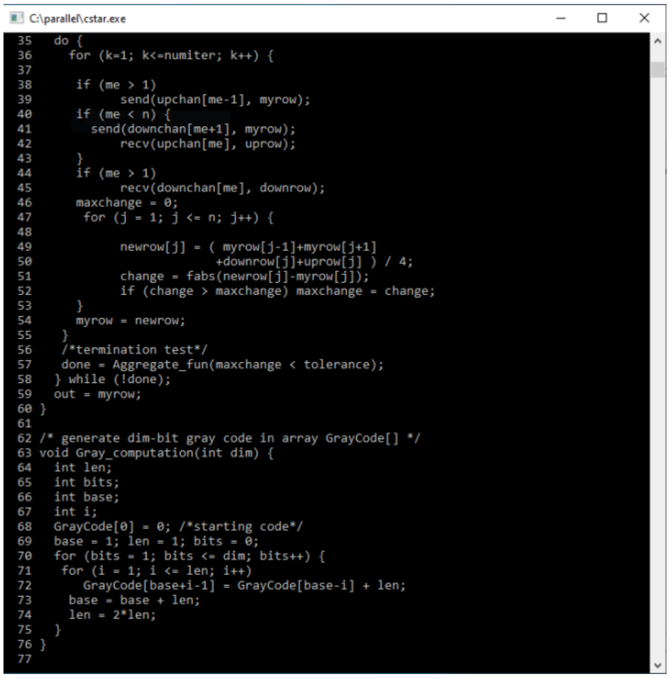
Yakshita B Rakholiya
yr92282n@pace.edu
Student ID: U01875270
Course: CS-610-22756
Project-5

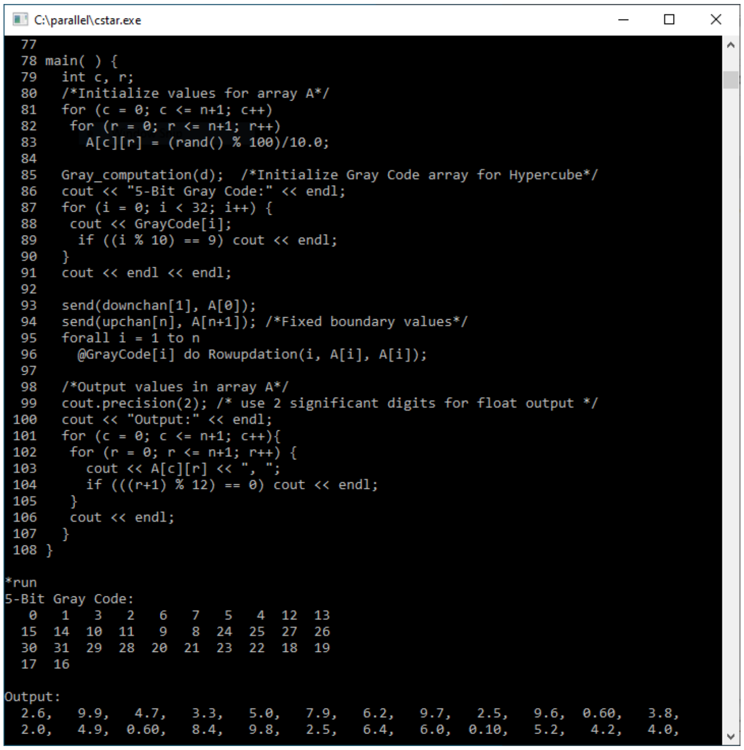
Developing an efficient parallel Jacobi relaxation program on a multiprocessor with convergence test and efficient barrier and aggregation functions.



