Author: Pupipat Singkhorn, 6532142421

Question 1

What is the main benefit of openmp over standard thread (eg. POSIX thread)?

Answer

OpenMP provides a higher-level abstraction for parallel programming compared to POSIX threads (Pthreads). The key benefits of OpenMP over Pthreads include:

- **Ease of Use**: OpenMP uses compiler directives (#pragma omp ...) that make parallelization simpler without manually managing threads.
- **Automatic Work Distribution**: OpenMP automatically distributes work among available threads, whereas Pthreads require explicit thread management.
- **Better Scalability**: OpenMP can dynamically adjust thread counts based on system resources.
- **Portability**: OpenMP code is easily portable across different architectures and compilers that support OpenMP.

Question 2

For a given code, modify it to take the benefit of simultaneous multithreading processor using openmp. With the new code, what is the potential speed up?

```
#include <stdio.h>
int main(void)
{
int a[100000];
for (int i=0;i<100000;i++) {
a[i]=2*i+i;
printf("a[%d],%d\n",i, a[i]);
}
return 0;
}
```

Answer

```
#include <stdio.h>
#include <omp.h>

#define SIZE 100000

void serial_version(double *serial_time) {
```

```c
    int a[SIZE];
    double start_time = omp_get_wtime();

    for (int i = 0; i < SIZE; i++) {
        a[i] = 2 * i + i; // Computes 3*i
    }

    double end_time = omp_get_wtime();
    *serial_time = end_time - start_time;
    printf("Serial Execution Time: %f seconds\n", *serial_time);
}

void parallel_version(double *parallel_time) {
    int a[SIZE];
    double start_time = omp_get_wtime();

    #pragma omp parallel for
    for (int i = 0; i < SIZE; i++) {
        a[i] = 2 * i + i;
    }

    double end_time = omp_get_wtime();
    *parallel_time = end_time - start_time;
    printf("Parallel Execution Time: %f seconds\n", *parallel_time);
}

int main(void) {
    int num_cores = omp_get_num_procs();
    printf("Number of CPU cores: %d\n", num_cores);

    double serial_time, parallel_time;

    printf("\nRunning Serial Version...\n");
    serial_version(&serial_time);

    printf("\nRunning Parallel Version...\n");
    parallel_version(&parallel_time);

    // Compute and print speedup
    if (parallel_time > 0) {
        double speedup = serial_time / parallel_time;
        printf("\nSpeedup: %.2fX\n", speedup);
    } else {
        printf("\nParallel execution time too small to compute speedup accurately.\n");
    }
    return 0;
}
```

```
gcc-14 -fopenmp -o q2 q2.c
./q2
```

Results:

```
Number of CPU cores: 8

Running Serial Version...
Serial Execution Time: 0.000661 seconds

Running Parallel Version...
Parallel Execution Time: 0.000341 seconds

Speedup: 1.94X
```

Question 3

Base on OpenMP, explain the concepts of work sharing constructs for

- loop constructs: for and do
- sections
- single
- Workshare

Answer

- **Loop Constructs (`for`, `do`):** Distributes loop iterations across threads. Example:

  ```
  #pragma omp parallel for
  for (int i = 0; i < N; i++) { ... }
  ```

- **Sections:** Divides code into independent blocks for parallel execution. Example:

  ```
  #pragma omp parallel sections
  {
    #pragma omp section
    { ... } // Task 1
    #pragma omp section
    { ... } // Task 2
  }
  ```

- **Single:** Ensures a code block is executed by only one thread. Example:

  ```
  #pragma omp single
  { printf("This runs once\n"); }
  ```

- **Workshare (Fortran-specific):** Parallelizes array operations and `FORALL`/`WHERE` statements in Fortran. Not applicable to C/C++. Example:

  ```fortran
  !$OMP WORKSHARE
  A = B + C
  !$OMP END WORKSHARE
  ```

References

- CURC, Using OpenMP with C
- OpenMP, OpenMP Application Programming Interface
- ChatGPT
- DeepSeek