Task 1

Program 2. Sum of all Values in an Array

Pavithra Purushothaman - FD Number: fd0001571

I tried three different approaches for adding the elements of a random array with sizes ranging from 100 to 1,000,000,000: **serial addition, parallel reduction, parallel sections and tasks**. These methods were tested to understand how they perform as the size of the array increases and how efficiently they use the available computing resources. The serial approach was straightforward. It involved looping through the entire array and adding each element to a total. For smaller arrays, this method worked well because the computation was quick and didn’t require any additional setup. However, as the array size grew, the serial approach became slow and impractical. Processing millions of numbers one at a time took a significant amount of time, making it clear that parallel methods, which can split the workload among multiple processors, are necessary for handling larger arrays efficiently.

The parallel methods used different ways to distribute the workload. In the **parallel reduction approach**, the array was divided into equal parts, and each part was processed simultaneously by separate threads. Each thread calculated the sum of its portion of the array, and the results were combined into the sum. This approach was the fastest for large arrays because it used all the available threads effectively, ensuring that no processing power was wasted. The **parallel sections** method divided the array into two halves, with each half processed by a separate thread. While it was simple to implement, it didn’t scale well for larger arrays because only two threads were used, leaving other processors idle. The **tasks approach** also divided the array into two parts but used a more dynamic method of assigning work to threads. While tasks allowed flexibility, they introduced extra management overhead, which made this approach slower than reduction for larger arrays. Among all methods, task approach showed better performance for smaller to medium sized arrays and parallel reduction was efficient, especially for large arrays, because it distributed the workload evenly and minimized the time required to calculate the sum. For smaller arrays, the simplicity of the serial approach made it the most practical choice.

Reference: Used Perplexity AI to understand the terms and to get the syntax for open mp task.