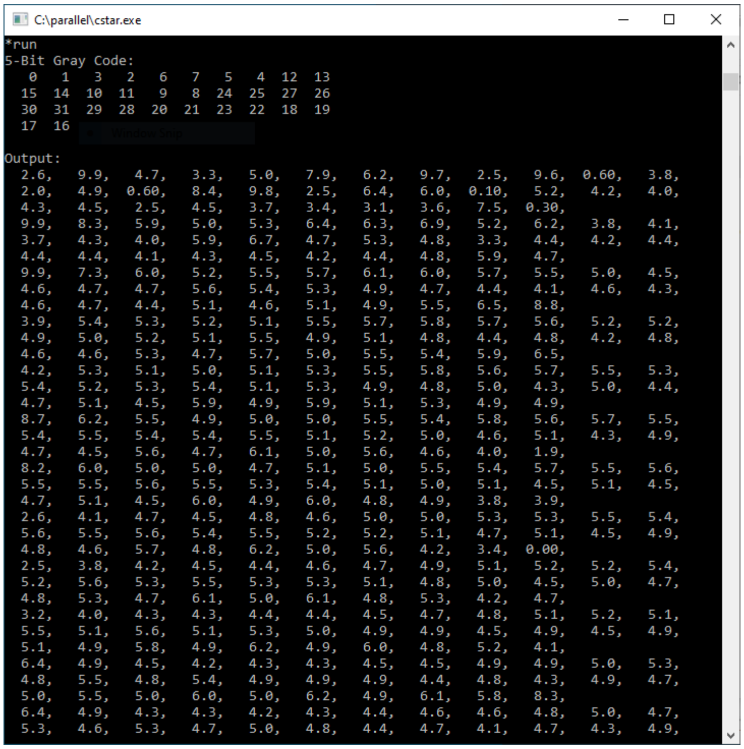
```
*open JacobiRelaxation.c


Program Successfully Compiled

To View a Complete Program Listing, See File LISTFILE.TXT

*view
/*
Pace University CS610
Yakshita Rakholiya
Project-5 @Dr.Lixin Tao @Kai Wang
*/

ARCHITECTURE HYPERCUBE(5);
#include <stdlib.h>
#include <math.h>
#define n 32 /*number of processors*/
#define d 5 /*dimension of Hypercube*/
#define numiter 2*d /*total iteration before termination*/
#define tolerance .1
typedef float rowtype[n+2];
rowtype A[n+2]; /*main data of array*/
rowtype stream upchan[n+1], stream downchan[n+1];
int GrayCode[n+1];
int i;
boolean stream inchan[n][d+1]; /*for Aggregation*/

boolean Aggregate_fun(boolean mydone) {
  /*Multiple Aggregation function from Fig 8.11*/
  return true;
}

void Rowupdation(int me, value rowtype myrow, rowtype out) {
  int j, k;
  float maxchange, change;
  rowtype newrow, uprow, downrow;
  boolean done;
  newrow[0] = myrow[0];
  newrow[n+1] = myrow[n+1];
  if (me == 1) recv(downchan[me], downrow);
  if (me == n) recv(upchan[me], uprow);
  do {
```



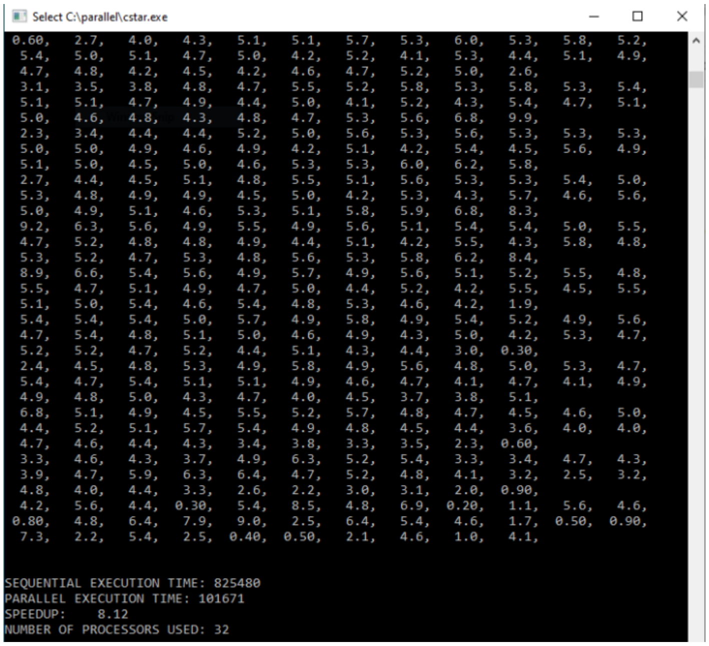
```
  do {
    for (k=1; k<=numiter; k++) {

     if (me > 1)
           send(upchan[me-1], myrow);
     if (me < n) {
       send(downchan[me+1], myrow);
           recv(upchan[me], uprow);
     }
     if (me > 1)
           recv(downchan[me], downrow);
     maxchange = 0;
      for (j = 1; j <= n; j++) {

           newrow[j] = ( myrow[j-1]+myrow[j+1]
                        +downrow[j]+uprow[j] ) / 4;
           change = fabs(newrow[j]-myrow[j]);
           if (change > maxchange) maxchange = change;
     }
     myrow = newrow;
   }
   /*termination test*/
   done = Aggregate_fun(maxchange < tolerance);
  } while (!done);
  out = myrow;
}

/* generate dim-bit gray code in array GrayCode[] */
void Gray_computation(int dim) {
  int len;
  int bits;
  int base;
  int i;
  GrayCode[0] = 0; /*starting code*/
  base = 1; len = 1; bits = 0;
  for (bits = 1; bits <= dim; bits++) {
   for (i = 1; i <= len; i++)
      GrayCode[base+i-1] = GrayCode[base-i] + len;
    base = base + len;
    len = 2*len;
  }
}

```



```

main( ) {
  int c, r;
  /*Initialize values for array A*/
  for (c = 0; c <= n+1; c++)
