Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Parallel Processing: Assignment 1

Objective:

The goal is to solve the FIND SUM problem using at least two different logical topologies and document our results for the various parameters that affect execution. In this report the observations for the following two topologies have been documented.

- a ring of k processors
- a hypercube of $k = 2^d$ processors

Ring formulations:

A ring is a static network in which each node has two neighbors, one on the left and right. To evaluate the performance metrics of a ring topology, the following two formulations have been used:

1. The source node creates an array A of random numbers between -1 and 1. The size of the array (n) is taken as an input parameter. Following the creation of the array, MPI_Scatter() function is used to distribute a single sub arrays of size(n/k) to the k processors. Each of the processors compute the local sum of the sub array that they have received. The total sum is sequentially calculated by each node in the ring by adding its local sum to the total sum and passing it along. This requires us to send p-1 messages starting from the source node to compute the total sum of the input array.

2. In the second formulation, the MPI_Scatter() and MPI_Gather() functions are used. First a ring topology is setup with MPI_Cart_create() command. Then the array is distributed in a one to all personalized manner. On each node the partial sum is computed and using MPI_Gather() an all to one reduction is performed on the ring.

MPI calls used in the ring formulations:

MPI Calls	Description
int MPI_Init(int *argc, char ***argv)	used to Initialize the MPI execution environment
Int MPI_Comm_size(MPI_Comm comm, int *size)	Determines the size of the group associated with a communicator
int MPI_Comm_rank(MPI_Comm comm, int *rank)	Determines the rank of the calling process in the communicator
int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int source, MPI_Comm comm)	Sends personalized data from one process to all other processes in a communicator
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)	Performs a blocking send

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)	Blocking receive for a message
int MPI_Barrier(MPI_Comm comm)	Blocks until all processes in the communicator have reached this routine.
int MPI_Finalize()	Terminates MPI execution environment

Parameter Ranges:

To test the two mentioned ring formulations, the following parameter ranges were used:

Array size: from 1600 to 204800

Processors: n=2, k=4 and n=4, k=4

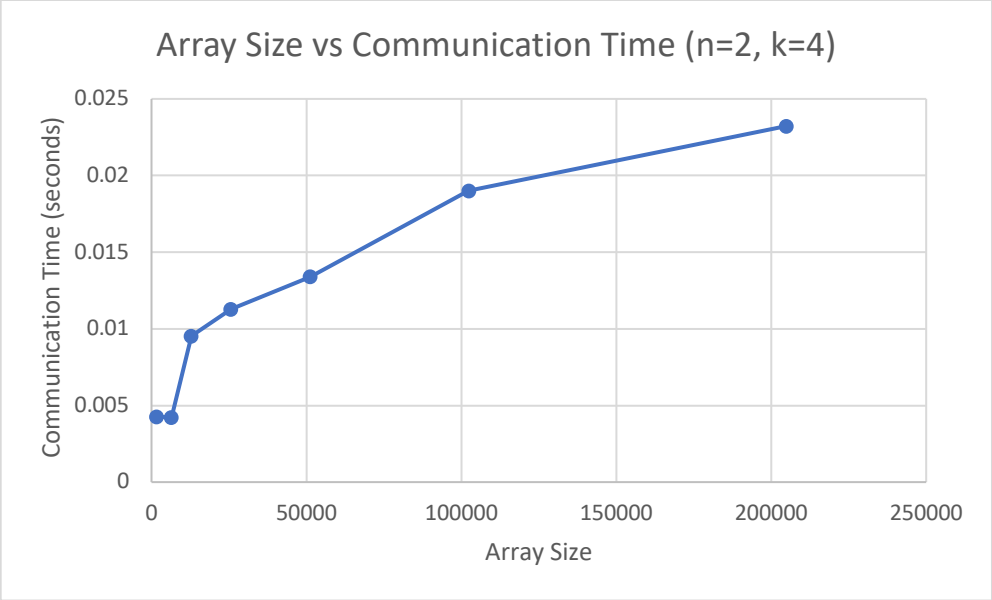
Results:

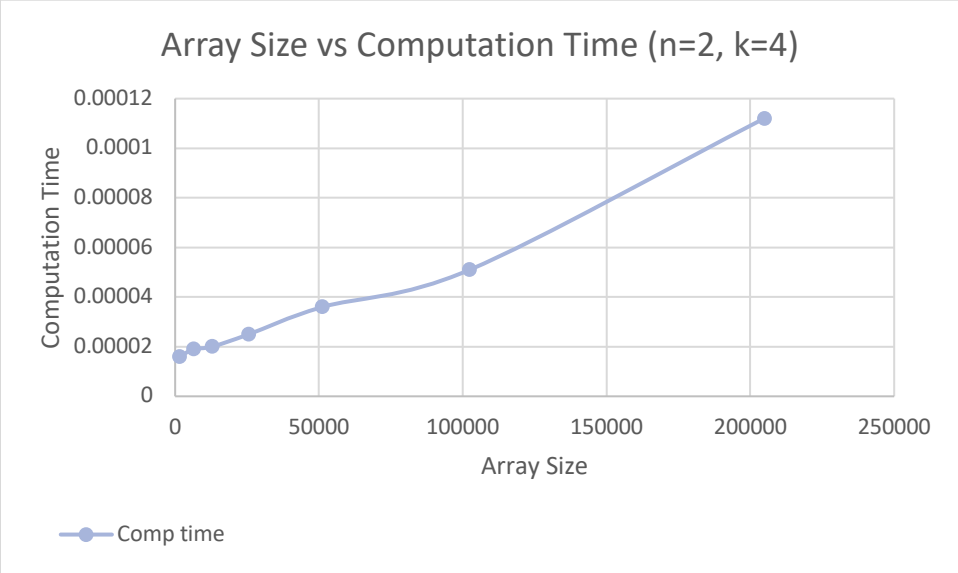
The tabulated results are the average reading observed for each of the above-mentioned parameters.

