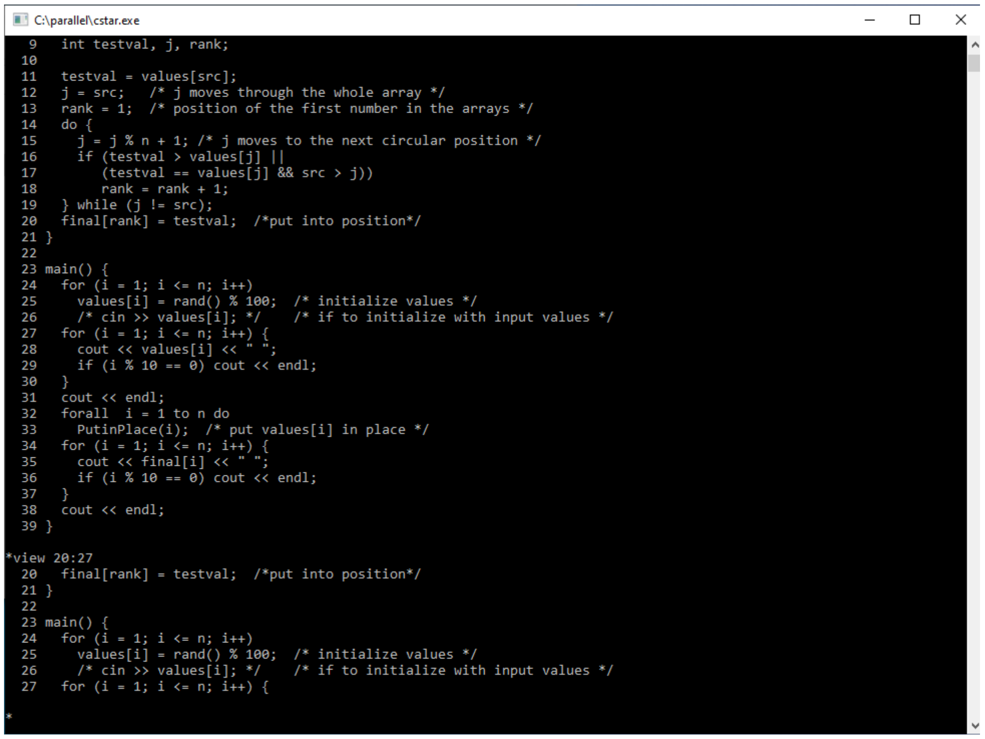
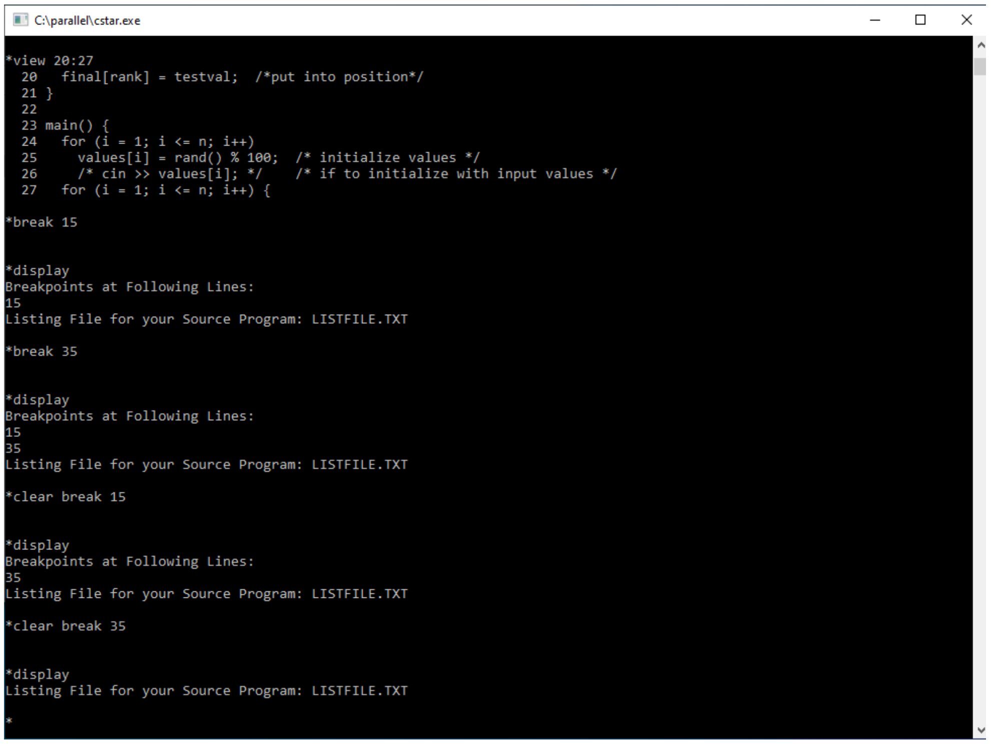
Yakshita B Rakholiya
yr92282n@pace.edu
Student ID: U01875270
Course: CS-610-22756
Project-2