   for (r = 0; r <= n+1; r++)
     A[c][r] = (rand() % 100)/10.0;

  Gray_computation(d);  /*Initialize Gray Code array for Hypercube*/
  cout << "5-Bit Gray Code:" << endl;
  for (i = 0; i < 32; i++) {
   cout << GrayCode[i];
    if ((i % 10) == 9) cout << endl;
  }
  cout << endl << endl;

  send(downchan[1], A[0]);
  send(upchan[n], A[n+1]); /*Fixed boundary values*/
  forall i = 1 to n
    @GrayCode[i] do Rowupdation(i, A[i], A[i]);

  /*Output values in array A*/
  cout.precision(2); /* use 2 significant digits for float output */
  cout << "Output:" << endl;
  for (c = 0; c <= n+1; c++){
   for (r = 0; r <= n+1; r++) {
     cout << A[c][r] << ", ";
     if (((r+1) % 12) == 0) cout << endl;
   }
   cout << endl;
  }
}

*run
5-Bit Gray Code:
   0   1   3   2   6   7   5   4  12  13
  15  14  10  11   9   8  24  25  27  26
  30  31  29  28  20  21  23  22  18  19
  17  16

Output:
  2.6,   9.9,   4.7,   3.3,   5.0,   7.9,   6.2,   9.7,   2.5,   9.6,  0.60,   3.8,
  2.0,   4.9,  0.60,   8.4,   9.8,   2.5,   6.4,   6.0,  0.10,   5.2,   4.2,   4.0,
```



```
*run
5-Bit Gray Code:
   0   1   3   2   6   7   5   4  12  13
  15  14  10  11   9   8  24  25  27  26
  30  31  29  28  20  21  23  22  18  19
  17  16

Output:
  2.6,   9.9,   4.7,   3.3,   5.0,   7.9,   6.2,   9.7,   2.5,   9.6,  0.60,   3.8,
  2.0,   4.9,  0.60,   8.4,   9.8,   2.5,   6.4,   6.0,  0.10,   5.2,   4.2,   4.0,
  4.3,   4.5,   2.5,   4.5,   3.7,   3.4,   3.1,   3.6,   7.5,  0.30,
  9.9,   8.3,   5.9,   5.0,   5.3,   6.4,   6.3,   6.9,   5.2,   6.2,   3.8,   4.1,
  3.7,   4.3,   4.0,   5.9,   6.7,   4.7,   5.3,   4.8,   3.3,   4.4,   4.2,   4.4,
  4.4,   4.4,   4.1,   4.3,   4.5,   4.2,   4.4,   4.8,   5.9,   4.7,
  9.9,   7.3,   6.0,   5.2,   5.5,   5.7,   6.1,   6.0,   5.7,   5.5,   5.0,   4.5,
  4.6,   4.7,   4.7,   5.6,   5.4,   5.3,   4.9,   4.7,   4.4,   4.1,   4.6,   4.3,
  4.6,   4.7,   4.4,   5.1,   4.6,   5.1,   4.9,   5.5,   6.5,   8.8,
  3.9,   5.4,   5.3,   5.2,   5.1,   5.5,   5.7,   5.8,   5.7,   5.6,   5.2,   5.2,
  4.9,   5.0,   5.2,   5.1,   5.5,   4.9,   5.1,   4.8,   4.4,   4.8,   4.2,   4.8,
  4.6,   4.6,   5.3,   4.7,   5.7,   5.0,   5.5,   5.4,   5.9,   6.5,
  4.2,   5.3,   5.1,   5.0,   5.1,   5.3,   5.5,   5.8,   5.6,   5.7,   5.5,   5.3,