- **First Formulation: For n=2, k=4:**

Array Size	Communication Time (seconds)	Comp Time	Sum
1600	0.000827	0.000016	46.625047
6400	0.001473	0.000019	214.519672
12800	0.002476	0.00002	412.200792
25600	0.003892	0.000025	797.205789
51200	0.005116	0.000036	1607.793271
102400	0.006069	0.000051	3242.985013
204800	0.009442	0.000112	6396.940557

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

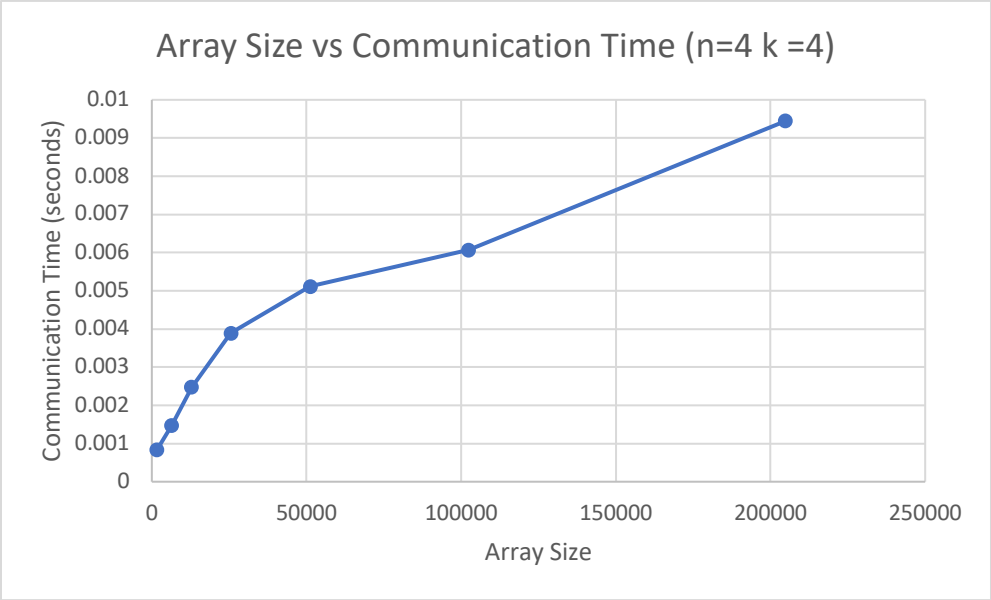


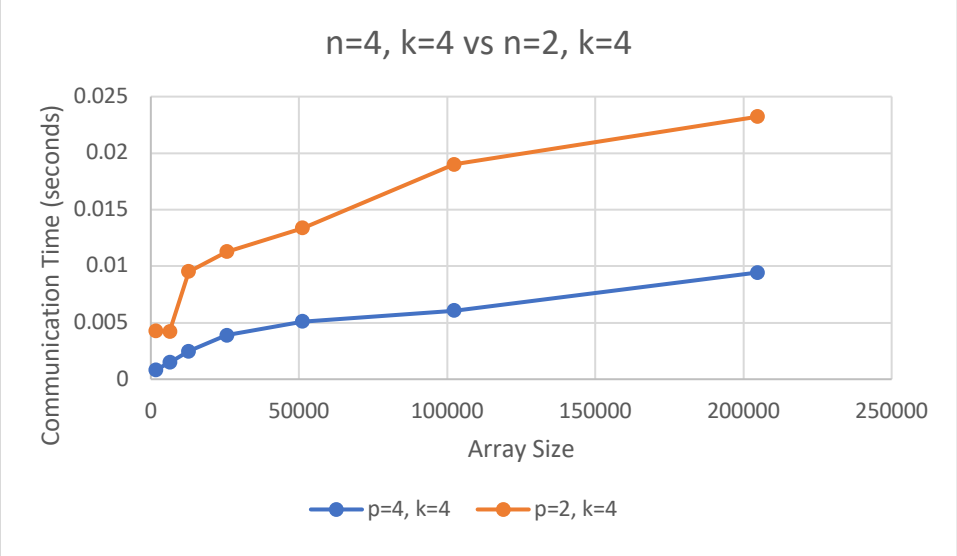


- **First Formulation: For n=4, k=4:**

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Array size	Communication Time(seconds)	Comp Time(seconds)	Sum
1600	0.004263	0.000171	47.594514
6400	0.004225	0.000107	203.796314
12800	0.009527	0.000131	405.757963
25600	0.011271	0.000096	806.212733
51200	0.013382	0.000093	1603.452141
102400	0.018999	0.000107	3202.598314
204800	0.023223	0.000102	6396.355134