- Part 1: Use code “parallel/RankSortParallel.c” as example code to work out contents of textbook Appendix C to show that you know how to use the various performance debugging tools provided by the C* environment. Submit screen captures organized into a Word file showing how you tried out the Appendix C performance debugging features on the sample code. This skill is valuable for your later projects.

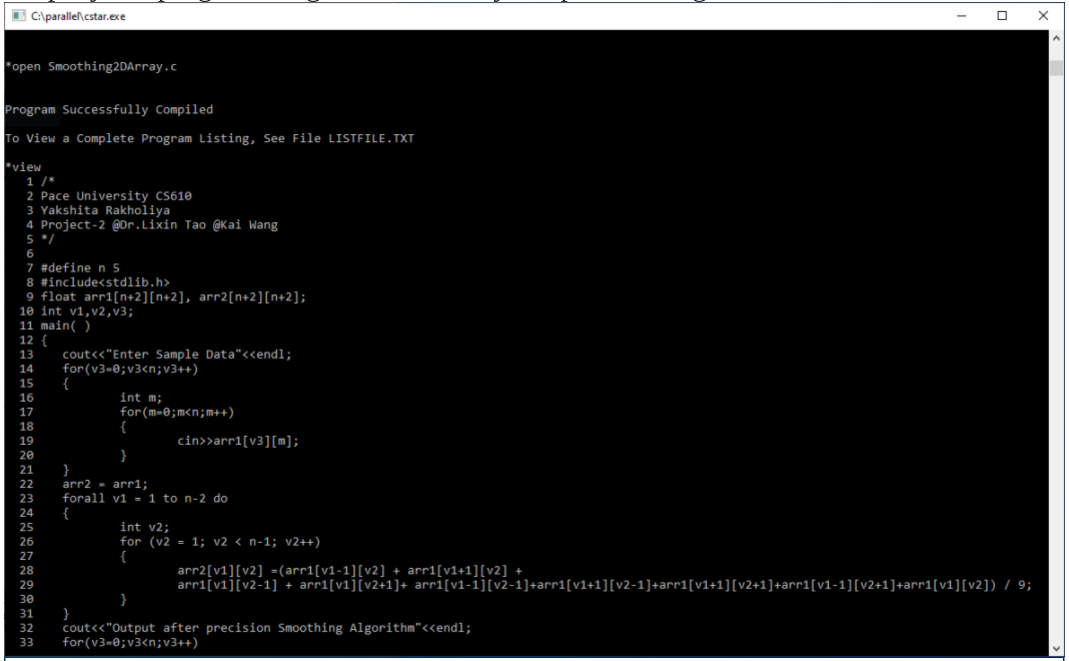

C:\parallel\cstar.exe

```
  9   int testval, j, rank;
 10
 11   testval = values[src];
 12   j = src;   /* j moves through the whole array */
 13   rank = 1;  /* position of the first number in the arrays */
 14   do {
 15     j = j % n + 1; /* j moves to the next circular position */
 16     if (testval > values[j] ||
 17        (testval == values[j] && src > j))
 18        rank = rank + 1;
 19   } while (j != src);
 20   final[rank] = testval;  /*put into position*/
 21 }
 22
 23 main() {
 24   for (i = 1; i <= n; i++)
 25     values[i] = rand() % 100;  /* initialize values */
 26     /* cin >> values[i]; */    /* if to initialize with input values */
 27   for (i = 1; i <= n; i++) {
 28     cout << values[i] << " ";
 29     if (i % 10 == 0) cout << endl;
 30   }
 31   cout << endl;
 32   forall  i = 1 to n do
