Page No.
Date

NAME: AENAN ATTAR PRN :- F19112003 CLASS : BE COMP II
SUBJECT :- HPC ASSIGNMENT No. :- 01

Q.1) How is CUDA programming useful to study parallel algorithm?

Ans 1. CUDA is a parallel computing platform and programming model developed by NVIDA for general computing on its own GPUs (graphics processing units).

2. CUDA enables developers to speed-up compute-intensive applications by harnessing the power of GPU for the parallelizable part of computations.

3. CUDA stands for Compute Unified Device Architecture.

4. While there have been other proposed APIs for GPUs, such as OpenCL, and there are competitive GPUs from other companies, such as AMD, the combination of NVIDIA and CUDA based GPUs domainates several domains such as deep learning and is the foundation for some of the fastest computers in the world.

5. The CUDA programming model lets programmers write scalable parallel programs using straightforward extension of C and also guides the programmers to expose substantial fine-grained parallelism sufficient for utilizing massively multi-threaded GPUs.

Q2) Discuss the importance of parallel reduction operations.

Ans 1. Reduction operators is a type of operator that is commonly used in parallel programming to reduce elements of an array into single result.

2. These operations are always associative but may or may not be commutative.

3. Reduction operators are associative and an integral part

of programming models such as Map Reduce, where a reduction operator is applied to all element before they are reduced.

4. Other parallel algorithms use reduction operators as primary operations to solve a complex problem.
5. Reduction is one of the main collective operations implemented in the message passing interface, where performance of used algorithm is important and evaluated constantly for different use cases.
6. Some parallel sorting algorithms use reduction to handle very big data sets.

Q3) Discuss the operations on vectors and the approach for parallel design for the same.

Ans Let us see the operations on vectors and the approaches for parallel design for them:-

I. Addition & Subtraction:-
- For vector addition and subtraction, vectors can be divided into sub-vectors and the addition/subtraction of these vectors can be parallelized.
- Results of the operations on sub-vector can be combined at the end.

II. Scalar multiplication :-
- Here each element of the vector can be multiplied by scalar parallelly.

III. Dot product:-
- Here we use parallel reduction algorithms, and divide vectors into sub-vectors, where each sub-vectors is multiplied in parallel.

IV. Cross product:-

- A common approach for cross product is to use parallel merge algorithm.
- Partial sub-vectors are combined using binary operations to calculate final result.

Q4) How is parallelism achieved in CUDA?

Ans 1. In the CUDA programming model, a group of blocks of threads that are running a kernel is called a grid.
2. In CUDA Dynamic Parallelism, a parent grid launches kernels called as child grids.
3. A child grid inherits certain attributes from parent grid certain attributes and limits such as L1 cache and stack size.
4. Every thread that encounters a kernel launch executes it.
5. Hence under dynamic parallelism one kernel may launch another kernel, and that kernel may launch another and so on.
6. Each sub-ordinate launch is considered a new "nesting level" and the total number of levels is the "nesting depth" of the program.
7. Therefore threads launch new grid according to amount of work that is newly discovered.

Q5) Explain Grid, Block and Thread structure in relation with parallel reduction.

Ans 1. In CUDA programming model, the parallel reduction operation involves dividing an input array into blocks and threads where each thread performs a certain computation on a subset of the data.

2. The grid refers to the entire collection of blocks that make up the computation, where, programmer specifies the number of blocks in the grid, and hardware divides grid into groupes of threads.
3. Block is a subset of grid that contains a groups of threads, which is specified by the programmer that typically correspond to number of processing elements in the GPU.
4. A thread is a unit of execution that runs on a processing element within a block.
5. In context of parallel reduction, each thread performs a computation on a subset of input data and stores result in shared memory.
6. Once all the threads in a block have completed their computations, a single thread in the block performs a reduction operation to combine intermediate results into single value.

NAME:- AENAN ATTAR PRN:- F19112003 CLASS:- BE COMP 2
SUBJECT:- HPC ASSIGNMENT NO.:- 02

Q1) How to parallelize addition of two large vectors?

