Parallel And Distributed Computing (UCS645)

Assignment 1

Submitted To: Dr. Saif Nalband

Submitted By: Rohini Bansal 102317175

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Deemed to be University)

PATIALA, PUNJAB

INDIA

January-May 2026

Experiment 1: Performance Evaluation of DAXPY Loop using OpenMP

Objective

The objective of this experiment is to analyze the performance improvement obtained by parallelizing the DAXPY loop using OpenMP and to study the effect of increasing the number of threads on execution time and speedup.

Problem Description

DAXPY stands for *Double precision A·X Plus Y*, where **A** is a scalar value and **X** and **Y** are one-dimensional vectors of size 2^16. The computation performed in each iteration is:

$X[i] = a \times X[i] + Y[i]$

Each iteration of the loop is independent of the others, making the DAXPY loop highly suitable for parallel execution.

Methodology

The DAXPY loop was parallelized using the OpenMP parallel for directive. Execution time was measured using the OpenMP timing function omp_get_wtime(). The experiment was started with two threads and repeated by increasing the number of threads. Speedup was calculated as the ratio of serial execution time to parallel execution time.

Results and Observations

It was observed that execution time decreases as the number of threads increases. Speedup improves significantly up to a certain number of threads.

Analysis

Maximum speedup was obtained when the number of threads was equal to the number of physical CPU cores. Increasing the number of threads beyond this point did not improve performance due to increased context switching, cache contention, and memory bandwidth limitations.

Conclusion

The DAXPY loop exhibits near-linear speedup up to the number of physical CPU cores. Beyond this limit, parallel overhead dominates and reduces efficiency.

Experiment 2: Parallel Matrix Multiplication using OpenMP

Objective

The objective of this experiment is to study and compare different parallel implementations of matrix multiplication using OpenMP.

Problem Description

Given two matrices A and B of size 1000 × 1000, the resulting matrix C is computed as:

$C[i][j] = \Sigma A[i][k] \times B[k][j]$

Matrix multiplication is computationally intensive and suitable for parallel execution.

Methodology

Two approaches were implemented:

1. **1D Threading**, where only the outer loop (rows) is parallelized.

2. **2D Threading**, where both row and column loops are parallelized.

Execution time was measured for both approaches and compared.

Results and Observations

1D threading showed significant improvement over serial execution. 2D threading further improved performance by distributing the workload more evenly among threads.

Analysis

In 1D threading, load imbalance may occur as each thread handles a fixed set of rows. In 2D threading, workload distribution is more uniform, resulting in better CPU utilization.

Conclusion

2D threaded matrix multiplication achieves better performance and scalability compared to 1D threading for large matrices.

Experiment 3: Calculation of π using OpenMP Reduction

Objective

The objective of this experiment is to compute the value of π using numerical integration and demonstrate the use of OpenMP reduction.

Problem Description

The value of π is approximated using the integral:

$$\pi = \int_0^1 4 / (1 + x^2)\, dx$$

The interval is divided into a large number of steps, and partial results are summed.

Methodology

The summation loop was parallelized using OpenMP. A reduction clause was used to safely combine partial sums computed by different threads.

Results and Observations

Parallel execution significantly reduced execution time while maintaining accuracy. The reduction clause prevented race conditions.

Analysis

Without reduction, multiple threads updating a shared variable would cause incorrect results. Reduction ensures correctness by maintaining private copies and combining them at the end.

Conclusion

OpenMP reduction provides an efficient and correct method for cooperative parallel computation of π.

Final Conclusion

OpenMP simplifies parallel programming on shared-memory systems. Experimental results demonstrate improved performance and scalability when parallelism is applied appropriately, though gains are limited by hardware and overhead constraints.

Note

All source codes corresponding to these experiments are provided separately in the GitHub repository as q1.cpp,q2.cpp,q3.cpp.