Assignment No. B4

Roll No:

• Title
To implement ODD-EVEN sort using CUDA

• Problem Definition
Implement a Parallel ODD-Even Sort algorithm using GPU or ARM equivalent.

• Learning Objective :
To study Parallel ODD-Even Sort algorithm using GPU or ARM equivalent.

• Learning Outcomes
Successfully implemented ODD-Even sort using CUDA.

• Software and Hardware Requirement

1. CUDA enabled machine
2. NVIDIA GEForce Graphics card
3. CUDA and nsight IDE

• Theory :
1.Introduction:
Parallelism on chip level is the hub for advancements in micro processor architectures for high performance computing. The core-processors, in personal computers, were not sufficient for high data- computation intensive tasks. As a result modular and specialized hardware in the form of sound cards or graphic accelerators are increasingly present in most personal computers. Graphics cards or graphics processing units (GPU), introduced primarily for high-end gaming requiring high resolution.
The GPU itself is a multi-core processor having support for thousands of threads running concurrently. GPU's are result of dozens of streaming processors with hundreds of core aligned in a particular way forming a single hardware unit. Performance evaluation in GFLOPS (Giga Floating Point Operations per Second) shows that GPU's outperforms their CPU counterparts. For example: a high-end Core I7 processor (3.46 GHz) delivers up to a peak of 55.36 GFLOPs1.

2. Concept use in programming:
i.Parallel sorting algorithms: Sorting on GPU require transferring data from main memory to on-board GPU global memory. Although on-device bandwidth is in the range of 144Gb/s, thus only those sorting techniques are efficient which require minimum amount of synchronization because the PCI bandwidth is to the range of 2.5Gb/s. i.e., synchronization and memory transfers between CPU and GPU will affect system performance adversely. Compared to serial sorting algorithms, parallel algorithms are designed requiring high data independence between various elements for achieving better performance. Those techniques which involve large data dependency are categorized as sequential sorting algorithms.

ii.Odd-Even Sort:
The odd-even sort is a parallel sorting algorithm and is based on bubble-sort technique. Adjacent pairs of items in an array are exchanged if they are found to be out of order. What makes the technique distinct from bubble-sort is the technique of working on disjointed pairs, i.e., by using alternating pairs of odd-even and even-odd elements of the array. The technique works in multiple passes on a queue Q of size N. In each pass, elements at odd-numbered positions perform a comparison check based on bubble-sort, after which elements at even- numbered positions do the same. The maximum number of iterations or passes for odd-even sort is N/2. Total running time for this technique is (log2)) . The algorithm works as:-

• Mathematical Model:

Let S is the solution for given problem as

$$\mathbf{S = \{\ s,i,o,f,dd,ndd,success,failure\}}$$

where,

s=Initial state of system =Intialistion of data variables

i=Input to the system={x | x belongs to 1,2,3,....}

o=Output given by system={y| y belongs to 1,2,3,...}

f=Transition function

f(i is even):compare-and-exchange-min(id+1);

f(i is odd):compare-and-exchange-min(id-1);

where id:process label

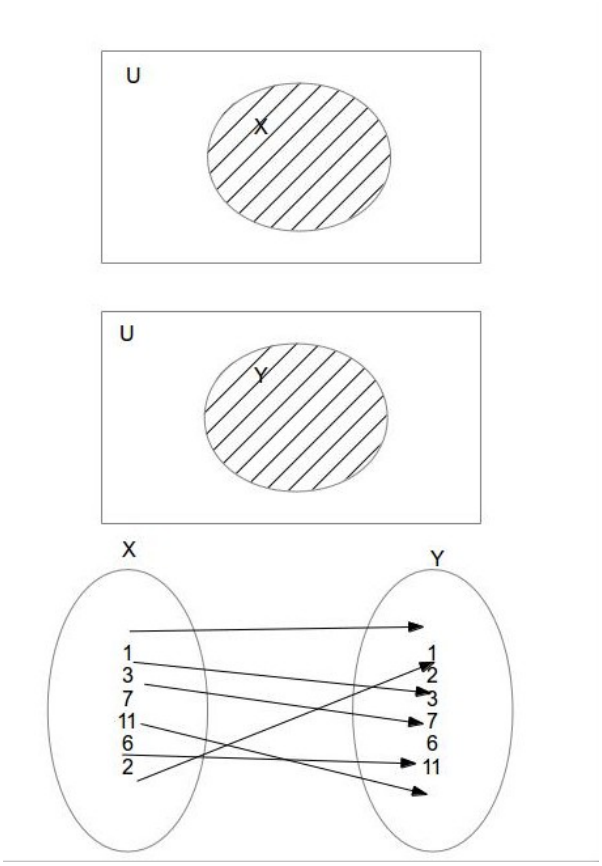
dd=deterministic data it helps identifying the load store functions or assignment functions.

