Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

PARALLEL PROCESSING: LAB1

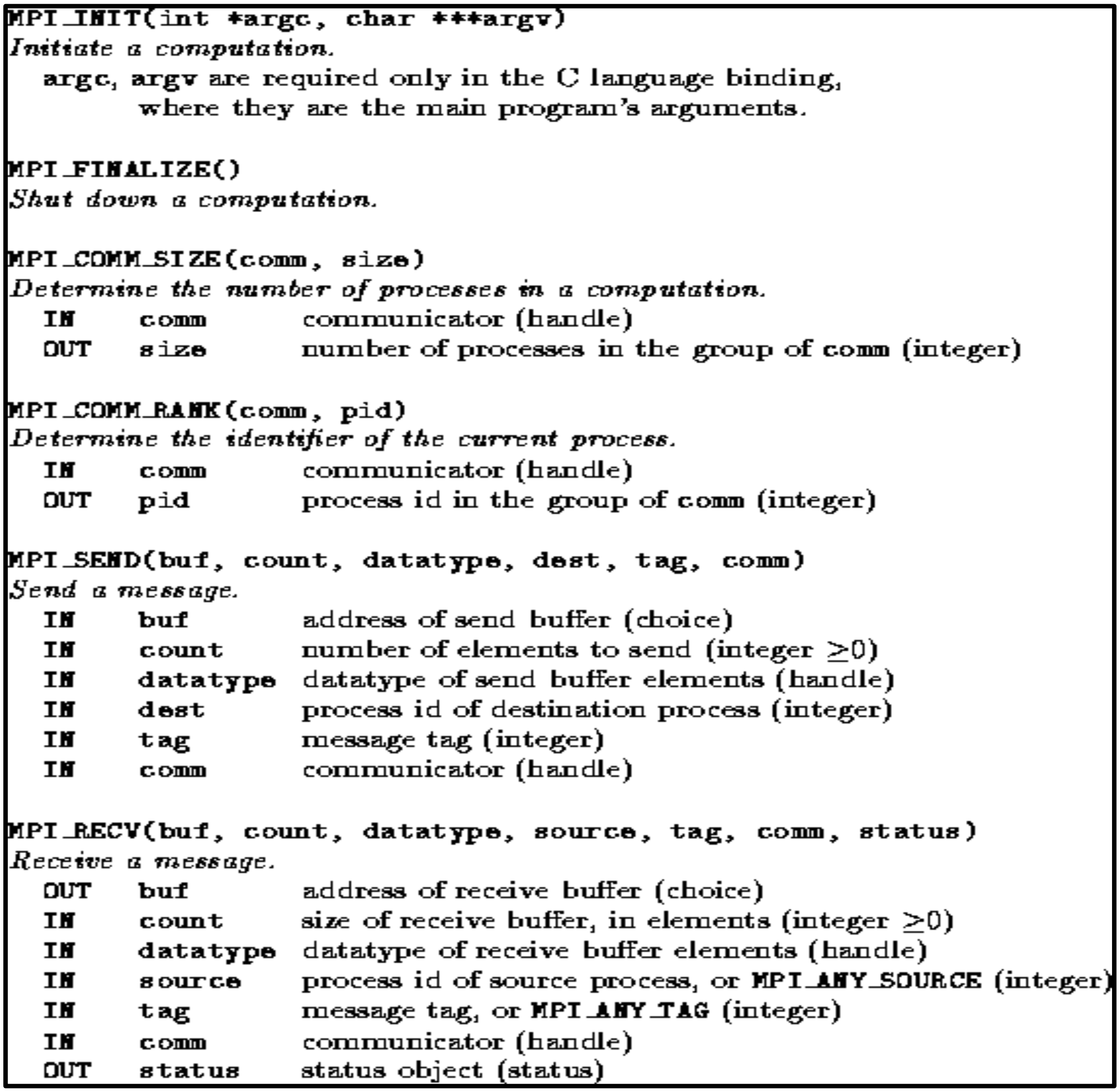
Goal: To create a large array A of n numbers at one node, distribute blocks of the array across the processors, and find the global sum of the numbers in A using ring and hypercube topology.