 33     PutinPlace(i);  /* put values[i] in place */
 34   for (i = 1; i <= n; i++) {
 35     cout << final[i] << " ";
 36     if (i % 10 == 0) cout << endl;
 37   }
 38   cout << endl;
 39 }

*view 20:27
 20   final[rank] = testval;  /*put into position*/
 21 }
 22
 23 main() {
 24   for (i = 1; i <= n; i++)
 25     values[i] = rand() % 100;  /* initialize values */
 26     /* cin >> values[i]; */    /* if to initialize with input values */
 27   for (i = 1; i <= n; i++) {

*
```


C:\parallel\cstar.exe

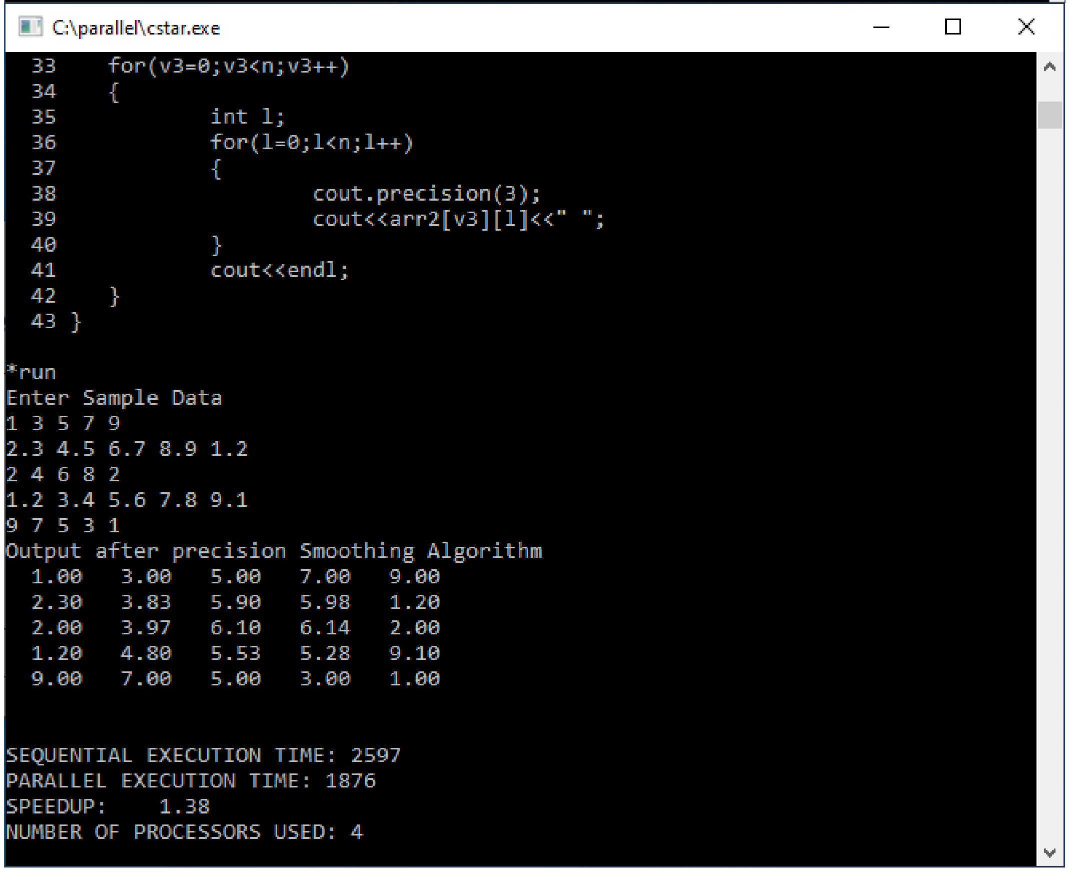
```
*view 20:27
 20   final[rank] = testval;  /*put into position*/
 21 }
 22
 23 main() {
 24   for (i = 1; i <= n; i++)
 25     values[i] = rand() % 100;  /* initialize values */
 26     /* cin >> values[i]; */    /* if to initialize with input values */
 27   for (i = 1; i <= n; i++) {

*break 15


*display
Breakpoints at Following Lines:
15
Listing File for your Source Program: LISTFILE.TXT

*break 35


*display
Breakpoints at Following Lines:
15
35
Listing File for your Source Program: LISTFILE.TXT

*clear break 15


*display
Breakpoints at Following Lines:
35
Listing File for your Source Program: LISTFILE.TXT

*clear break 35


*display
Listing File for your Source Program: LISTFILE.TXT