Ans 1. Domain decomposition also called as data decomposition to sum vectors parallelly.

2. We divide the array/vector into equal parts and assign each part to a processor.

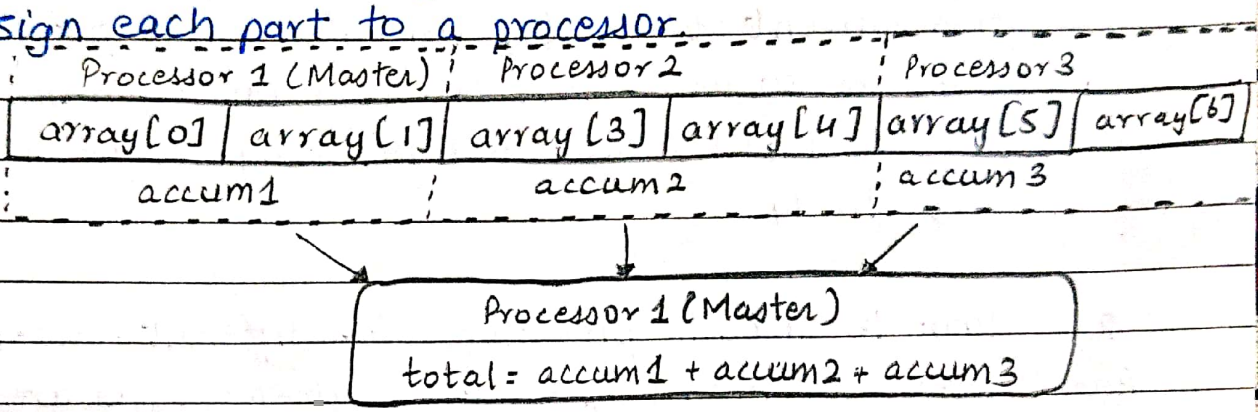


Fig:- Parrallel Addition of Vectors

3. As shown above, we use three processors where one processor acts as master processor.

4. Work is divided equally and each accumalated addition of a sub-array is added to get the final answer.

5. Dividing the process is the first phase and accumalating the additions, second.

Q2) Comment on data dependency in matrix and vector multiplication.

Ans

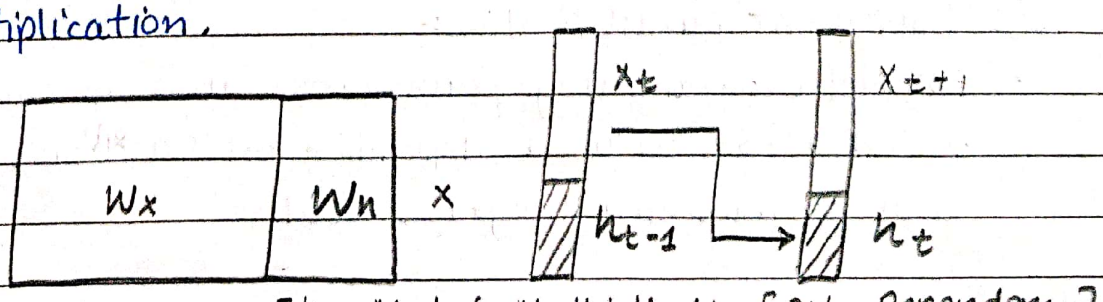


Fig: Matrix Multiplication [Data Dependency]

1. Conventional matrix-vector multiplication involves the entire vector of $(x_t, h_{t-1})$ and a whole row of the weights matrix at a time.
2. However additional stalling is introduced since system needs to wait for additional computed hidden units before starting the next computation.
3. This mainly due to data dependency between output from current time-step and the vector for the next time as shown where $W_x$ and $W_h$ represent the weights for the input vector and the weights for hidden vector respectively.

Q3) How to allocate processors for different sub-problems?

Ans Steps to allocate processors sub-problems are as follows:-

I) Identify the sub-problems:-
- One must identify the sub-problems that can be executed in parallel.
- Sub-problem created must be independent of other sub-problems.