MPI Functions used in the Ring and Hypercube Algorithm:

1. **MPI_INIT**: Initiate an MPI computation. (Called only once during the execution of a program)
2. **MPI_FINALIZE**: Terminate a computation.
3. **MPI_COMM_SIZE**: Determine number of processes.
4. **MPI_COMM_RANK**: Determine my process identifier.
5. **MPI_SEND**: Send a message.
6. **MPI_RECV**: Receive a message.
7. **MPI_Get_processor_name (processor name, &name_len)**: Returns the name of the processor on which it was called now of the call.
8. **MPI_Wtime:** Returns the floating-point number of seconds, representing elapsed wall-clock time since some time in the past.

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

Explanation for each MPI_COMMAND primitves:



MPI_INIT(int *argc, char ***argv)
Initiate a computation.
argc, argv are required only in the C language binding,
where they are the main program's arguments.

MPI_FINALIZE()
Shut down a computation.

MPI_COMM_SIZE(comm, size)
Determine the number of processes in a computation.

IN	comm	communicator (handle)
OUT	size	number of processes in the group of comm (integer)

MPI_COMM_RANK(comm, pid)
Determine the identifier of the current process.

IN	comm	communicator (handle)
OUT	pid	process id in the group of comm (integer)

MPI_SEND(buf, count, datatype, dest, tag, comm)
Send a message.

IN	buf	address of send buffer (choice)
IN	count	number of elements to send (integer $\geq$0)
IN	datatype	datatype of send buffer elements (handle)
IN	dest	process id of destination process (integer)
IN	tag	message tag (integer)
IN	comm	communicator (handle)

MPI_RECV(buf, count, datatype, source, tag, comm, status)
Receive a message.

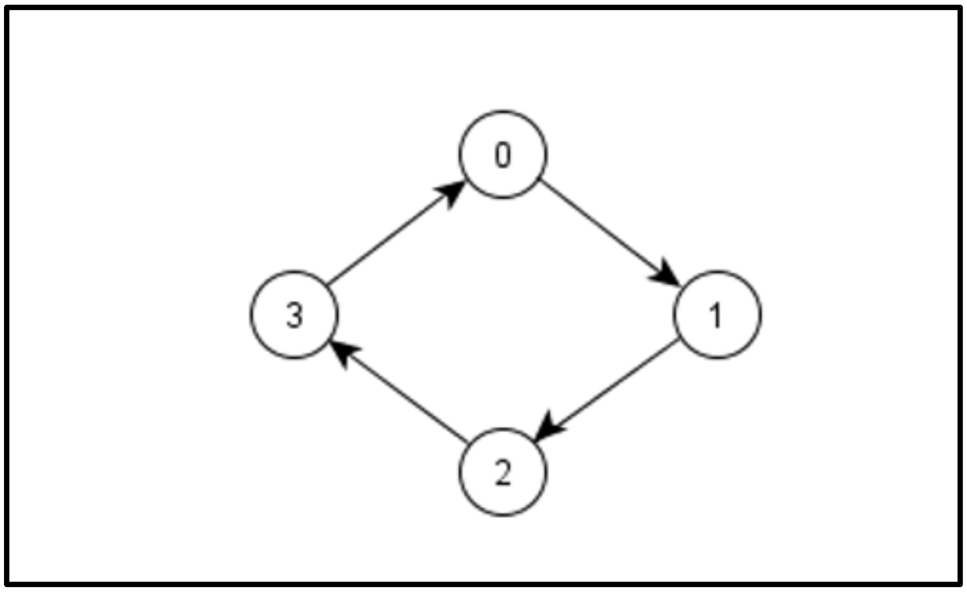
OUT	buf	address of receive buffer (choice)
IN	count	size of receive buffer, in elements (integer $\geq$0)
IN	datatype	datatype of receive buffer elements (handle)
IN	source	process id of source process, or **MPI_ANY_SOURCE** (integer)
IN	tag	message tag, or **MPI_ANY_TAG** (integer)
IN	comm	communicator (handle)
OUT	status	status object (status)

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

RING TOPOLOGY ALGORITHM TO COMPUTE_SUM EXPLAINED:

A linear array is a static network in which each node (except the node at the ends), has two neighbors, one each to its left and right. The extension of a linear array is a ring. It has a wraparound connection between the extremities of the linear array. In this case each node has two neighbours .One-One data distribution has been used in the ring topogy. This means that One processor(rank) sends the array to the processor next to it in the ring(rank+1).Once all the processors have the array , they will add the numbers on the array indices .

