Parallel Computing - PB20000178李笑

This is a demo:

Introduction:

Fast Fourier Transform (FFT) is a widely used algorithm in signal processing, image processing, and other areas of science and engineering. It is used to efficiently calculate the discrete Fourier transform of a signal or a sequence of data points. In this lab, we implemented a parallel FFT using OpenMP, which is a shared-memory parallel programming model that allows us to parallelize the loop iterations in the algorithm.

Code Description:

The code consists of two main parts: the FFT class and the main function. The FFT class contains two member functions: pfft and fft, which perform parallel and sequential FFT, respectively. The class also contains a constructor that initializes the required variables, including the Rader struct and the w vector.

The main function initializes the number of threads and the power of the transform length as command-line arguments. It then creates an instance of the FFT class with the specified transform length and initializes a vector with the same length. It measures the time taken to perform a sequential FFT and a parallel FFT using the initialized vector.

The parallel FFT function (pfft) uses OpenMP parallelism to parallelize the loop iterations in the algorithm. It uses the at function to map the indices to their corresponding positions in the Rader struct. The algorithm then performs the FFT in parallel using OpenMP parallel for directives.

Results and Analysis:

We ran the code with different numbers of threads and transform lengths and recorded the execution time for each case. We then plotted the execution time against the number of threads and transform length to analyze the performance of the parallel FFT algorithm.

We observed that the parallel FFT algorithm achieved a significant speedup over the sequential FFT algorithm. The speedup increased with the number of threads used and the transform length. However, the speedup plateaued at a certain number of threads and transform length, indicating that the algorithm is limited by memory bandwidth and cache size.

Conclusion:

In this lab, we implemented a parallel FFT using OpenMP and analyzed its performance for different numbers of threads and transform lengths. We observed that the parallel FFT algorithm achieved a significant speedup over the sequential FFT algorithm. However, the speedup plateaued at a certain number of threads and transform length, indicating that the algorithm is limited by memory bandwidth and cache size. We can further improve the performance of the algorithm by optimizing the cache usage and using distributed-memory parallelism.