II) Determine the number of processors:-
- Number of cores or number of nodes in a distributed computing system.

III) Divide Workload:-
- Workload can be divided by strategies such as data, task and pipeline parallelism.

IV) Implement parallelization:-
- Parallel programming frameworks such as OpenMP, MPI or CUDA can be used depending on hardware architecture and programming language used.

Page No.
Date

Q4) Explain concept of multi-threading in OpenMP with an example.

Ans 1. OpenMP is an implementation of multi-threading, a method of parallelizing whereby a master thread forks a specific number of slave threads and the system divides a task among them.

2. Threads run concurrently and the runtime environment allocates threads to processors.

3. Example:-

- Consider a C program:- omp_simple.c

→
```
#include<stdio.h>
int main(int argc, char* argv[]) {
    #pragma omp parallel
        printf("This is a thread\n");
        return 0;
    }
```

- To execute run:

→ gcc -fopenmp omp_simple.c

- Output produced:-

→ This is a thread
This is a thread
:
:

- Output depends on number of threads.

NAME :- AFNAN ATTAR PRN :- F19112003 CLASS :- BE COMP II
SUBJECT :- HPC ASSIGNMENT NO. :- 03

Q1) Do analysis of bubble sort and find out its time complexity.

Ans 1. Let us see the algorithm of bubble sort :-

```
Begin BubbleSortAlgorithm(Array):
    For all the elements of the array
        if array[i] > array[i+1]
            switch(array[i], array[i+1])
        endif
    return array
End BubbleSort Algorithm
```

2. Time Complexity :-
- Bubble sort uses two loops one inner and one outer.
- Inner loop performs O(n) comparisions.
- In worst case the outer loop runs O(n) times.
- As a result, the worst case time complexity is $O(n \times n) = O(n^2)$

Q2) Write difference between parallel approach bubble sort and merge sort.

Ans. Parallel Bubble Sort :-
1. Bubble sort repeatedly steps through the list compares adjacent elements and swaps them if they are in the wrong order.
2. Parallel version of bubble sort distributes the sorting process across multiple processor where each processor works on a subset of data.
3. However it is not efficient due to its high communication overhead and poor scalibility.

- Parallel Merge Sort:-

1. Merge Sort uses divide & conquer approach to sort a list.
2. In its parallel version each processor is assigned a subset of data and sorted subsets are merged in a parallel manner.
3. It is highly efficient and scalable.

- The main difference is that parallel merge sort is better in performance and scalibity compared to parallel bubble sort.

Q3. Comment on scaling parallel merge sort.