The partial sum which each processor has will be sent to the next processor in the ring. This processor which receives the partial sum from the previous node, will add its own partial sum and then send the new partial sum to the next processor. Since this is a ring topology, at the end of one cycle, the 0th processor will have the computation sum from all the processors.

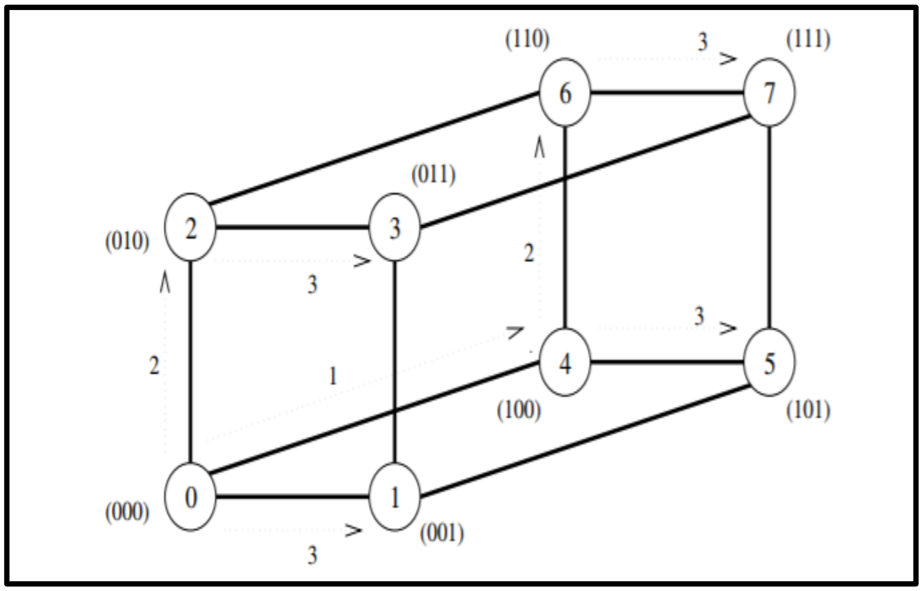


Ring Topology

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

HYPERCUBE ALGORITHM TO COMPUTE_SUM EXPLAINED:

One to all Broadcast algorithm has been used in the Hypercube architecture with cut-through routing. In this algorithm, a single processor is required to send identical data to all other processors or a subset of them. A hypercube with 2^d nodes can be regarded as a d-dimensional mesh with two nodes in each dimension. This process is now carried out in d steps- one in each direction. A 3-d hypercube is shown below:

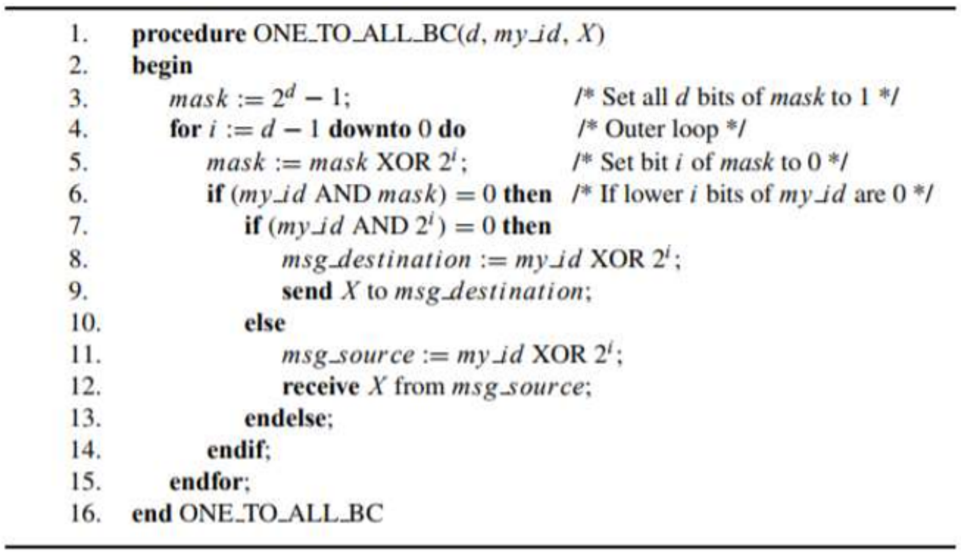


Example of communication of message among the processor in hypercube

The 0th Node has the array containing the N numbers. Here, the processors can send data to one other processor at a time. The 0th processor will first send the array to the 4th processor of the hypercube. The 4th processor also contains the data, thus it is eligible to send data to one other processor.