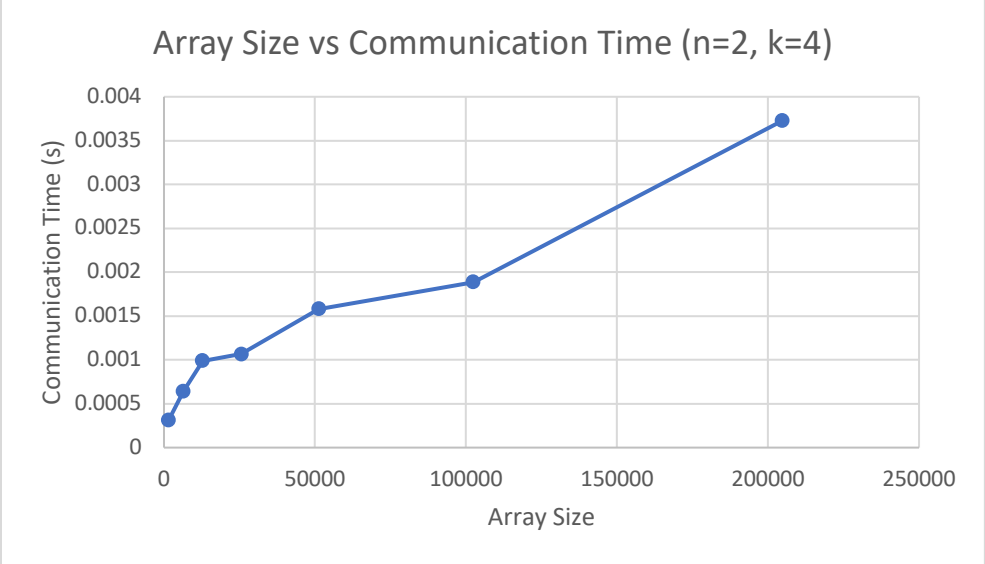




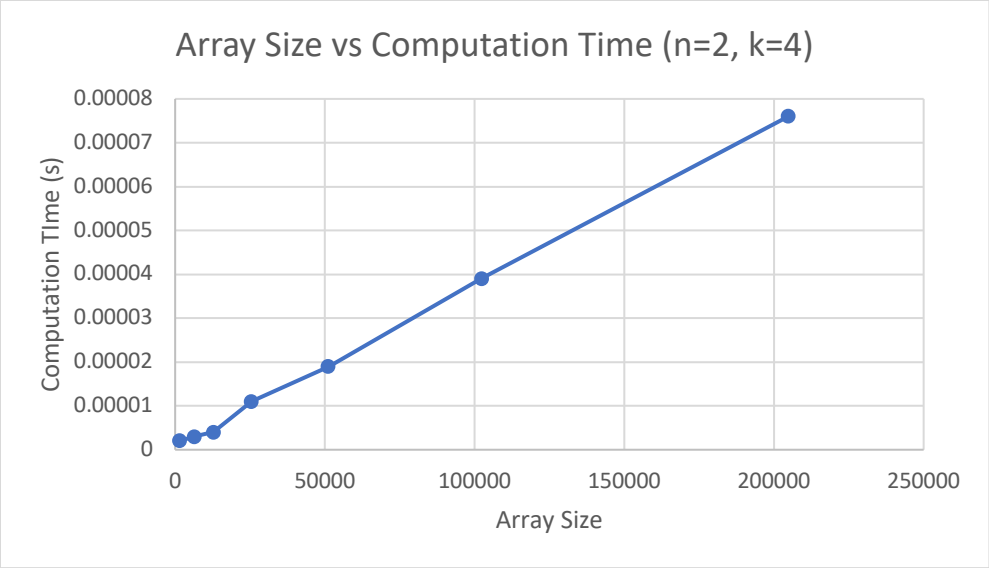
Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

- **Second Formulation: n = 2, K = 4**

Ring size	Comm time	Comp time	Sum
1600	0.000316	0.000002	45.594527
6400	0.000641	0.000003	207.796454
12800	0.000989	0.000004	423.756863
25600	0.001065	0.000011	810.212323
51200	0.001579	0.000019	1601.45216
102400	0.001886	0.000039	3206.59823
204800	0.003728	0.000076	6392.35563

