Muhammad Nadeem

201980050

Assignment # 2

Parallel and Distributed Computing

Question:

What is Reduction on the base of What, Why, When, How?

Answer:

What:

In simple terms, reduction is like bringing together a bunch of data and creating a single, meaningful result. Think of it as adding up numbers, finding the biggest or smallest one, calculating averages, or doing other operations that make sense with a group of things.

Why:

Reduction is used to make things faster when we're working on multiple things at the same time. Instead of going through data one by one, which can take a lot of time, we use parallel reduction to deal with different parts of the data all at once. The final answer comes from putting together the bits and pieces each part figured out.

In the world of computers, especially when lots of them are working together, reduction helps sync up and combine the results from each part. It's like teamwork for computers.

When:

We use reduction when we have a bunch of data and want to add it up or combine it in some way. This comes in handy when we're doing big calculations, like in science, data analysis, or simulations where things can get pretty complex.

Whenever we're dealing with huge amounts of data or lots of computers working together, reduction helps speed things up.

How:

There are different ways to do reduction, especially when many computers or processors are involved. Imagine these computers working together like a team, and there are various strategies to make sure they're all on the same page.

The operations we do during reduction are kind of like puzzle pieces fitting together. We use operations that can be done in any order and still give us the same result. For example, adding numbers is like this – whether you add A + B first and then add C, or add B + C first and then add A, you still get the same answer.

Example of Open MP:

```
#include <stdio.h>
#include <omp.h>
int main() {
// Set the number of threads to use (optional)
omp_set_num_threads(4);
// Define an array and initialize it
int size = 1000;
int array[size];
for (int i = 0; i < size; i++) {
        array[i] = i;
```

```
}
// Parallelize a loop using OpenMP
#pragma omp parallel for
for (int i = 0; i < size; i++) {
        // Each iteration of the loop is executed by a different thread
        printf("Thread %d: array[%d] = %d\n", omp_get_thread_num(), i, array[i]);
}
return 0;
}
```