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

The following figure is the algorithm used to send data on the hypercube.



```
procedure ONE_TO_ALL_BC($d$, $my\_id$, $X$)
begin
    $mask := 2^d - 1$;                       /* Set all $d$ bits of $mask$ to 1 */
    for $i := d - 1$ downto 0 do             /* Outer loop */
        $mask := mask$ XOR $2^i$;            /* Set bit $i$ of $mask$ to 0 */
        if ($my\_id$ AND $mask$) = 0 then   /* If lower $i$ bits of $my\_id$ are 0 */
            if ($my\_id$ AND $2^i$) = 0 then
                $msg\_destination := my\_id$ XOR $2^i$;
                send $X$ to $msg\_destination$;
            else
                $msg\_source := my\_id$ XOR $2^i$;
                receive $X$ from $msg\_source$;
            endelse;
        endif;
    endfor;
end ONE_TO_ALL_BC
```

The basic principal of this algorithm is based on a mask variable. The mask variable contains a number that is equal to (2d - 1). When this number is converted into binary we see that it only contains 1s. Our

algorithm takes advantage of this property and does some calculations to get a 0 on the ith bit of the mask variable. Once we get this we do an AND with 2i to know whether we have to send the data or receive it. Then we do an XOR of the rank of the processor and 2i to know where to send or receive the data. This part of algorithm will be used to find the communication time between the processsors.

Next algorithm to find the sum is called **SINGLE_NODE_ACCUMULATION** in hypercube with d dimension is as follow:

```
procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
begin
   for j := 0 to m - 1 do sum[j] := X[j];
   mask := 0;
   for i := 0 to d - 1 do
     /* Select nodes whose lower i bits are 0 */
     if (my_id AND mask) = 0 then
        if (my_id AND 2i) $\neq$ 0 then
           msg_destination := my_id XOR 2i;
           send sum to msg_destination;
        else
```

```
11.             msg_source := my_id XOR 2i;
12.             receive X from msg_source;
13.             for j := 0 to m - 1 do
14.                 sum[j] :=sum[j] + X[j];
15.         endelse;
16.       mask := mask XOR 2i; /* Set bit i of mask to 1 */
17.   endfor;
18. end ALL_TO_ONE_REDUCE
```

RESULTS FOR RING FORMULATION :

1. **Parameter Range**: k=1, p=10 where k is no of logical processors (or cores) and p=no of physical processors per core. Total=10 processors

Array Size	Computation Time (in seconds)	Communication Time (in seconds)	Sum of the array
1000	0.001537812630	0.000028	4545
10,000	0.001537812703	0.000031	44901
50,000	0.001537812799	0.000046	224357
100,000	0.001537812881	0.000054	448706
200,000	0.001537812926	0.000087	899337

Table 1

2. **Parameter Range**: k=2, p=10 where k is no of logical processors (or cores) and p=no of physical processors per core. Total=20 processors

Array Size	Computation Time (in seconds)	Communication Time (in seconds)	Sum of the array
1000	0.001537813552	0.000574	4545
10,000	0.001537813507	0.008605	44901
50,000	0.001537813384	0.000588	224357
100,000	0.001537813316	0.008090	448706
200,000	0.001537813199	0.005522	899337

Table 2

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

3. **Parameter Range**: k=4, p=8 where k is no of logical processors (or cores) and p=no of physical processors per core. Total=32 processors

Array Size	Computation Time (in seconds)	Communication Time (in seconds)	Sum of the array
1000	0.001537813693	0.000031	4518
10,000	0.001537813751	0.000032	44846
50,000	0.001537813837	0.000034	224286
100,000	0.001537813885	0.000038	448706
200,000	0.001537813927	0.00004	899337