*
```

- Part 2: Exercise 2 on textbook page 91.
  To improve the visual quality of an image represented as a two-dimensional array of pixel values, a smoothing algorithm is sometimes applied. A simple smoothing algorithm is to replace the value of each pixel by the average of its immediate neighbors. Each pixel has eight immediate neighbors, including the diagonal neighbors. This algorithm replaces the value at each pixel by the average of nine pixels, consisting of itself and the eight neighbors.
  In this exercise, you are to write and run a parallel program to apply this smoothing algorithm to an image. To simplify the programming, do not modify the pixels along the four outer boundaries of the image array.



C:\parallel\cstar.exe

```
*open Smoothing2DArray.c


Program Successfully Compiled

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

*view
/*
Pace University CS610
Yakshita Rakholiya
Project-2 @Dr.Lixin Tao @Kai Wang
*/

#define n 5
#include<stdlib.h>
float arr1[n+2][n+2], arr2[n+2][n+2];
int v1,v2,v3;
main( )
{
   cout<<"Enter Sample Data"<<endl;
   for(v3=0;v3<n;v3++)
   {
           int m;
           for(m=0;m<n;m++)
           {
                   cin>>arr1[v3][m];
           }
   }
   arr2 = arr1;
   forall v1 = 1 to n-2 do
   {
           int v2;
           for (v2 = 1; v2 < n-1; v2++)
           {
                   arr2[v1][v2] =(arr1[v1-1][v2] + arr1[v1+1][v2] +
                   arr1[v1][v2-1] + arr1[v1][v2+1]+ arr1[v1-1][v2-1]+arr1[v1+1][v2-1]+arr1[v1+1][v2+1]+arr1[v1-1][v2+1]+arr1[v1][v2]) / 9;
           }
   }
   cout<<"Output after precision Smoothing Algorithm"<<endl;
   for(v3=0;v3<n;v3++)
```



C:\parallel\cstar.exe

```
   for(v3=0;v3<n;v3++)
   {
           int l;
           for(l=0;l<n;l++)
           {
                   cout.precision(3);
                   cout<<arr2[v3][l]<<" ";
           }
           cout<<endl;
   }
}

*run
Enter Sample Data
1 3 5 7 9
2.3 4.5 6.7 8.9 1.2
2 4 6 8 2
1.2 3.4 5.6 7.8 9.1
9 7 5 3 1
Output after precision Smoothing Algorithm
  1.00   3.00   5.00   7.00   9.00
  2.30   3.83   5.90   5.98   1.20
  2.00   3.97   6.10   6.14   2.00
  1.20   4.80   5.53   5.28   9.10
  9.00   7.00   5.00   3.00   1.00


SEQUENTIAL EXECUTION TIME: 2597
PARALLEL EXECUTION TIME: 1876
SPEEDUP:    1.38
NUMBER OF PROCESSORS USED: 4
```