ndd=Non deterministic data of the system S to be solved

Success=desired outcome generated=sorted array elements

Failure=Desired outcome not generated or forced exit due to system error.

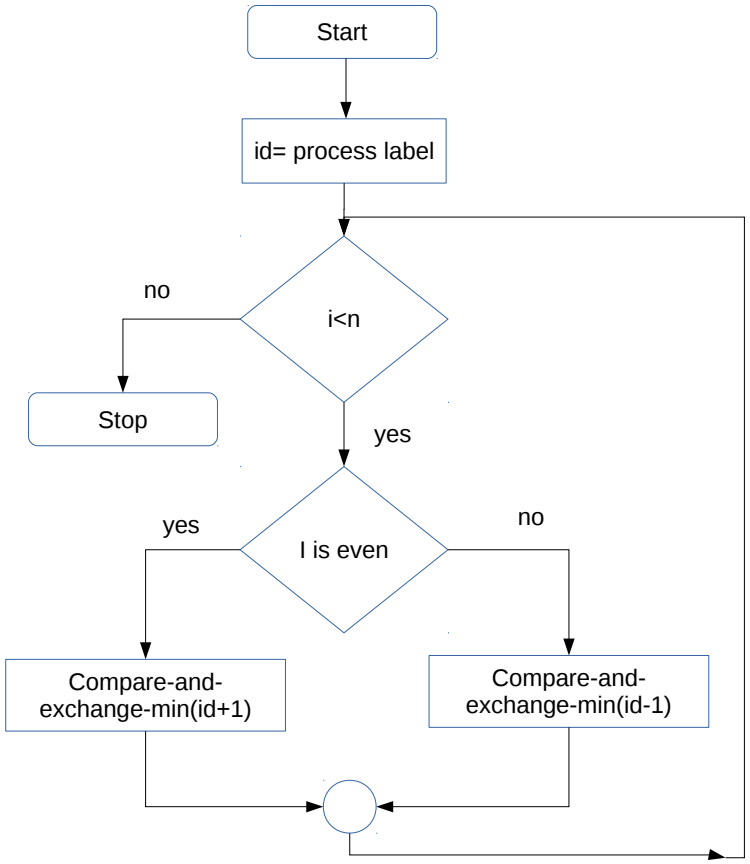
• Venn Diagram:



- **Algorithm:**

```
void ODD-EVEN(n)
    {
        id = process label
        for (i= 1; i¡= n; i++)
         {
              if (i is odd)
                  compare-and-exchange-min(id+1);
              else
                  compare-and-exchange-max(id-1);
              if (i is even)
                  compare-and-exchange-min(id+1);
              else
                  compare-and-exchange-max(id-1);
        }
    }
```