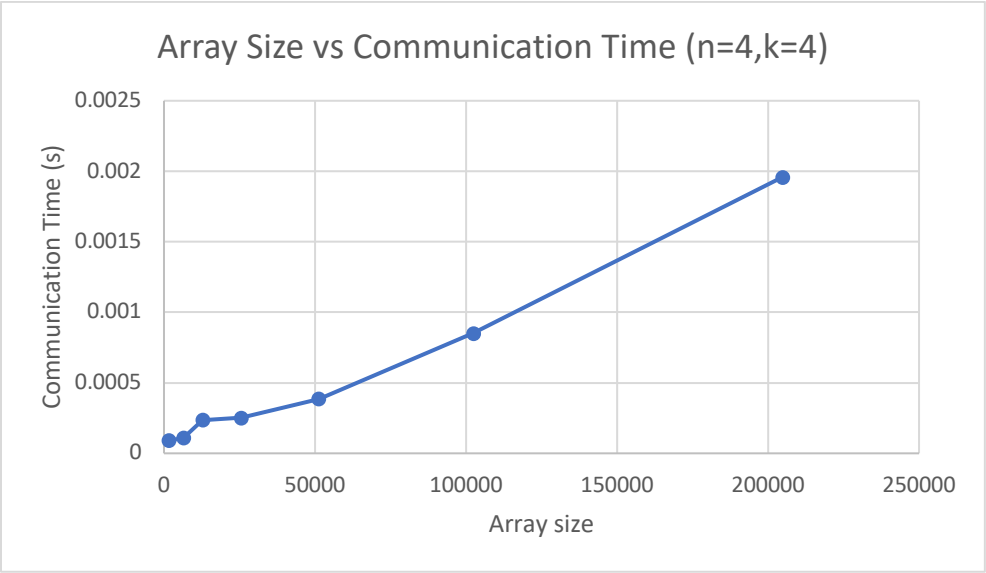


Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

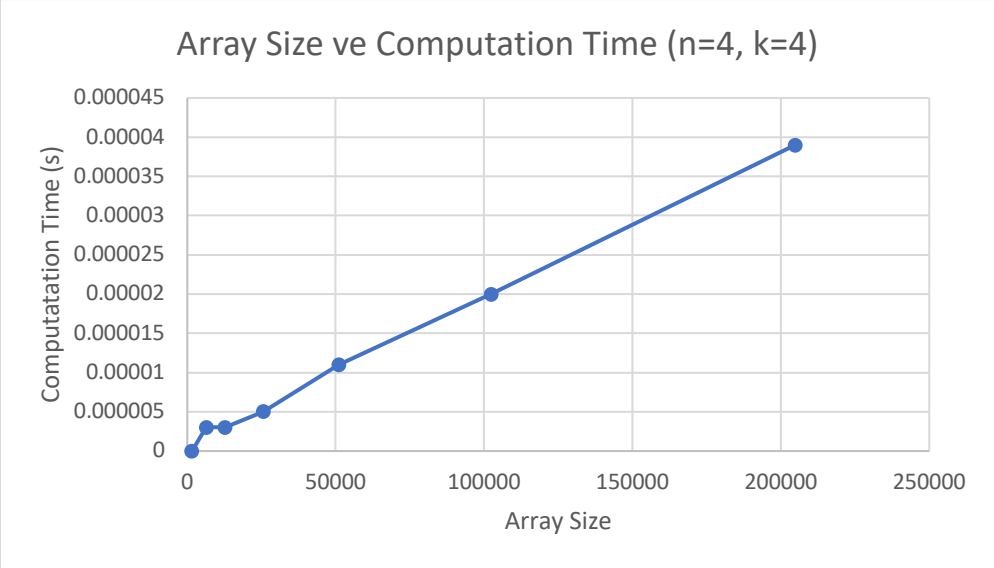


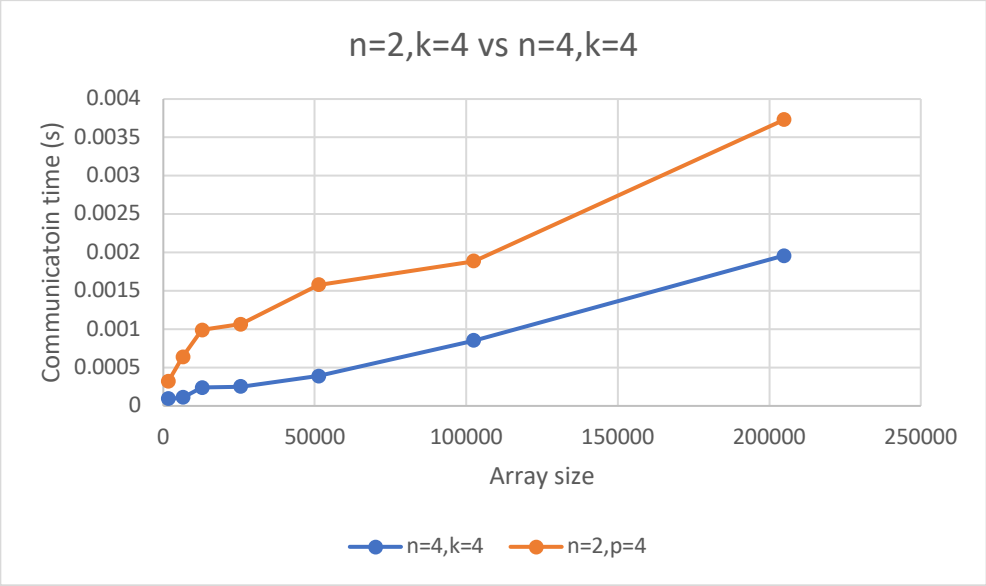
- **Second Formulation: n = 4, K = 4**

Ring size	comm time	Comp time	Sum
1600	0.000091	0	46.625054
6400	0.000109	0.000003	214.519831
12800	0.000237	0.000003	412.200801
25600	0.000253	0.000005	797.204589
51200	0.000387	0.000011	1607.82271
102400	0.000852	0.00002	3242.98001
204800	0.001959	0.000039	6396.94046



Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283





Analysis for Ring:

- We can observe from the above graphs that there is a linear increase in the total computation time with respect to the array size for a give number of processors. This is because with an increase in array size. The amount of work done by each processor increases.
- We also observe an increase in the communication time between the nodes with an increase in array size. This is because the communication time is dependent on the per

word transfer time. When we have a larger array, a larger dataset is needed to be sent to the different nodes.

- But we can observer from the final graph that the communication for 16 processors is lesser than that of 8 processors for data of the same size. This is because the data is split into smaller chunks. Thus, fewer words need to be transferred between each of the nodes.
- Similarly, the computation time is reduced as well with an increase in the number of processors as the number of elements per subarray that each node calculates the sum of is much smaller.
- When comparing the two formulations of the ring topology we see that the scatter and gather technique is faster than sending the store and forward technique used in formulation 1. This is because in formulation 1, each node has to forward its data to the next node where the local sum is added. The source node get the total sum in the final step. Thus p-1 communication steps are needed. Where as while doing gather, each node can send its locally computed sum directly to the source node where the total sum is calculated.

Hypercube Design:

- MPI_CART_CREATE can be used to describe Cartesian structures of arbitrary dimension. *n* -dimensional hypercube is an *n* -dimensional torus with 2 processes per coordinate direction. Thus MPI_CART_CREATE is used to create n-dimensional hypercube.
- To distribute array to processes MPI_SCATTER API was used.
- To collect the result, two different implementations were used : MPI_Gather and MPI_Reduce.
- In one implementation, the MPI_Gather is used to gather all the local sums of all the processes to the source node (Process 0) to perform the final sum operation.
- In another implementation, The MPI_Reduce operation is used to perform the summation of the local sums of all the processes and send the final sum to the source node.

Hypercube Experiment Results:

MPI_Scatter+MPI_Gather

1. p=8, k = 1

Array Size	Computation Time	Communication Time	Total Array Sum
100	0.001090	0.000121	54.682484
1000	0.002192	0.000211	508.125427
10000	0.029856	0.000067	4971.321289

100000	0.359836	0.056048	49986.851562
1000000	3.745659	0.633525	500006.562500

2. p = 4, k = 2

Array Size	Computation Time	Communication Time	Total Array Sum
100	0.002446	0.000101	54.682484
1000	0.004132	0.000162	508.125427
10000	0.031551	0.000777	4971.321289
100000	0.251401	0.000317	49986.851562
1000000	2.983288	0.049664	500006.562500

MPI_Scatter+MPI_Reduce:

1. p = 8, k = 1

Array Size	Computation Time	Communication Time	Total Array Sum
100	0.015227	0.000092	54.682484
1000	0.008061	0.000184	508.125427
10000	0.060227	0.000182	4971.321289
100000	0.381484	0.009154	49986.851562
1000000	3.968842	0.003773	500006.562500