  5.4,   5.2,   5.3,   5.4,   5.1,   5.3,   4.9,   4.8,   5.0,   4.3,   5.0,   4.4,
  4.7,   5.1,   4.5,   5.9,   4.9,   5.9,   5.1,   5.3,   4.9,   4.9,
  8.7,   6.2,   5.5,   4.9,   5.0,   5.0,   5.5,   5.4,   5.8,   5.6,   5.7,   5.5,
  5.4,   5.5,   5.4,   5.4,   5.5,   5.1,   5.2,   5.0,   4.6,   5.1,   4.3,   4.9,
  4.7,   4.5,   5.6,   4.7,   6.1,   5.0,   5.6,   4.6,   4.0,   1.9,
  8.2,   6.0,   5.0,   5.0,   4.7,   5.1,   5.0,   5.5,   5.4,   5.7,   5.5,   5.6,
  5.5,   5.5,   5.6,   5.5,   5.3,   5.4,   5.1,   5.0,   5.1,   4.5,   5.1,   4.5,
  4.7,   5.1,   4.5,   6.0,   4.9,   6.0,   4.8,   4.9,   3.8,   3.9,
  2.6,   4.1,   4.7,   4.5,   4.8,   4.6,   5.0,   5.0,   5.3,   5.3,   5.5,   5.4,
  5.6,   5.5,   5.6,   5.4,   5.5,   5.2,   5.2,   5.1,   4.7,   5.1,   4.5,   4.9,
  4.8,   4.6,   5.7,   4.8,   6.2,   5.0,   5.6,   4.2,   3.4,  0.00,
  2.5,   3.8,   4.2,   4.5,   4.4,   4.6,   4.7,   4.9,   5.1,   5.2,   5.2,   5.4,
  5.2,   5.6,   5.3,   5.5,   5.3,   5.3,   5.1,   4.8,   5.0,   4.5,   5.0,   4.7,
  4.8,   5.3,   4.7,   6.1,   5.0,   6.1,   4.8,   5.3,   4.2,   4.7,
  3.2,   4.0,   4.3,   4.3,   4.4,   4.4,   4.5,   4.7,   4.8,   5.1,   5.2,   5.1,
  5.5,   5.1,   5.6,   5.1,   5.3,   5.0,   4.9,   4.9,   4.5,   4.9,   4.5,   4.9,
  5.1,   4.9,   5.8,   4.9,   6.2,   4.9,   6.0,   4.8,   5.2,   4.1,
  6.4,   4.9,   4.5,   4.2,   4.3,   4.3,   4.5,   4.5,   4.9,   4.9,   5.0,   5.3,
  4.8,   5.5,   4.8,   5.4,   4.9,   4.9,   4.9,   4.4,   4.8,   4.3,   4.9,   4.7,
  5.0,   5.5,   5.0,   6.0,   5.0,   6.2,   4.9,   6.1,   5.8,   8.3,
  6.4,   4.9,   4.3,   4.3,   4.2,   4.3,   4.4,   4.6,   4.6,   4.8,   5.0,   4.7,
  5.3,   4.6,   5.3,   4.7,   5.0,   4.8,   4.4,   4.7,   4.1,   4.7,   4.3,   4.9,
```



```
0.60,   2.7,   4.0,   4.3,   5.1,   5.1,   5.7,   5.3,   6.0,   5.3,   5.8,   5.2,
 5.4,   5.0,   5.1,   4.7,   5.0,   4.2,   5.2,   4.1,   5.3,   4.4,   5.1,   4.9,
 4.7,   4.8,   4.2,   4.5,   4.2,   4.6,   4.7,   5.2,   5.0,   2.6,
 3.1,   3.5,   3.8,   4.8,   4.7,   5.5,   5.2,   5.8,   5.3,   5.8,   5.3,   5.4,
 5.1,   5.1,   4.7,   4.9,   4.4,   5.0,   4.1,   5.2,   4.3,   5.4,   4.7,   5.1,
 5.0,   4.6,   4.8,   4.3,   4.8,   4.7,   5.3,   5.6,   6.8,   9.9,
