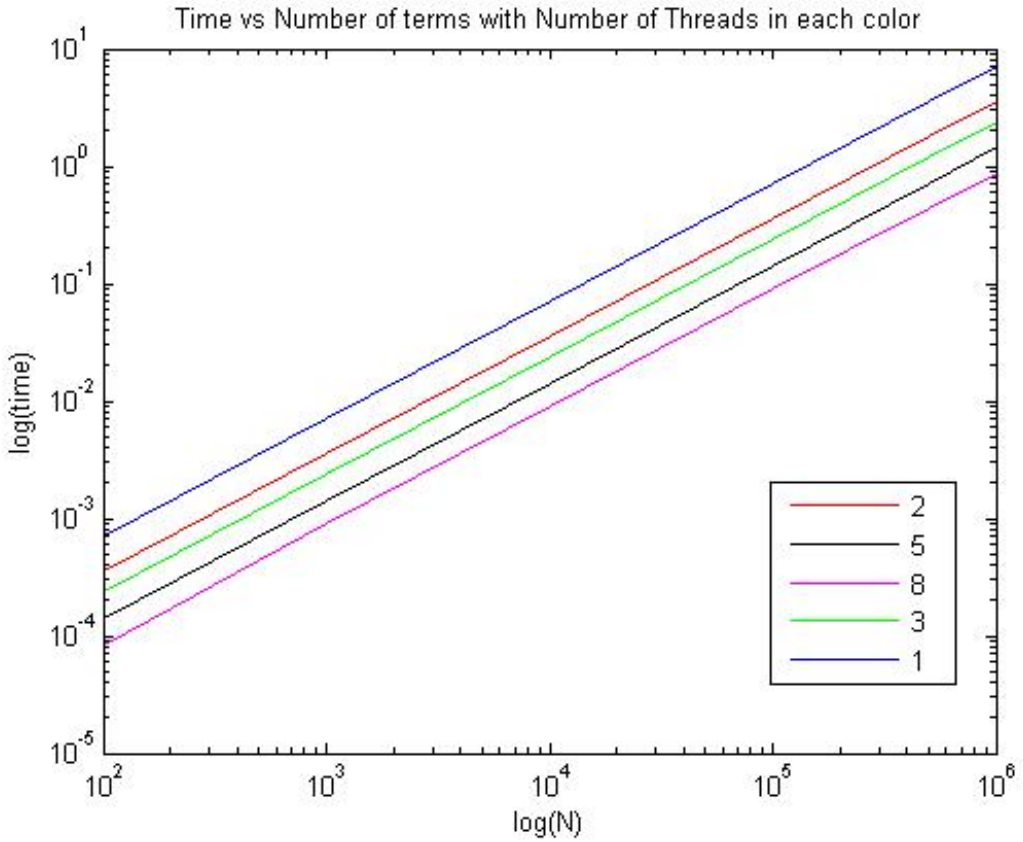
Question 1

Following is the graph for time vs N for the calculation of pi



it is clear from the above graph that time will increase as we increase the number of terms used for calculation of pi, which gives more accurate results as well. Also there are different lines, each for different number of threads used for calculations. The color code gives the number of threads used in OPENMP.

This also clearly indicates that using more number of threads increase the efficiency of the code.

Parallel Speedup

In order to calculate parallel speedup, the time taken for 1 thread was used as the sequential time taken and put in the following equation to get the speedup:

$$Parallel\ Speedup = \frac{Time\ taken\ for\ 1\ thread}{Taken\ for\ n\ threads}$$