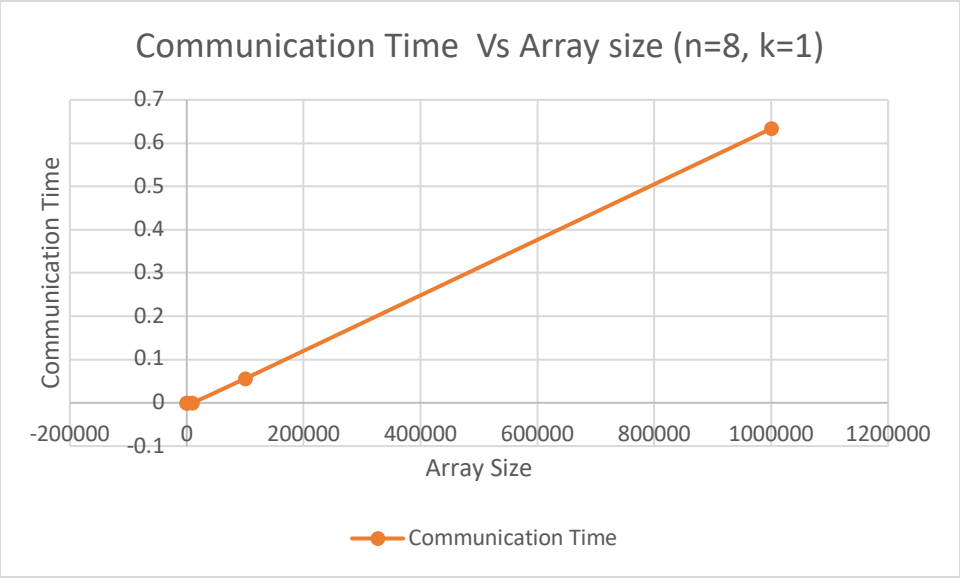
2. P = 4, k = 2

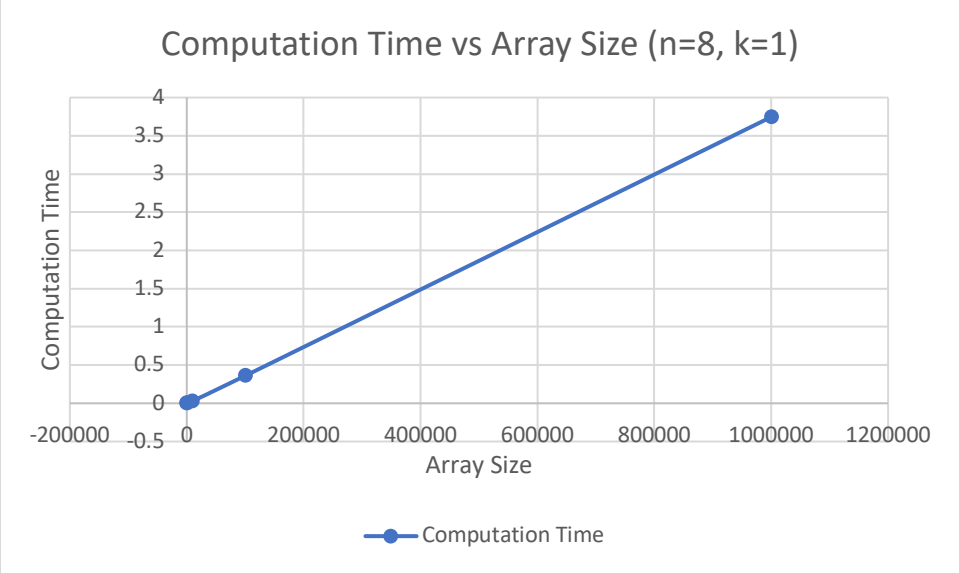
Array Size	Computation Time	Communication Time	Total Array Sum
100	0.015586	0.000051	54.682484
1000	0.009829	0.000057	508.125427
10000	0.053814	0.000084	4971.321289
100000	0.257763	0.000380	49986.851562
1000000	2.223196	0.004692	500006.562500

Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

Graphs for the above experiment:

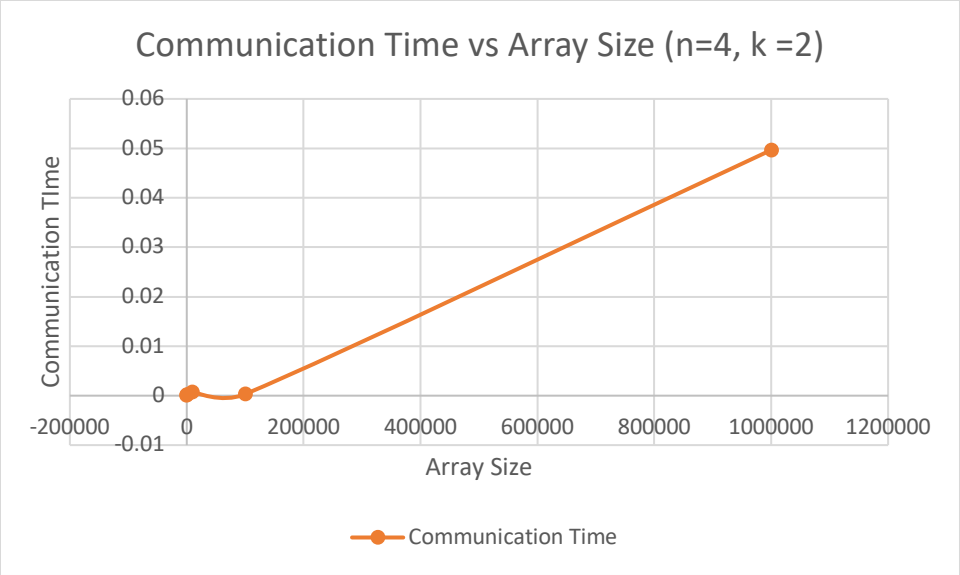
1.Design 1, p=8, k = 1:

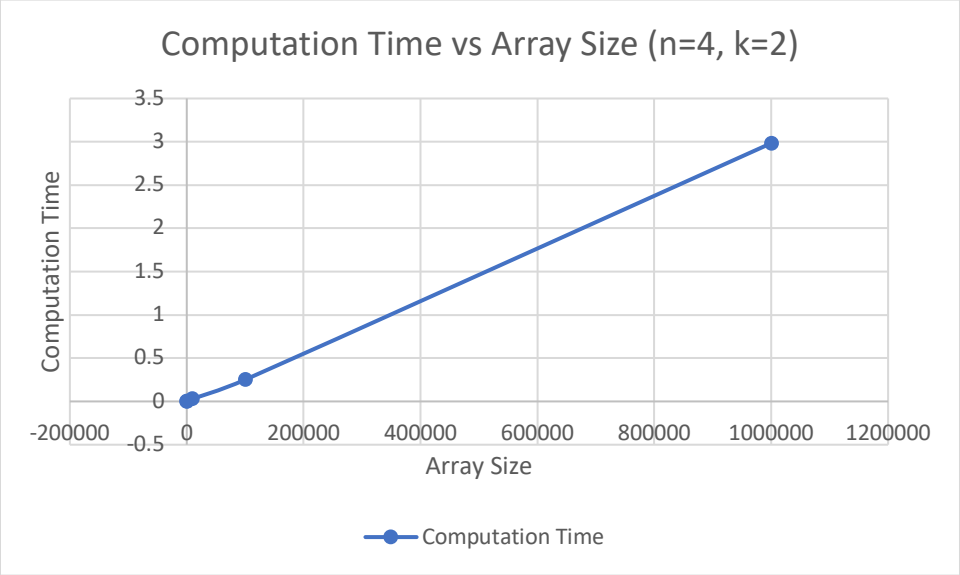




2.Design 1, p=4,k=2

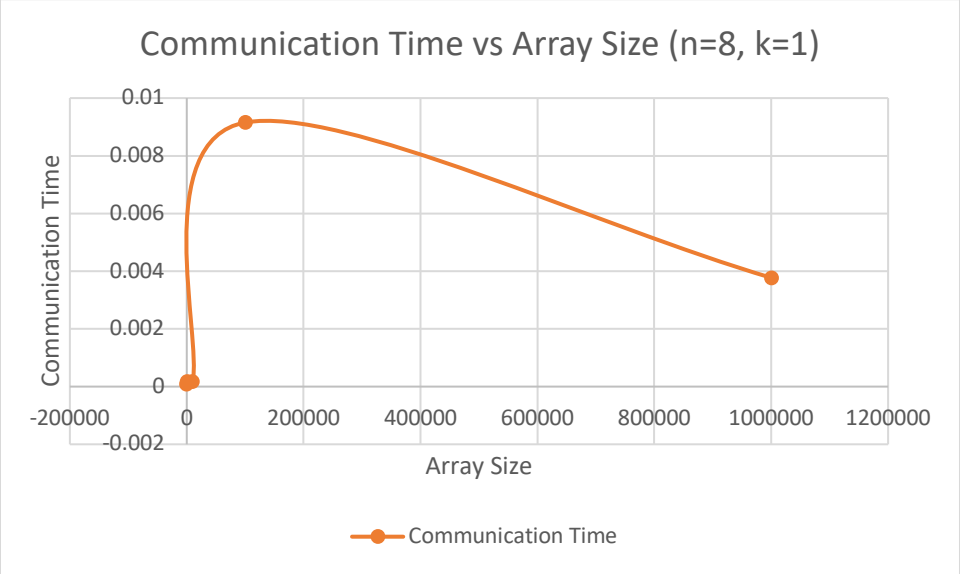
Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

