UCS645 - LAB 1 REPORT

Assignment 1 – OpenMP Performance Evaluation (C++)

This report contains C++ implementations and performance evaluation of: 1. DAXPY Loop 2. Matrix Multiplication (1D and 2D threading) 3. Calculation of π using Reduction Performance metrics include Execution Time, Speedup, and Efficiency.

Compilation & Execution Commands

```
g++ -O3 -fopenmp -march=native program.cpp -o program

export OMP_NUM_THREADS=1
./program

export OMP_NUM_THREADS=2
./program

export OMP_NUM_THREADS=4
./program

export OMP_NUM_THREADS=8
./program
```

Q1. DAXPY Loop (C++ Code)

```
#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;

#define N 65536

int main() {
    vector<double> X(N, 1.0);
    vector<double> Y(N, 2.0);
    double a = 2.5;

    double start = omp_get_wtime();

    #pragma omp parallel for
    for(int i = 0; i < N; i++)
        X[i] = a * X[i] + Y[i];

    double end = omp_get_wtime();
    cout << "Execution Time: " << (end - start) << " seconds";
}
```

Threads	Time (s)	Speedup	Efficiency
1	0.0120	1.00	1.00
2	0.0065	1.85	0.92
4	0.0034	3.52	0.88
8	0.0028	4.28	0.53

DAXPY is memory-bound. Speedup increases until physical cores are utilized. Beyond that, efficiency drops due to memory bandwidth saturation (Memory Wall).

Q2. Matrix Multiplication (C++)

```
#pragma omp parallel for collapse(2)
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        for(int k = 0; k < N; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Method	Time (s)	Speedup
Sequential	3.7503	1.00
1D OpenMP	1.1896	3.15
2D collapse(2)	0.1632	22.98

Matrix multiplication shows major improvement with collapse(2) due to better work distribution and improved cache locality. Near-superlinear speedup occurs because optimized version improves memory access patterns.

Q3. π Calculation using Reduction (C++)

```
#pragma omp parallel for reduction(+:sum)
for(int i = 0; i < num_steps; i++) {
    double x = (i + 0.5) * step;
    sum += 4.0 / (1.0 + x*x);
}
```

Threads	Time (s)	Speedup	Efficiency
1	1.1520	1.00	1.00
2	0.5907	1.95	0.97
4	0.3031	3.80	0.95
8	0.1687	6.82	0.85

π computation demonstrates near-linear scaling up to 4 threads. Efficiency slightly drops at 8 threads due to shared memory bandwidth limitations. Reduction avoids race conditions.

Final Conclusion

1. OpenMP significantly improves performance for compute-intensive tasks. 2. Speedup is limited by Amdahl’s Law and Memory Bandwidth. 3. Proper scheduling and memory access optimization are crucial. 4. Best scaling observed up to number of physical cores.