Ans

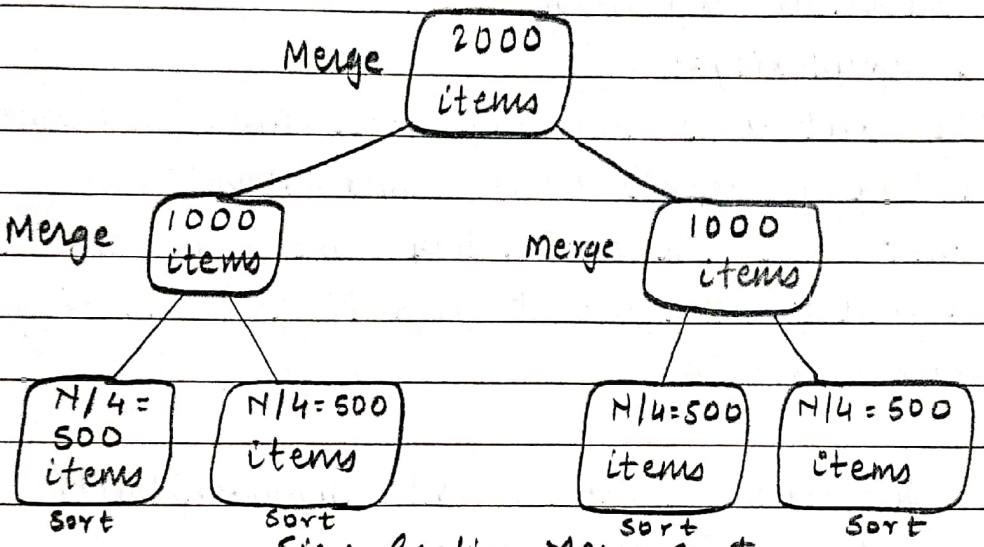


Fig: Scaling Merge Sort

1. If we have far more elements then the number of elements to be sorted, we need to make our parallel merge sort algorithm parallel.
2. Given the number of processors to use, n, we start by using a binary tree that has number of leaves as n.
3. Each of the n nodes will first sort N/n of the original input data values, using algorithms such as quick sort.

4. Two sorted lists can be used by 'parent' process that will merge them.
5. That newly sorted list can be used by another parent process and merged with a second child sorted list.
6. This process is repeated until the two sublists get merged together.

Q4) How can bubble sort algorithm be parallelized?

Ans 1. To make bubble sort work parallelly we divide sorting of the unsorting into two phases:- odd and even.
2. We compare all pairs of elements in the list/array side by side (parallel).
3. When the sorting is in odd phase we compare element at index 0 with the element at index 1, the element at index 2 with index 3 and so on.
4. In even phase we compare index 1 element with index 2 element, index 3 element with index 4.
5. While comparing we simply swap the element if the initial element is greater than next element in comparision.
6. Both phases occur one after the other and the algorithm stops if no swap occurs.
7. In the case of multi-processor systems both phases can occur simultaneously.

NAME:- AFNAN ATTAR PRN:- F19112003 CLASS:- BE COMP II
SUBJECT:- HPC ASSIGNMENT No.:- 04

Q1) Do the analysis of parallel bubble sort and find out its time complexity.

Ans 1. Bubble Sort Algorithm:-

```
Begin BubbleSortAlgorithm(Array):
    For all the elements of the array
        if array[i] > array[i+1]
            switch(array[i], array[i+1])
        endif
    return array
End BubbleSortAlgorithm
```

2. Time Complexity analysis:-

- Bubble Sort uses two loops: inner and outer.
- Inner loop performs O(n) comparisions.
- In worst case outer loop may perform O(n) operations.
- Time Complexity = $O(n \times n) = O(n^2)$.

Q2) Discuss parallel approach for tree and graph traversal algorithms.

Ans Some parallel approaches for tree and graph traversal algorithms are as follows:-

I) Divide & Conquer:-

- Tree/Graph is divided into sub-graphs that can be traversed independently.
- Each processor is assigned a subset to traverse.

II) BFS/DFS with Work Queue:-

- A work queue is used to assign traversal tasks to multiple processors.

- Each processor retrieves a task from the work queue, traverses the graph and adds a new task to the work queue.
- This is an efficient approach when branching factor is too high.

III) Parallel Recursive DFS :-
- Different branches of recursion tree are assigned to different processors.
- Each processor traverses the assigned branch and the results are combined to get final output.
- Useful when graph is irregular.

IV) Parallel BFS with Graph Partitioning :-
- Graph is partitioned using algorithms such as spectral partioning or metis.
- Each partion is assigned to the processor and traversed independently.

Q3) Under what situations, would the following search algorithms be most appropriate ? [Give Examples]
a. Depth First Search.
b. Breadth First Search.
c. Best First Search.

Ans A] Depth First Search :-
1. Depth First Search or DFS is appropriate when the search space has a high branching factor, and the goal is to find any solution.
2. For example :- Puzzle solving problems such as eight queens or Sudoku, as the state space tree can be quite large and our goal is to find any solution, DFS can be used.

Page No.
Date

B] Breadth First Search :-

1. Breadth first search is appropriate when the state space tree has a low branching factor and our goal is to find shortest path to the solution.
2. Example :- Path finding problems, finding shortest points between two points. BFS explores all possible paths and hence guarantees optimality.

C] Best First Search :-

1. Best First search is appropriate when there is an estimate of the distance from current state to goal state, also called as heuristics.
2. For example :- Finding best route to a destination based on traffic conditions. The route with less traffic should be choosed.