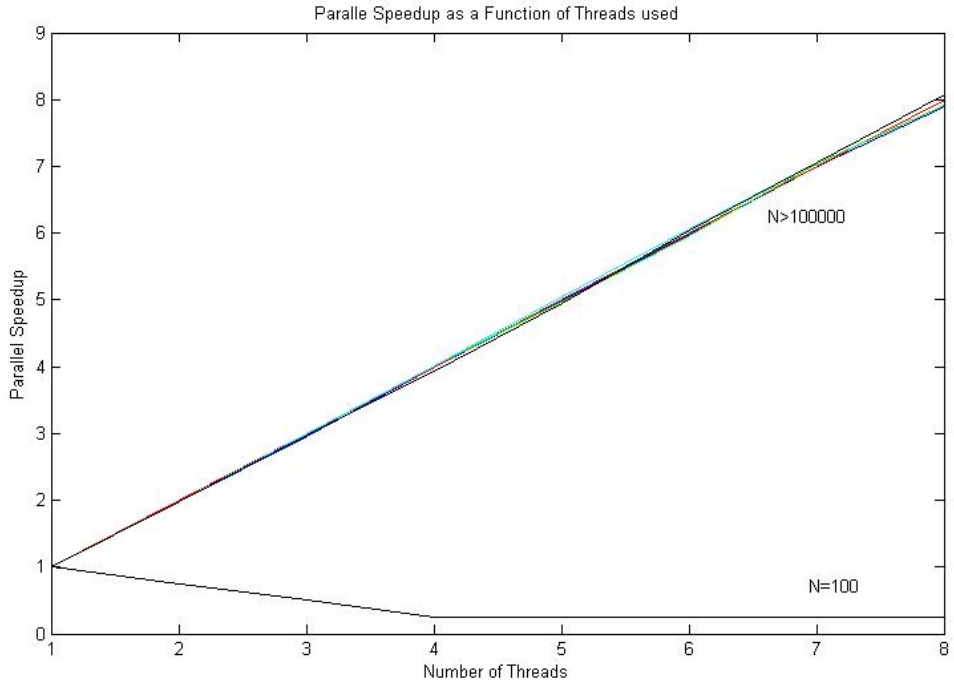
Parallel Efficiency

Efficiency is defined as the speedup per thread, so the equation for parallel efficiency becomes:

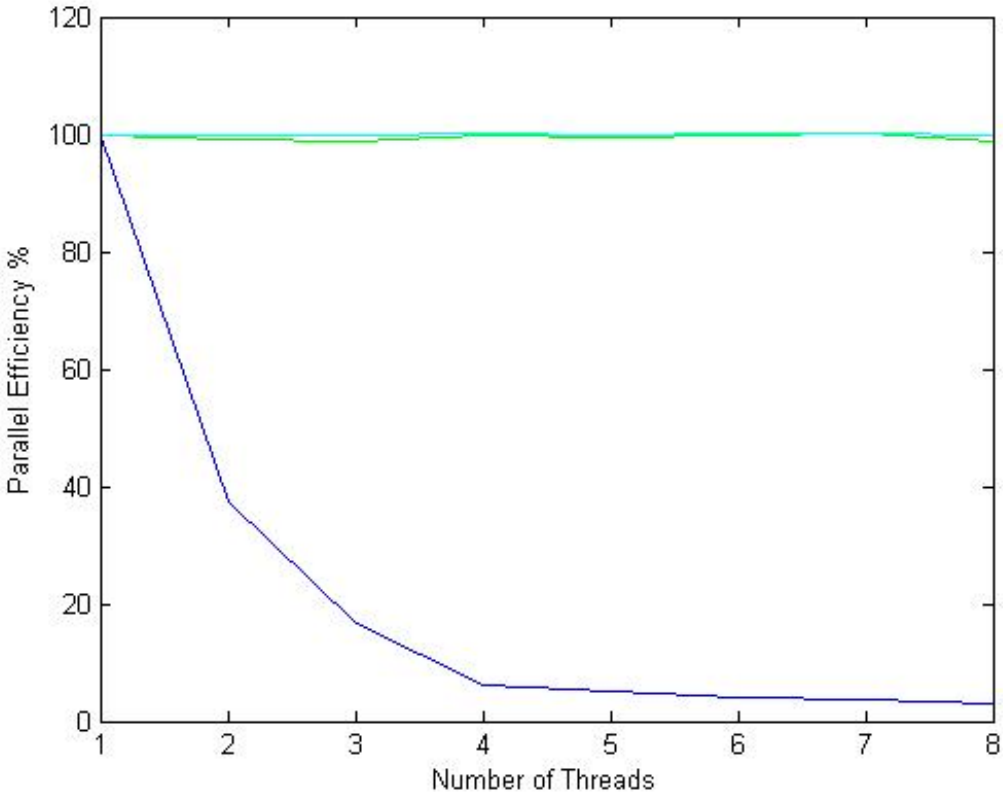
$$Parallel\ Efficiency = \frac{1}{P} * \frac{Time\ taken\ for\ 1\ thread}{Taken\ for\ n\ threads}$$

