# Project 1

## CSCI 475 – Parallel Computing

## Saint Cloud State University

Mark Christenson

# 3. Transpose.c

## Observations

Transpose assigns data to 4 processes from a 2 by 2 block of data where P1 and P2 would swap data to transpose the sub matrix. The implementation of the program doesn't seem to scale well, and the data of the whole matrix isn't processed.

## Output

```
A original
96 5 70 87
57 74 89 1
10 86 77 37
12 94 99 83
Result calculated
96 5 70 87
57 74 89 1
10 86 77 37
12 94 99 83
The data from matrix A received by process 0
96 5 70 87

The data from matrix A received by process 2
10 86 77 37

2 going to wait
2 done waiting
0 waited
The data recieved from vector b recieved by process 0
1
The data from matrix A received by process 1
57 74 89 1

1 going to wait
1 done waiting
The data recieved from vector b recieved by process 1
2
The data from matrix A received by process 3
12 94 99 83

3 going to wait
3 done waiting
```

The data recieved from vector b recieved by process 3
4
The data recieved from vector b recieved by process 2
3
data in 2
70
data in 1
5
data in 3
87
data in 0
96

## Source

```
#include<stdlib.h>
#include<stdio.h>
#include "mpi.h"


int main(int argc, char *argv[])
{
  int rows = 4;
  int N, npes;
    int myrank;
    int i, size, grp_size;
    int* buf; /*receive buffer*/
  int* vbuf;
    int recv_count;
    int myresult = 0;
    int result;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    MPI_Comm_size(MPI_COMM_WORLD, &grp_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */

  N = npes;
    int A[rows][rows];  /*data to be distribued from the root process*/
  int tA[rows][rows];
  int Result[rows][rows];
  int b[rows];

    if(myrank == 0)
    {
    //open file and place it into 2d array
    FILE *data;
    data = fopen("Data2.csv", "r");
    char currentChar = getc(data);
    char str[] = "";
    for(i=0;i<rows; i++){
      for(int j=0; j<rows; j++){
        A[i][j] = 0;
        while(currentChar!=',' && currentChar!=EOF){
          A[i][j] = 10 * A[i][j] + (currentChar - '0');
          currentChar = getc(data);
        }//end of searching for end of number
        currentChar = getc(data);
      }//end of filling current row
    }//end of filling matrix

    for(int i = 0; i < rows; i++){
```

```
48        b[i] = i + 1;
49      }//end filling vector b
50
51      //show example result
52      printf("A original\n");
53      for (size_t i = 0; i < rows; i++) {
54        for (size_t j = 0; j < rows; j++) {
55          printf("%d ", A[i][j]);
56          tA[i][j] = A[i][j];
57        }
58        printf("\n");
59      }//end of printing A and copying it into tA
60
61      Result[0][0] = A[0][0];
62      for (size_t i = 0; i < rows; i++) {
63        for (size_t j = 0; j < rows; j++) {
64          Result[j][i] = A[i][j];
65        }
66      }
67      Result[rows - 1][rows - 1] = A[rows - 1][rows - 1];
68      //End of transposing A into Result serially
69
70      printf("Result calculated\n");
71      for (size_t i = 0; i < rows; i++) {
72        for (size_t j = 0; j < rows; j++) {
73          printf("%d ", tA[i][j]);
74        }
75        printf("\n");
76      }//end of printing transposed A
77      }//end of initialization by P0
78
79      recv_count = rows*rows/npes;
80      buf = (int* )malloc(recv_count*sizeof(int));
81    vbuf = (int* )malloc(recv_count*sizeof(int));
82
83    MPI_Scatter(A,recv_count,MPI_INT,buf,recv_count, MPI_INT,0,MPI_COMM_WORLD);
84
85      printf("The data from matrix A received by process %d\n", myrank);
86      for (i = 0; i<recv_count; i++)
87          printf("%d ", buf[i]);
88      printf("\n\n");
89
90    MPI_Request request;
91
92    if(myrank == 0){
93      int flag;
94      int tbuf;
95      MPI_Status status;
96      MPI_Irecv(&tbuf, 1, MPI_INT, MPI_ANY_TAG, 0, MPI_COMM_WORLD, &request);
97      MPI_Wait(&request, &status);
98      printf("%d waited\n", myrank);
99    }//P0 sent message to continue
100   if(myrank > 0){
101     int flag;
102     MPI_Status status;
103     MPI_Isend(&flag, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
104     MPI_Test(&request, &flag, &status);
105     printf("%d going to wait\n", myrank);
106     while (!flag) {
107       MPI_Test(&request, &flag, &status);
108     }
109     printf("%d done waiting\n", myrank);
110   }//other proceses wait for P0
111
112   MPI_Scatter(b,rows/npes,MPI_INT,vbuf,rows/npes, MPI_INT,0,MPI_COMM_WORLD);
113
114   printf("The data recieved from vector b recieved by process %d\n", myrank);
115   for (size_t i = 0; i < rows/npes; i++) {
116     printf("%d ", vbuf[i]);
117   }//end of printing vector b
118   printf("\n");
119
```

```
120    MPI_Alltoall(tA,rows/npes,MPI_INT,buf,rows/npes,MPI_INT,MPI_COMM_WORLD);
121    printf("data in %d\n", myrank);
122    for (size_t i = 0; i < rows/npes; i++) {
123      printf("%d ", buf[i]);
124    }
125    printf("\n");
126
127    /*
128    if(myrank == 0){
129      printf("The difference is\n");
130      for (size_t i = 0; i < rows; i++) {
131        for (size_t j = 0; j < rows; j++) {
132          printf("%d ", Result[i][j] - A[i][j]);
133        }
134        printf("\n");
135      }
136    }//check if the result is correct
137    */
138
139    MPI_Finalize();
140
141      return 0;
142  }//end main
```