Table 3

GRAPHS FOR RING FORMULATIONS

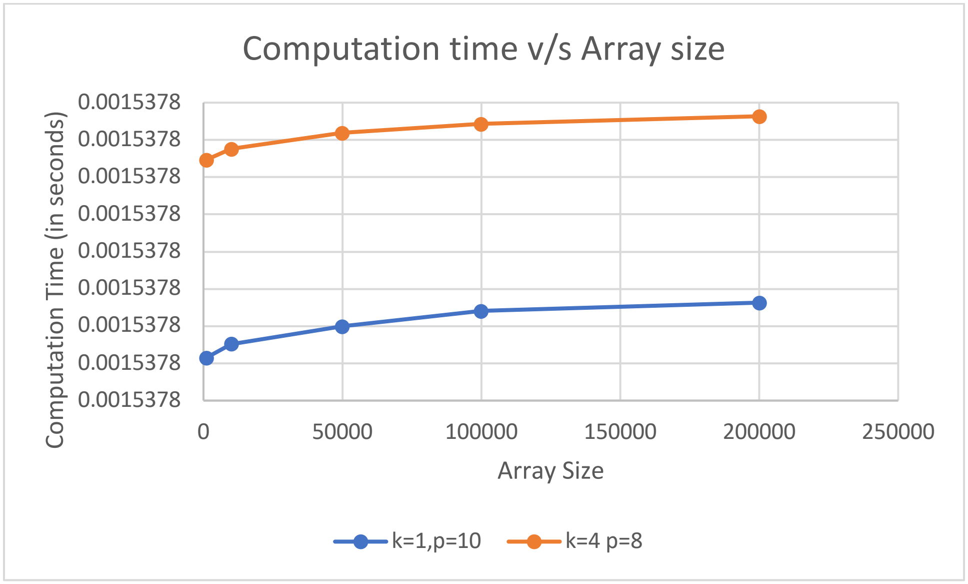


Figure 1

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

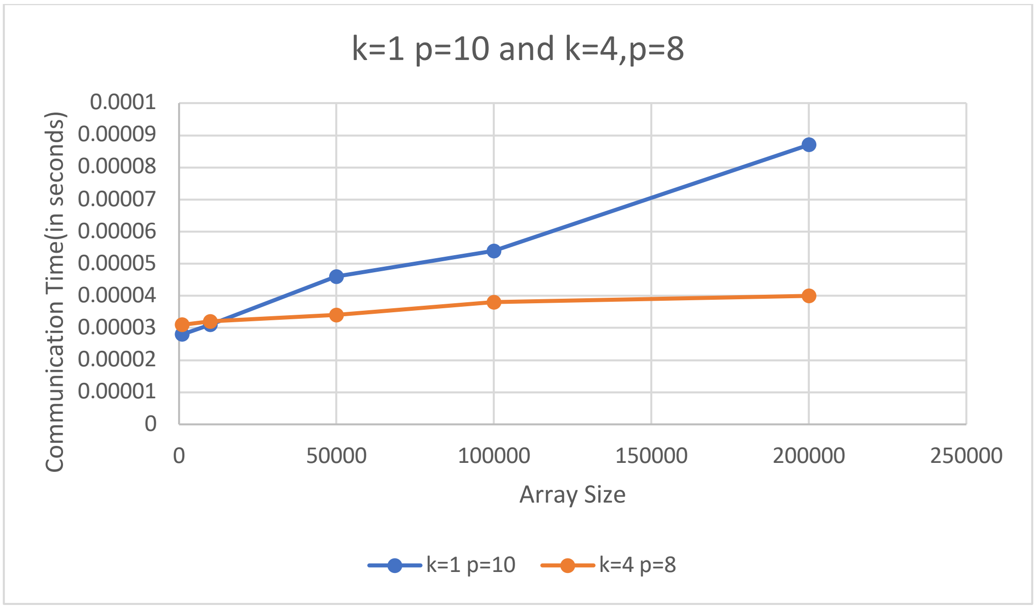


Figure 2

<u>ANALYSIS FOR RING FORMULATION w.r.t COMMUNICATION & COMPUTATION TIME OVERHEAD:</u>

1. From figure 1, it can be observed that the computation time increases with the increase in the array size keeping the number of parameter range constant for each array size. This is because, when we increase the number of array size, the load on each processor increases proportionally and thus the time taken to compute the sum also increases.
2. Also, it can observed from figure 1 that for two different parameter ranges like k=1,p=10 and k=4,p=8, as the total number of processors increase the overall communication between the processor increases and thus the computation time increases for two different ranges.
3. Major reason for computation time being different for k=4, p=8 is increase in the number of cores. Since it is 4 for second observation and k=1 for first there is a significant difference in computation time.
4. From Figure 2, as the total number of processors increases for instance k=4, p=8 i.e. total processors are 32 as compared to 10(k=1, p=10) processors, the communication time decreases. This is because, more the number of processors, less the computation and since we normalize the numbers of array to each processor the communication time decreases. But for different array size in the same parameter range, the time to communicate increases gradually.