where P is the number of threads.

The results are as follows:



parallel speedup is around 8 for large N but for smaller N it goes below 1.



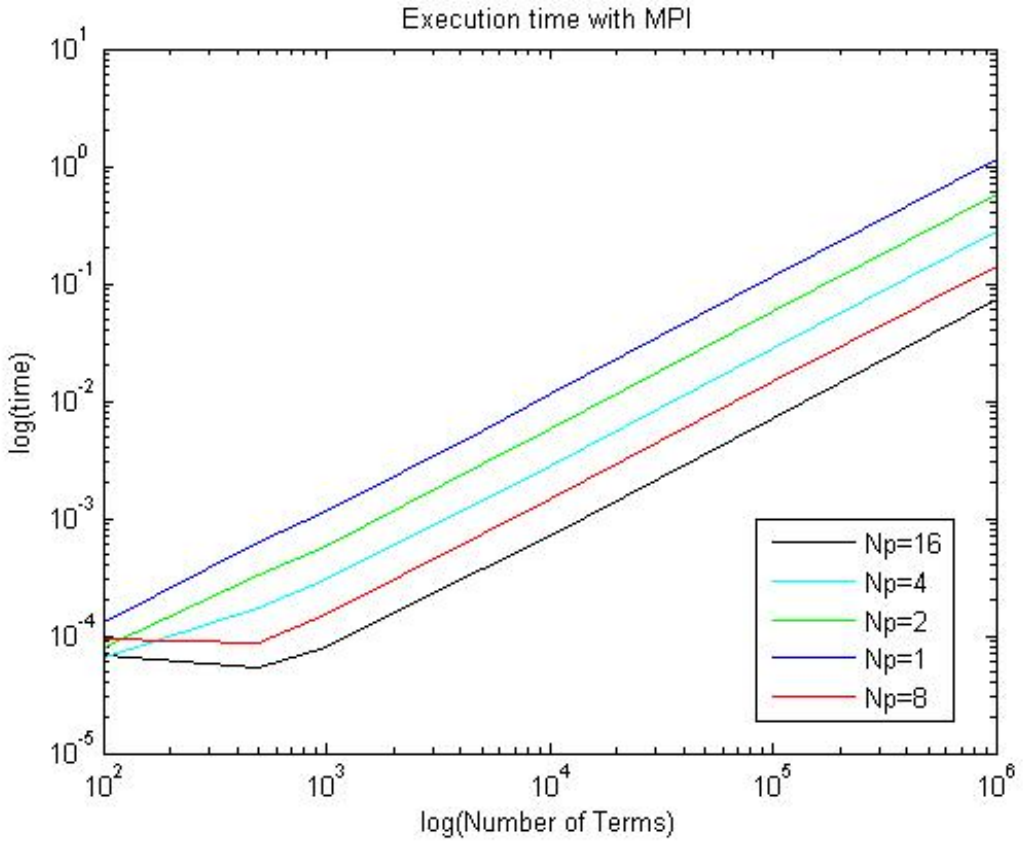
The lower line in the efficiency refers to N=100, which decreases by increasing the number of threads. In case of N>10000, the parallel efficiency is more or less constant.