- **Flowchart:**



- **UML Diagrams:**

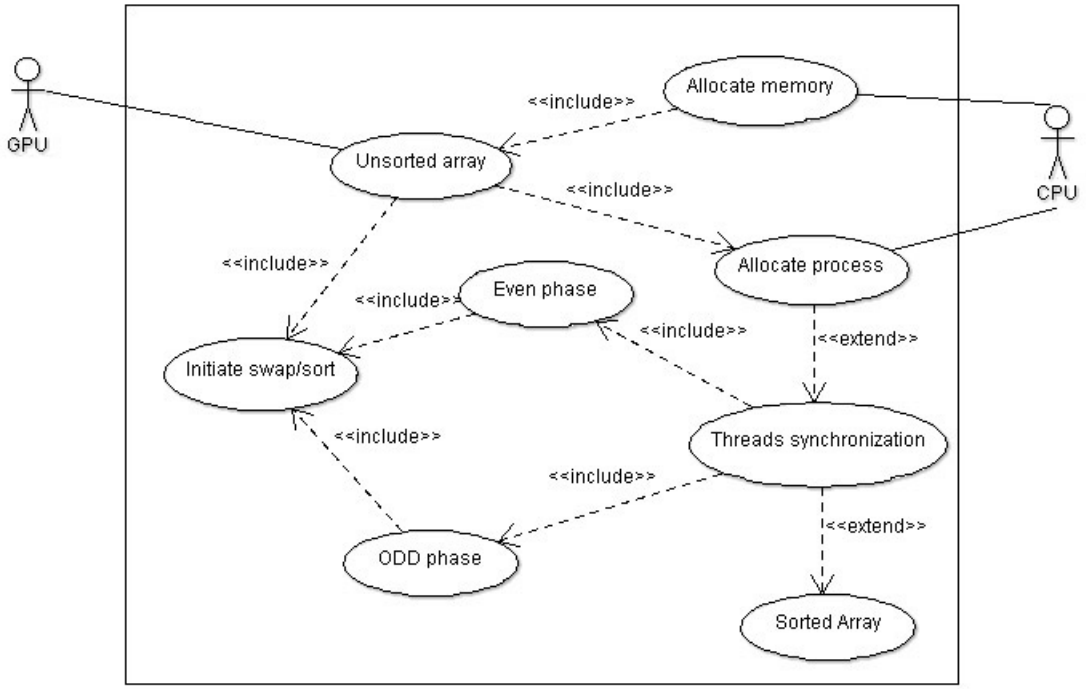


Figure 1: Use case Diagram

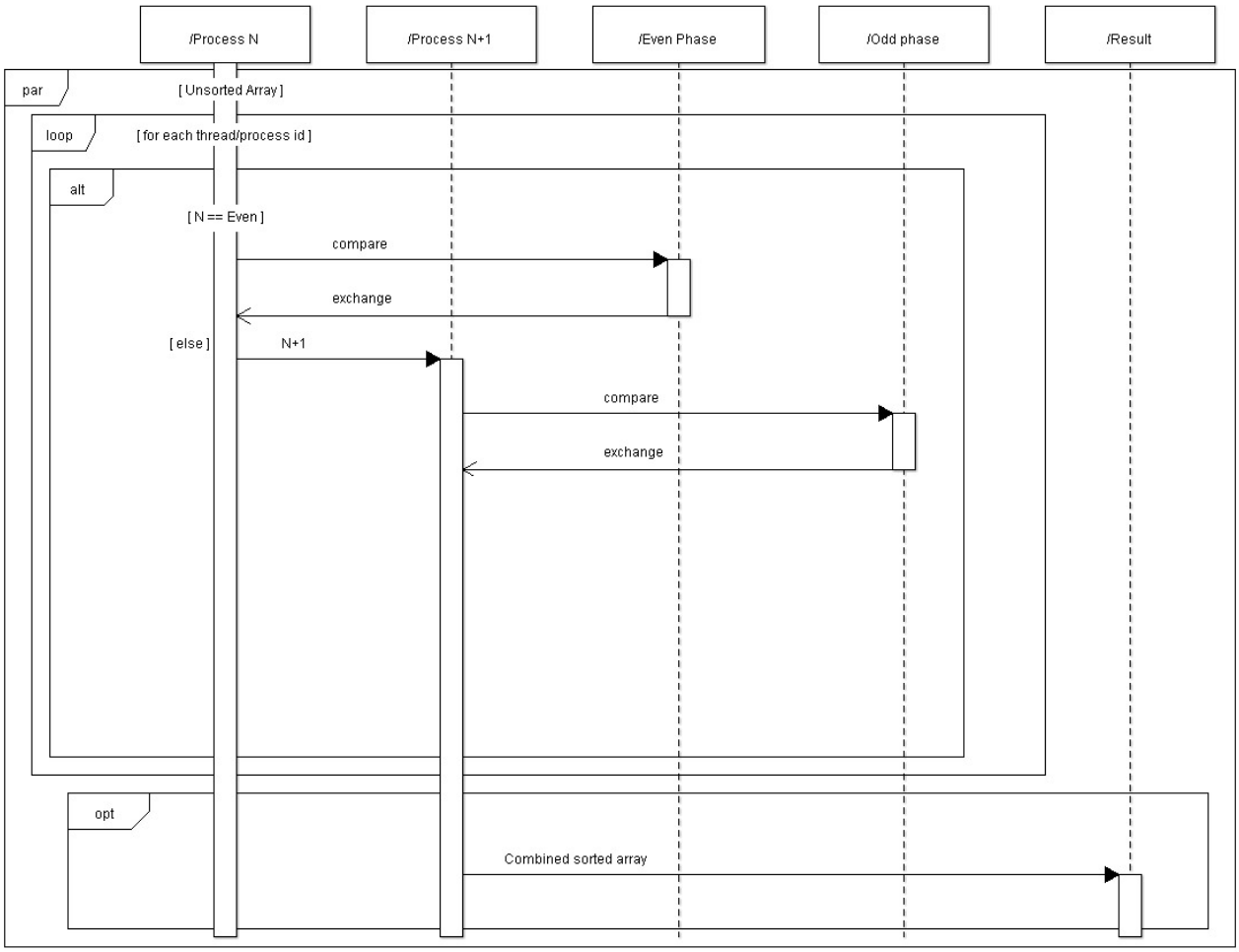


Figure 2: Sequence Diagram

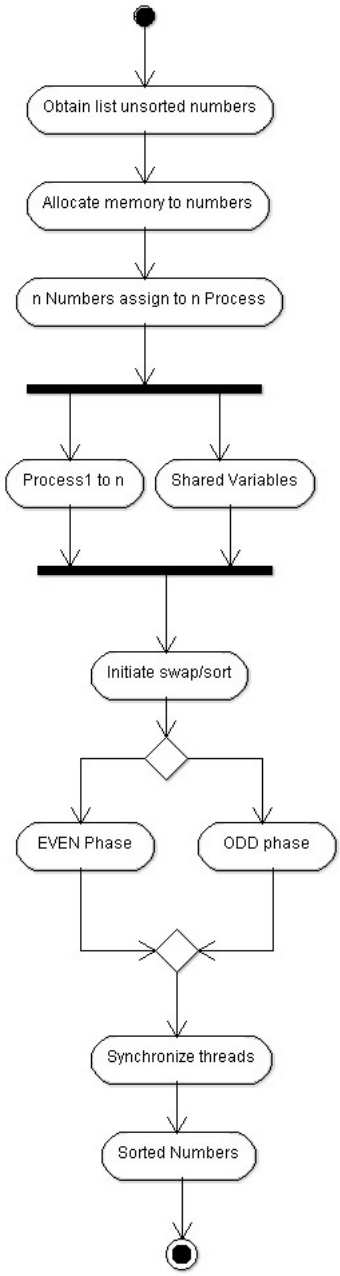


Figure 3: Activity Diagram

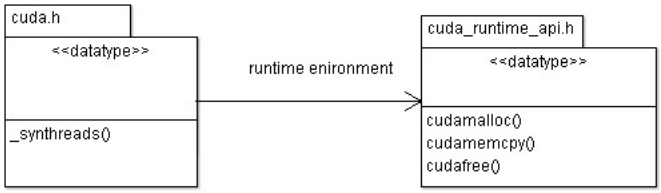


Figure 4: Class Diagram

- **Conclusion :**

Hence we have studied a Parallel ODD-Even Sort algorithm using GPU or ARM equivalent.

- **Input:**

```
12
62
45
95
65
59
25
75
```

- **Output:**

```
12 25 45 59 62 65 75 95
```

Course Outcomes	Achieved Outcome (Tick √)
CO I : Ability to perform multi-core, Concurrent and Distributed Programming.	
CO II : Ability to perform Embedded Operating Systems Programming using Beagle-bone.	
CO III : Ability to write Software Engineering Document.	√
CO IV : Ability to perform Concurrent Programming using GPU	√

- **FAQs**

Q1. What is parallelism?
Q2. How Parallelism is achieved?
Q3. How algorithm works in this problem?
Q4. What is kernel function for this problem statement?
Q5. What is device code and host code?
Q6. What is difference between GPU and CPU?
Q7. How CPU allocates memory to variables?