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

RESULTS FOR HYPERCUBE FORMULATION

Note: To run hypercube.c run the following command:
mpicc -o hypercube hypercube.c -lm

1. **Parameter Range:** k=1, p=8 here k is no of logical processors (or cores) and p=no of physical processors per core. Total=8 processors

Array Size	Computation Time (in seconds)	Communication Time (in seconds)	Sum
16	0.000073	0.01537923379997139	74
32	0.000072	0.01537923519143266	143
64	0.000044	0.01537923619769989	280
128	0.000068	0.01537923784478911	581
256	0.000062	0.01537923867714868	1204
1024	0.000057	0.01537924019357774	4643
4096	0.000074	0.01537924068541104	18494
8192	0.000112	0.01537924147058831	36955
16384	0.000114	0.01537924204808870	73549

Table 4

2. **Parameter Range:** k=2, p=4 here k is no of logical processors (or cores) and p=no of physical processors per core. Total=8 processors

Array Size	Computation Time (in seconds)	Communication Time (in seconds)	Sum
16	0.000066	0.01537924724209603	74
32	0.000075	0.01537924780313821	143
64	0.000079	0.01537924832879336	280
128	0.000065	0.01537924899653819	581
256	0.000099	0.01537924959517256	1204
1024	0.000080	0.01537925027682191	4643
4096	0.000109	0.01537925076621595	18494
8192	0.000151	0.0153792513641884	36955
16384	0.000410	0.01537925208728355	73549