Both parallel efficiency and speedup clearly shows that for small N it is better to run the code in sequential form since parallel only reduces the performance of the code.

Question 2

In case of MPI the results are as follows:

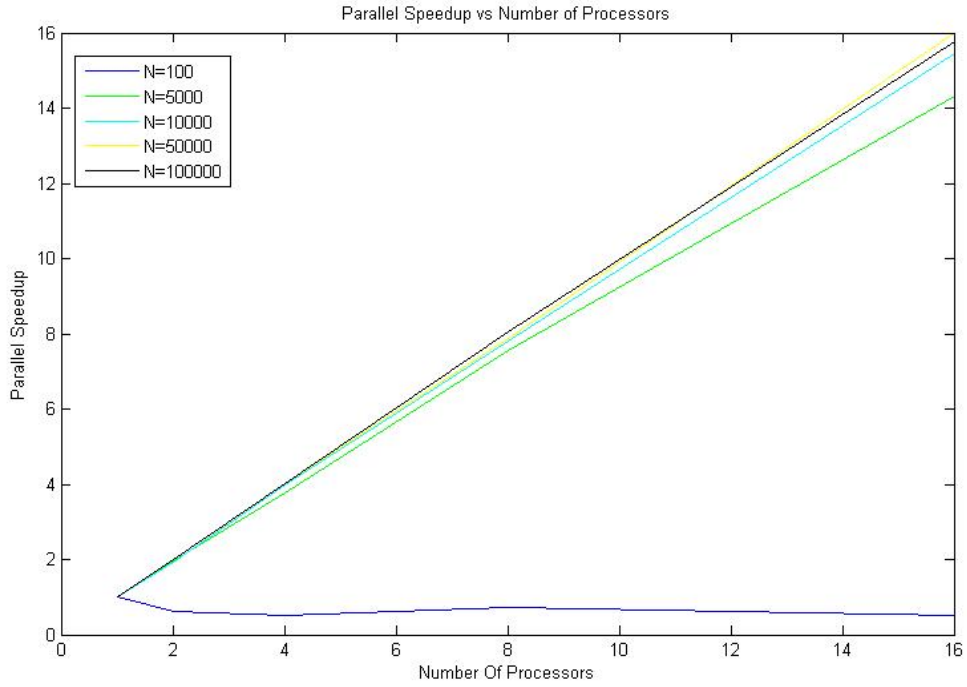
A

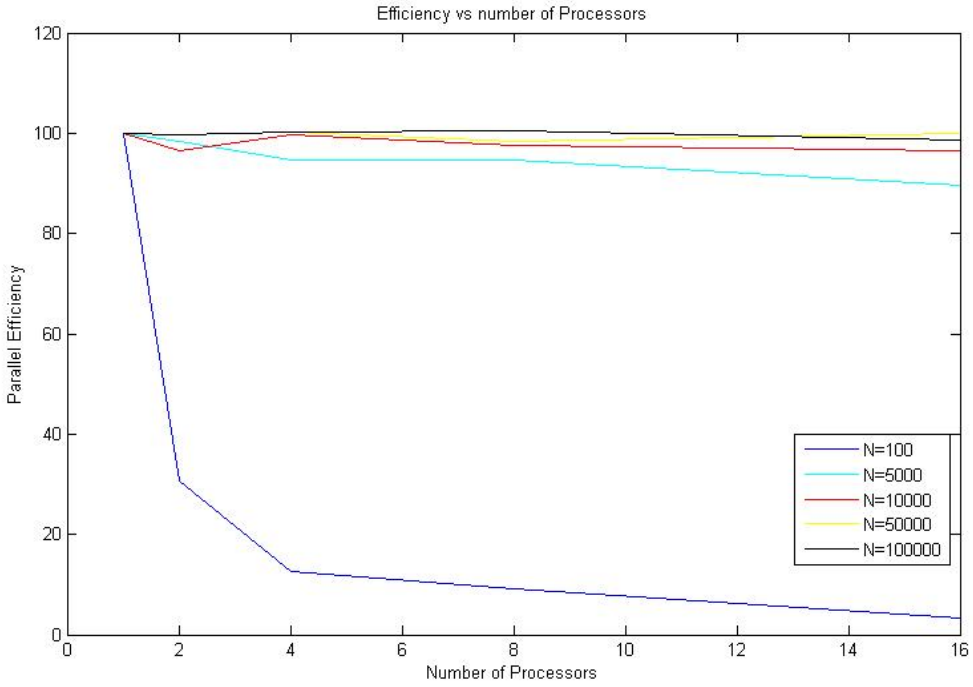


Again as expected, as we increase the number of terms for calculation, the time goes up. There is also a difference in the number of processors used which reduces the time. This is better visualized by the parallel efficiency of the code.

Parallel efficiency and speedup are calculated using the same equation mentioned in 1, and calculations were done in MATLAB.

Quite clearly in the following figures the speedup as well as the efficiency is very low for N=100, that means the sequential code runs better for smaller n, but again will give more error in the calculation of pi.





One thing is worth noting that for small N , both OPENMP and MPI have worse results that sequential code, which might be due to the reason that the most of the time is taken up in communicating with MPI and OPENMP and hence not so significant performance. For higher N, it is obviously clear that OPNEMP and MPI give superior performance.

Appendix 1

```
program PIomp
Use omp_lib
implicit none

double precision   :: mysum,pi
double precision   :: t_start, t_end, time
integer          :: Mflops, myid,Np,Nt, Nperprocess
integer          ::i,j,k,N, ntr
integer, dimension(1) :: Ninput
integer, dimension(8) :: Nthreads

Ninput =(/1000000 /)
Nthreads=(/ 1, 2, 3, 4, 5, 6, 7, 8/)
