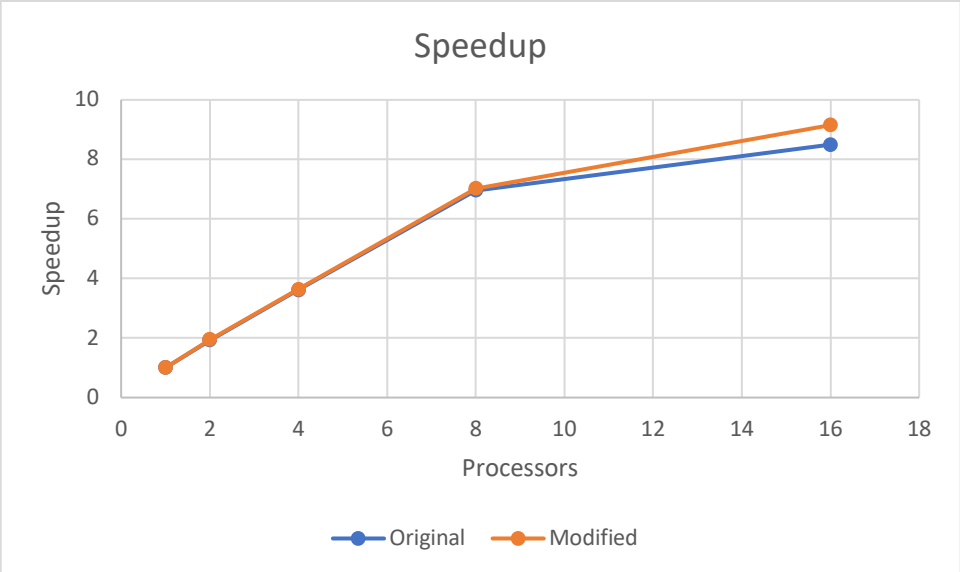
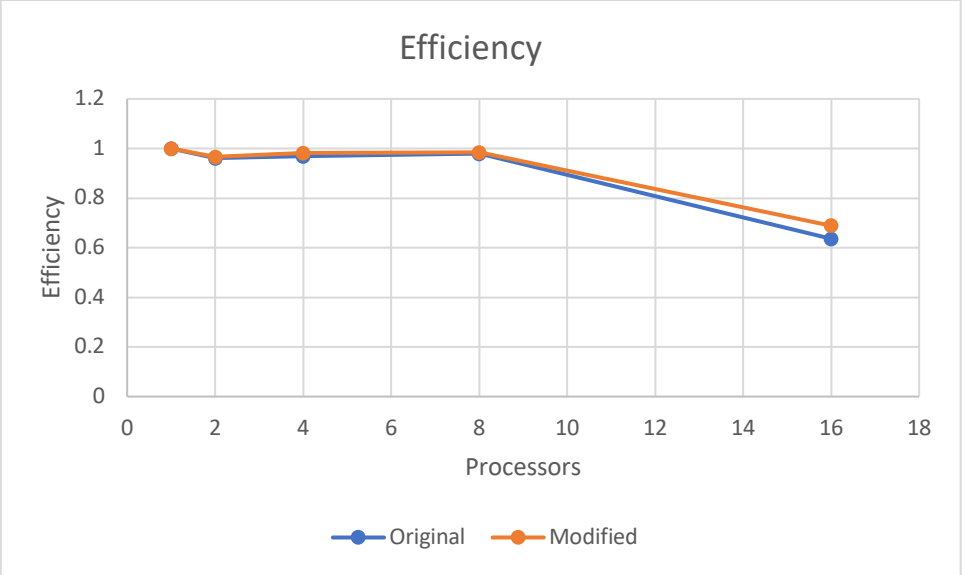
```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

int main( int argc, char *argv[])
{
    int n, i;
    double PI25DT = 3.14159265358979323846264338327950288419716939937510;
    double pi, h, sum, x;

    int numprocs, myid;
    double startTime, endTime;

    /* Initialize MPI and get number of processes and my number or rank*/
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);    //used to find an individual
process's rank in a communicator
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);   //used to find the number of processes
in a communicator

    /* Processor zero sets the number of intervals and starts its clock*/
    if (myid==0)
    {
       n=numprocs*100000000;
       startTime=MPI_Wtime();
       for(int i = 1; i < numprocs ; i++){
           MPI_Send(&n, 1, MPI_INT, i, myid, MPI_COMM_WORLD);
       }
    }
    else {
        MPI_Recv(&n, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }


    /* Broadcast number of intervals to all processes */
    //MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    /* Calculate the width of intervals */
    h   = 1.0 / (double) n;

    /* Initialize sum */
    sum = 0.0;
    /* Step over each inteval I own */
    for (i = myid+1; i <= n; i += numprocs)
    {
            /* Calculate midpoint of interval */
            x = h * ((double)i - 0.5);
            /* Add rectangle's area = height*width = f(x)*h */
            sum += (4.0/(1.0+x*x))*h;
    }
    /* Get sum total on processor zero */
    // MPI_Reduce(&sum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if (myid==0)
    {
     double local_sum;
```

```
     for(int i = 1; i < numprocs ; i++){
         MPI_Recv(&local_sum, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
         pi += local_sum;
     }
     pi += sum;
     }
     else {
         MPI_Send(&sum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
     }


     /* Print approximate value of pi and runtime*/
     if (myid==0)
     {
        printf("pi is approximately %.16f, Error is %e\n", pi, fabs(pi - PI25DT));
        endTime=MPI_Wtime();
        printf("runtime is=%.16f",endTime-startTime);
     }
     MPI_Finalize();
     return 0;
}
```

Result

I ran the program 3 times and used the median to get the value used for calculating speedup and scaled efficiency.





Discussion

The results of the original code are slightly slower than the modified one after 8 processors although I'm unsure as I only ran it 3 times and took the median. Having more runs would probably confirm whether this is true or not. The other results seem pretty consistent and close to each other.