Zamin Ahmadi **Matrix Multiplication** **222283975**

Roadmap for Parallelizing Matrix Multiplication

1. Decomposition of the Problem:

   Matrix multiplication:
   - Matrix A: 2x4
   - Matrix B: 4x3
   - Result matrix: 2x3

   Task decomposition:

   1. Data partitioning:
      a. Matrix rows: each thread processes a specific row of the result matrix.
      b. Matrix columns: each thread calculates all columns for its assigned row.
   2. Thread management:
      a. Thread creation: create a thread for each row of the result matrix.
      b. Thread execution: each thread executes the `multiplyRow` function to compute its assigned rows of the result matrix.
      c. Thread joining ensures all threads complete their execution before proceeding.

2. Parallel tasks vs Sequential tasks

Parallel tasks:

- Matrix row computation:
  - Each row computes one row of the result matrix
  - Threads work independently to compute their assigned rows

Sequential tasks:

- Matrix initialization:
  - Initialize matrices before starting threads.
- File I/O:
  - Write results to a file after all threads have completed their computations.
- Time measurement:

- Measure time after all threads have finished executing.

Code summary

Thread function: multiplyRow

- Multiplies a specific row of matrix_a with matrix_b and stores the result in the corresponding row of the result matrix.

Main function:

- Initializes matrices
- Creates a result matrix to store the output
- Sets up threads to handle each row of the matrix multiplication
- Manages the creation and joining of threads
- Measures execution time and writes result to a file

Evaluation

1. Sequential program:
   Resulting matrix (Sequential Multiplication):
   43 27 39
   127 95 131
   Execution Time (ms): 3

2. Parallel program:
   Resulting matrix (Parallel Multiplication):
   43 27 39
   127 95 131
   Execution Time (ms): 42

3. OMP program:
   Resulting matrix (OMP Parallelism):
   43 27 39
   127 95 131
   Execution Time (ms): 2

The sequential program, which performs matrix multiplication without any parallelism, has the fastest execution time at 3 milliseconds. However, when parallelism is introduced using pthreads, the execution time increases to 42 milliseconds. This higher time might be due to the overhead of creating and managing threads, which can be significant for smaller tasks. In contrast, using OpenMP for parallelism results in the fastest execution time of 2 milliseconds. This is because OpenMP handles thread management and load balancing more efficiently, reducing overhead and optimizing performance. Overall, OpenMP provides the best performance for matrix multiplication, especially when parallelism is beneficial.