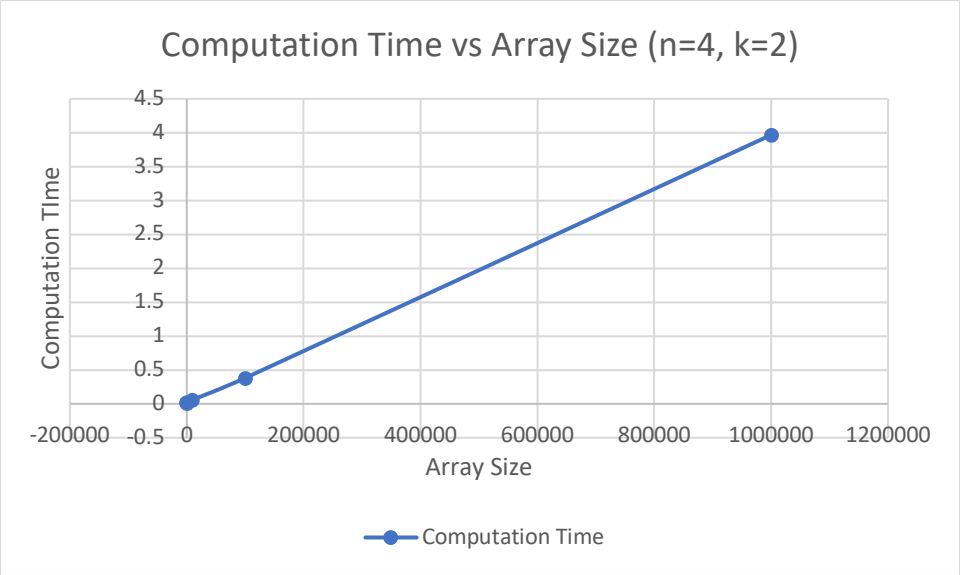




Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

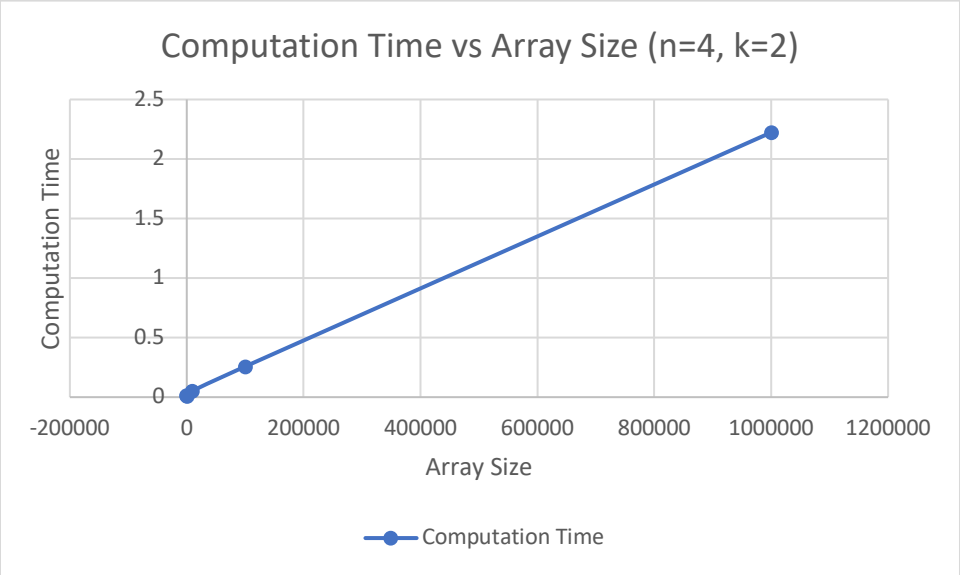
3.Design 2, p=8,k=1

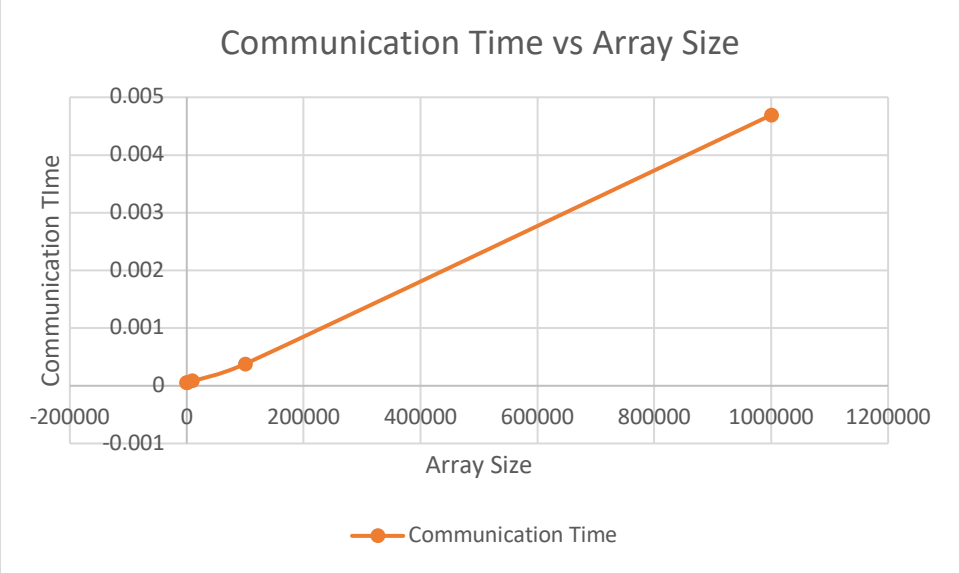




Aishwarya Ganesh: 665021069
Ankith C Kowshik: 678324283

4.Design2, p = 4, k = 2





Analysis for Hypercube implementation:

- From the above figures, in both implementations the computation time increases with increase in the size of the array. The communication time also increases with the increase in the size of the array.

- This is understandable since the increase in array size causes an increase in the load on the processors.
- The experiment was conducted with 2 different parameter ranges. In case of the first implementation with MPI_Scatter and MPI_Gather, an increase in the number of cores and a decrease in the number of physical processors, decreased the computation time as well as communication time for very large array sizes, to be specific, 100000, 1000000.
- However an increase in number of cores and decrease in the number of physical cores, increased communication time as well as computation time for small array sizes like 100,1000, 10000.
- In case of the second implementation with MPI_Scatter and MPI_Reduce, with parameters of p=4,k=2 yields a lower computation time and communication time.
- It can be understood that an increase in the number of cores caused the above mentioned difference in computation time.

Analysis Ring VS Hypercube:

- From the above tables we see as the number of processors increases, communication time is lesser for the hypercube as compared to the ring. This is because in a hypercube at maximum each node is $\log_2(p)$ hops away. While in a ring the max distance between the nodes increases with increase in the number of processors.
- Also the hypercube has the added advantage of having redundancy or alternative paths in case of failure of one of the nodes or links. A ring ca have a maximum of 2 failed links or nodes.

Lessons Learnt:

- This assignment introduced MPI programming through the use of topologies such as ring and hypercube on the extreme cluster. We were introduced to many MPI functions and learnt how the transfer of messages within a topology affect the overall performance of a given program or system.
- An experimental approach was used to figure out the differences and effects of changing various parameters withing a given topology. It provided the opportunity to evaluate how parameters such as array size and number of processors affect computation and communication time.
- We were able to see the advantages of techniques such as cut through routing compared to store and forward, one to all personalized broadcast through scatter and all to one personalized reduction through gather operations.