Arunima Singh Thakur, 180905218, C, 31, CSE, PCAP Assignment 1, Arun.

---

1)

CPU Design principle :-

- The design of a CPU is optimized for sequential code performance.
- It makes use of sophisticated control logic to allow instructions from a single thread of execution to execute in parallel.
- The large cache memories are provided to ~~reduce~~ reduce the instruction ~~and~~ and data access latencies of large complex applications → latency-oriented design
- Memory bandwidth limit the speed of applications by limiting the rate at which data can be ~~d~~ delivered from the memory system to processors.

whereas GPU design principle :-

- The design philosophy of the GPUs is shaped by the fast growing video game industry, which requires the ability to perform a massive number of ~~fl~~ floating-point calculations per video frame.
- GPU performs well by ~~ex~~ executing massive numbers of threads.
- GPU hardware takes advantage of a large number of execution threads to find work to do when some of them are waiting for long-latency memory accesses.
- Small cache memories are provided to help the bandwidth requirements of applications, so multiple threads that access the same memory

data need not always access the DRAM →
throughput - oriented design

- Most apps will use both CPUs & GPUs, executing the sequential parts on the CPU & numerically intensive parts on the GPUs.

---

2)

```
#include<stdlib.h>
#include<stdio.h>
#include<mpi.h>
int main (int argc, char *argv[])
{
    int size, rank, mat[4][4];
    int *arr; int n, crr[4];
    mpi_init(&argc, &argv);
    mpi_Comm_rank(mpi_COMM_WORLD, &rank);
    mpi_Comm_size(mpi_comm_WORLD, &size);
    if (rank==0)
    {
        printf("Enter size of 1D array: ");
        scanf("%d", &n);
        arr = malloc(n* sizeof(int));
        printf("Enter value for 1D array:\n");
        for(int i=0; i<n; i++)
            scanf("%d", &arr[i]);
        printf("Enter value for 2D matrix:\n");
        for (int i=0; i<4; i++)
          for (int j=0; j<4; j++)
             scanf("%d", &mat[i][j]);
    }
    n = n/4;
    int brr[4];
    ~~int main~~
```

```
MPI_Scatter(arr, n, MPI_INT, brr, n, MPI_INT, 0,
            MPI_COMM_WORLD);
int min = brr[0];
for(int i=1; i<n; i++)
    if(min > brr[i])
        min = brr[i];
MPI_Scatter(mat, 4, MPI_INT, crr, 4, MPI_INT,
            0, MPI_COMM_WORLD);
int max = crr[0];
for(int i=1; i<4; i++)
    if(max < crr[i])
        max = crr[i];
printf("Rank = %d, Minimum = %d", rank, min);
printf("Minimum in array + Maximum in
      column %d in rank %d = %d",
      rank, rank, (min + max));
MPI_Finalize();
return 0;
}
```