Table 5

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

GRAPHS FOR HYPERCUBE FORMULATIONS

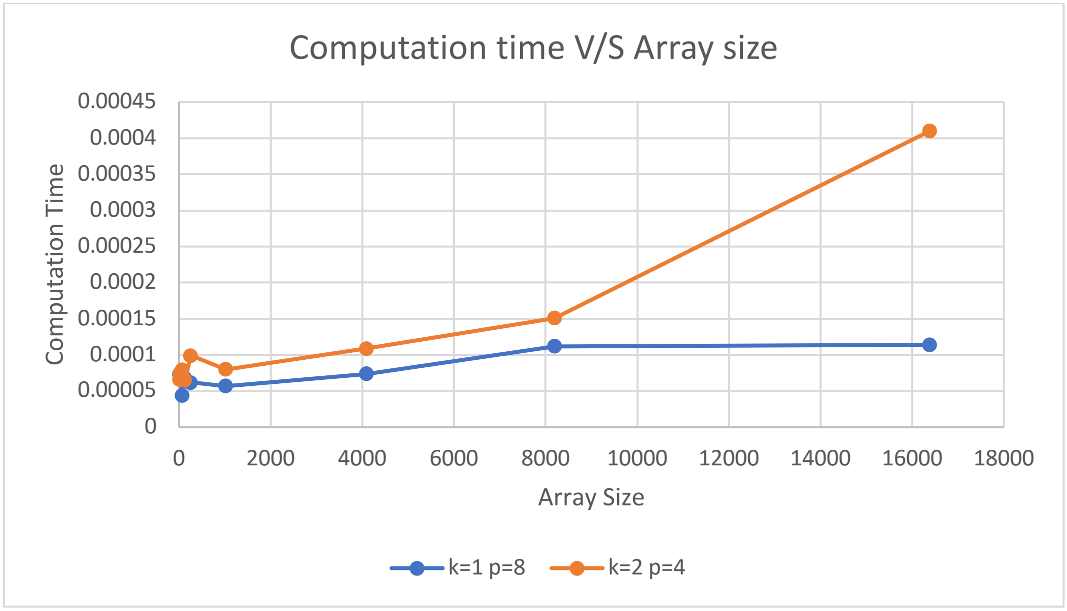


Figure 3

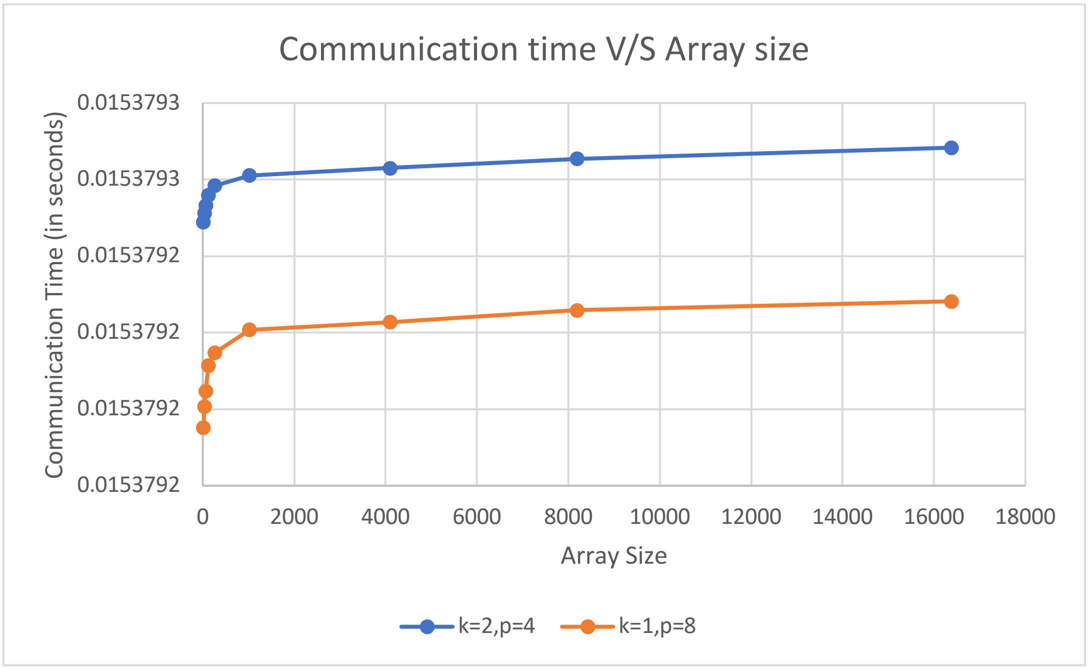


Figure 4

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

ANALYSIS FOR HYPERCUBE FORMULATION w.r.t COMMUNICATION & COMPUTATION TIME:

1. It is observed from the figure 3 that when the array size is small (e.g., 16,32,64,128, etc) there is not much difference between the computation time of the hypercube with 8 processors and that with 8 processors with 2 cores. As, the computation time is made up of the time to process sums on each processor (average time across all the processors as they are run parallelly) plus the time required to communicate between the processors. When the array size is small, both the times compensate each other, and we get almost equal computation time for both the hypercubes. So, we can say that as the array size increases the computation time also increases for a particular range of configurations in the hypercube.
2. Most importantly, from figure 4, we can observe that communication time is almost constant for the increasing array size for both the configurations of the hypercube. This is because in a hypercube the data is not send or received in every iteration for every processor. So, on an average the communication times comes out to be equal.

COMPARISON BETWEEN THE TWO FORMULATIONS:

1. As observed from the result tables of hypercube formulations and ring formulations, it is evident that the computation time decreases for the hypercube with the increasing array size. This is because, as the dimension of the hypercube is increasing, the routing of messages in the topology from each processor to d-connected processor increases. Thus, the load gets distributed evenly and eventually the computation time decreases,
2. The ring formulations performs better for performing sum in terms of communication time as evident from figure 2 when the parameter is changed to 4 cores and 8 processor to each core. Since, in ring topology each node is connected to its succeeding processor, the communication becomes faster by sending the local sum to the next set of processors.
3. Hypercube formulation has the most important advantage of it being highly scalable, increasing the number of dimension directly increases the chances of routing the messages through the processors easily. Also, even if one of the processors fails then the entire communication do not stop whereas this is not the case for ring topology.

Chirag Soni: 665930262
Sanamdeep Singh Anand: 677787570

LESSONS LEARNT:

1. In this coding assignment, we have learned different topologies which are widely used mainly: Ring and Hypercube.
2. Different message routing algorithms for each topology was theoretically understood and practically implemented such as **One to All Broadcast for communicating the local sums**, and **Single Node accumulation for adding the local sums to the source processor**.
3. We have learned about the different aspects and condition where computation and the communication time varies and behaves differently for different formulations and parameter ranges and array size.
4. We learnt to formulate the problem using different approaches. We tried various approaches but mainly we used the method where each processor is mapped to a chunk of numbers so as to decrease the latency in the routing mechanism.