Assignment 3 CS4032D Computer Architecture

Team members

S.L. No.	Name	Roll number
1	Pavithra Rajan	B190632CS
2	Cliford Joshy	B190539CS
3	Karthik Sridhar	B190467CS
4	Jesvin Madonna Sebastian	B190700CS

Q1. Performance comparison of pthread and openMP for a parallelizable problem

A parallelizable problem in computer architecture is a problem that can be divided into smaller independent sub-problems that can be solved concurrently using multiple processors or cores. One such example of a parallelizable problem is Matrix addition.

It is a parallelizable program as it involves adding corresponding elements of two matrices to produce a third matrix. Each element of the output matrix is computed independently of the others, which makes it a good candidate for parallelization.

Parallelizing matrix addition can improve the performance of the program by distributing the workload among multiple processors or cores. Each processor or core can be assigned a subset of the matrix elements to compute, and the results can be combined to produce the final output matrix.

As the size of the input matrices increases, the workload can be distributed among more processors or cores, which can help to maintain performance and reduce the execution time.

Program and Results

In both the cases, we are performing matrix addition of two matrices of size `15000 x 15000`.

Using Pthread

```
gcc matrix_add_posix.c
./a.out
```

```
Execution time without threads: 1.174638 seconds
Execution time with threads: 0.877375 seconds
Speedup: 1.338810
```

Using OpenMP

```
gcc matrix_add_openmp.c -fopenmp
./a.out
```

```
Execution time with threads: 0.250800 seconds
Memory usage: 2.515755 MB
Execution time without threads: 0.797220 seconds
Speedup: 3.178708
```

Analysis

OpenMP and Pthreads are both programming models that can be used to implement parallelism in shared-memory systems. However, OpenMP often provides better speedup than Pthreads for several reasons.

- OpenMP provides a high-level programming interface that is designed to simplify the development of parallel applications. OpenMP allows the programmer to specify parallelism via compiler directives in the codebase, which are automatically translated into threaded code by the compiler. This can make it easier to implement parallelism in a program and can reduce the likelihood of errors.
- OpenMP can automatically manage thread creation and scheduling, which can optimize the use of system resources. OpenMP provides a runtime library that can dynamically allocate threads and distribute workloads among them, which can help to balance the workload and minimize idle time. This can lead to better performance and more significant speedup compared to manually managing threads using Pthreads.
- There is optimization of code at compiler level in the case of OpenMP. The OpenMP compiler can automatically perform optimizations such as loop unrolling and memory prefetching, which can improve cache usage and reduce memory access latency. This can result in faster execution times and more significant speedup compared to hand-written Pthreads code.

Hence, OpenMP often provides better speedup than Pthreads due to its high-level programming interface, automatic thread management, and compiler optimizations. These features can make it easier to implement parallelism in a program, optimize resource usage, and generate more efficient code.