do ntr=1,size(Nthreads)
Nt=Nthreads(ntr)
!print *, Nt
call OMP_SET_NUM_THREADS(Nt)
!$OMP PARALLEL DO
do i=1,size(Ninput)
 N=Ninput(i)
 mysum =0.0



 myid = OMP_GET_THREAD_NUM()
 Nt = omp_get_num_threads()
 Np=omp_get_num_procs()
 Nperprocess= N/Nt
  t_start = OMP_GET_WTIME()
!print *, N, Np, Nt, myid, Nperprocess
 do j = 1,1000

  do k=myid*(N/Nt)+1,(myid+1)*(N/Nt),2

   mysum = mysum + 1.0/real(2*k-1)
   mysum = mysum - 1.0/real(2*k+1)

  end do

 end do

 pi = mysum*4/1000


 t_end = OMP_GET_WTIME()
 time = t_end - t_start


print *, N, Nt, tot_time

end do
!$OMP END PARALLEL DO
end do

end program
```

Appendix 2

```
program PImpi
implicit none
include 'mpif.h'

double precision    :: sum, pi
double precision    :: t_start, t_end, time
integer             :: myid, Np, Nt
integer             :: i,j,k,N,ierr
integer, allocatable, dimension(:) :: number

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, Np, ierr)

 N=100000;

call MPI_BCAST(n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
allocate(number(1:n))
number = 0

call MPI_BCAST(number, n, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)

do i =1,n
t_start = MPI_WTIME()

  do j = 1,1000
 sum = 0.0
    do k = myid*(number(i)/Np)+1, (myid+1)*(number(i)/Np), 2
       sum = sum + 1.0/real(2*k - 1)
       sum = sum - 1.0/real(2*k + 1)
    end do
 end do

 pi = sum*4

 call MPI_ALLREDUCE(sum,pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)
t_end = MPI_WTIME()
   time = t_end - t_start

  print *, number(i), pi, time

end do

call MPI_FINALIZE(ierr)

end program
```