 2.3,   3.4,   4.4,   4.4,   5.2,   5.0,   5.6,   5.3,   5.6,   5.3,   5.3,   5.3,
 5.0,   5.0,   4.9,   4.6,   4.9,   4.2,   5.1,   4.2,   5.4,   4.5,   5.6,   4.9,
 5.1,   5.0,   4.5,   5.0,   4.6,   5.3,   5.3,   6.0,   6.2,   5.8,
 2.7,   4.4,   4.5,   5.1,   4.8,   5.5,   5.1,   5.6,   5.3,   5.3,   5.4,   5.0,
 5.3,   4.8,   4.9,   4.9,   4.5,   5.0,   4.2,   5.3,   4.3,   5.7,   4.6,   5.6,
 5.0,   4.9,   5.1,   4.6,   5.3,   5.1,   5.8,   5.9,   6.8,   8.3,
 9.2,   6.3,   5.6,   4.9,   5.5,   4.9,   5.6,   5.1,   5.4,   5.4,   5.0,   5.5,
 4.7,   5.2,   4.8,   4.8,   4.9,   4.4,   5.1,   4.2,   5.5,   4.3,   5.8,   4.8,
 5.3,   5.2,   4.7,   5.3,   4.8,   5.6,   5.3,   5.8,   6.2,   8.4,
 8.9,   6.6,   5.4,   5.6,   4.9,   5.7,   4.9,   5.6,   5.1,   5.2,   5.5,   4.8,
 5.5,   4.7,   5.1,   4.9,   4.7,   5.0,   4.4,   5.2,   4.2,   5.5,   4.5,   5.5,
 5.1,   5.0,   5.4,   4.6,   5.4,   4.8,   5.3,   4.6,   4.2,   1.9,
 5.4,   5.4,   5.4,   5.0,   5.7,   4.9,   5.8,   4.9,   5.4,   5.2,   4.9,   5.6,
 4.7,   5.4,   4.8,   5.1,   5.0,   4.6,   4.9,   4.3,   5.0,   4.2,   5.3,   4.7,
 5.2,   5.2,   4.7,   5.2,   4.4,   5.1,   4.3,   4.4,   3.0,  0.30,
 2.4,   4.5,   4.8,   5.3,   4.9,   5.8,   4.9,   5.6,   4.8,   5.0,   5.3,   4.7,
 5.4,   4.7,   5.4,   5.1,   5.1,   4.9,   4.6,   4.7,   4.1,   4.7,   4.1,   4.9,
 4.9,   4.8,   5.0,   4.3,   4.7,   4.0,   4.5,   3.7,   3.8,   5.1,
 6.8,   5.1,   4.9,   4.5,   5.5,   5.2,   5.7,   4.8,   4.7,   4.5,   4.6,   5.0,
 4.4,   5.2,   5.1,   5.7,   5.4,   4.9,   4.8,   4.5,   4.4,   3.6,   4.0,   4.0,
 4.7,   4.6,   4.4,   4.3,   3.4,   3.8,   3.3,   3.5,   2.3,  0.60,
 3.3,   4.6,   4.3,   3.7,   4.9,   6.3,   5.2,   5.4,   3.3,   3.4,   4.7,   4.3,
 3.9,   4.7,   5.9,   6.3,   6.4,   4.7,   5.2,   4.8,   4.1,   3.2,   2.5,   3.2,
 4.8,   4.0,   4.4,   3.3,   2.6,   2.2,   3.0,   3.1,   2.0,  0.90,
 4.2,   5.6,   4.4,  0.30,   5.4,   8.5,   4.8,   6.9,  0.20,   1.1,   5.6,   4.6,
0.80,   4.8,   6.4,   7.9,   9.0,   2.5,   6.4,   5.4,   4.6,   1.7,  0.50,  0.90,
 7.3,   2.2,   5.4,   2.5,  0.40,  0.50,   2.1,   4.6,   1.0,   4.1,


SEQUENTIAL EXECUTION TIME: 825480
PARALLEL EXECUTION TIME: 101671
SPEEDUP:    8.12
NUMBER OF PROCESSORS USED: 32
```