Day4

Table of Contents

1. Parallel Programming

1.1. Definition

Parallel programming involves dividing a problem into smaller tasks that can be executed simultaneously on multiple processors or cores.

1.2. Why Parallel Programming?

- To solve large-scale problems faster.
- To fully utilize multi-core CPUs.
- Examples include simulations, data analysis, and image processing.

1.3. Key Concepts

- Concurrency: Tasks progress simultaneously but not necessarily at the same time.
- Parallelism: True simultaneous execution of tasks on multiple processors.

1.4. Parallel Architectures

- Shared Memory Systems:
    - Single memory space shared by multiple processors.
    - Threads communicate using shared variables.

- Distributed Memory Systems:
  - Each processor has its own memory.
  - Communication happens through message passing.

1.5. Types of Parallelism

- Data Parallelism: Same operation performed on different parts of the data.
- Task Parallelism: Different tasks executed in parallel.

1.6. Challenges in Parallel Programming

- Synchronization: Managing access to shared resources.
- Load Balancing: Distributing work evenly across processors.
- Debugging: Detecting and fixing race conditions and deadlocks.

2. OpenMP

2.1. What is OpenMP?

OpenMP (Open Multi-Processing) is an API for parallel programming on shared memory systems. It allows developers to write parallel code in C, C++, and Fortran.

2.2. Why OpenMP?

- Easy to use with simple directives.
- Portable across platforms.
- Provides efficient parallelism on multi-core CPUs.

2.3. OpenMP Programming Model

- Fork-Join Model:
  - Starts with a single master thread.
  - Forks into multiple threads in parallel regions.

    - Threads join back at the end of parallel regions.
- Threads share memory and require synchronization for shared data.

2.4. Basic hello World in C

```c
#include<stdio.h>
int main(){
    printf("Hello, World\n");
}
```

```
Hello, World
```

2.5. Basic OpenMP Example

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

int main() {
    #pragma omp parallel
    {
        printf("Hello, World : line1\n");
        printf("Hello, World : line2\n");
    }
    return 0;
}
```

2.6. Compiling OpenMP Programs

Use the `-fopenmp` flag to compile OpenMP programs. Example:

```
gcc -fopenmp hello_omp.c -o hello_omp.out
```

2.7. Running OpenMP Programs

Example:

```
./hello_omp.out
```

```
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
Hello, World : line1
Hello, World : line2
```

2.8. OpenMP Directives

- #pragma omp parallel: Defines a parallel region.
- #pragma omp for: Distributes loop iterations among threads.

Example:

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    a[i] = b[i] + c[i];
```

```
}
```

2.9. OpenMP Clauses

- private(var): Each thread has its own copy of the variable.
- shared(var): The variable is shared among all threads.
- reduction(op:var): Combines values from all threads using the specified operation.

Example:

```
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < n; i++) {
    sum += a[i];
}
```

2.10. Environment Variables

- OMP_NUM_THREADS: Sets the number of threads.
- OMP_DYNAMIC: Enables or disables dynamic adjustment of threads.
- OMP_SCHEDULE: Controls the schedule type for loops (e.g., static, dynamic).

Example:

```
export OMP_NUM_THREADS=4
export OMP_SCHEDULE="dynamic"
```

3. Creating a particular number of threads

3.1. code

```
#include<stdio.h>
#include<omp.h>         //for openmp support
int main(){
```

```
    omp_set_num_threads(14); //it will create 4 threads
    #pragma omp parallel num_threads(6)
    {
        printf("Hello from first parallel region\n");
    }

    #pragma omp parallel
    {
        printf("Hello from second parallel region\n");
    }
    return 0;
}
```

3.2. compile

```
gcc hello1.c -fopenmp -o hello1.out
```

3.3. run

```
./hello1.out
```

```
Hello from first parallel region
Hello from first parallel region
Hello from first parallel region
Hello from first parallel region
Hello from first parallel region
Hello from first parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
Hello from second parallel region
```

```
Hello from second parallel region
Hello from second parallel region
```

4. Printing total number of threads inside a region

4.1. code

```c
#include<stdio.h>
#include<omp.h>         //for openmp support
int main(){
    printf("Total number of threads before parallel region : %d\n", omp_get_num_threads());
    omp_set_num_threads(14); //it will create 4 threads
    #pragma omp parallel num_threads(6)
    {
        printf("Hello from first parallel region\n");
        printf("Total number of threads inside first parallel region : %d\n", omp_get_num_threads());
    }

    #pragma omp parallel
    {
        printf("Hello from second parallel region\n");
        printf("Total number of threads inside second parallel region : %d\n", omp_get_num_threads());
    }
    printf("Total number of threads after parallel region : %d\n", omp_get_num_threads());
    return 0;
}
```

4.2. compile

```
gcc hello2.c -fopenmp -o hello2.out
```

4.3. run

```
./hello2.out
```

```
Total number of threads before parallel region : 1
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from first parallel region
Total number of threads inside first parallel region : 6
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Hello from second parallel region
Total number of threads inside second parallel region : 14
Total number of threads after parallel region : 1
```

5. Giving your threads an identity (threadID)

5.1. code

```c
#include<stdio.h>
#include<omp.h>         //for openmp support
int main(){
    #pragma omp parallel num_threads(6)
    {
        printf("Hello from thread %d\n", omp_get_thread_num());
    }

    return 0;
}
```

5.2. compile

```
gcc hello3.c -fopenmp -o hello3.out
```

5.3. run

```
./hello3.out
```

```
Hello from thread 5
Hello from thread 2
Hello from thread 1
Hello from thread 3
Hello from thread 0
Hello from thread 4
```

6. Assign different threads to perform different tasks

6.1. code

```c
#include<stdio.h>
#include<omp.h>         //for openmp support

void task1(int tid){
    printf("executing task1 by thread %d\n", tid);
}

void task2(int tid){
    printf("executing task2 by thread %d\n", tid);
}

int main(){
    #pragma omp parallel num_threads(2)
    {
        int tid = omp_get_thread_num();
        if(tid == 0){
            task1(tid);
        }
        else{
            task2(tid);
        }

    }

    return 0;
}
```

6.2. compile

```
gcc hello4.c -fopenmp -o hello4.out
```

6.3. run

```
./hello4.out
```

```
executing task2 by thread 1
executing task1 by thread 0
```

7. Task

Create four threads. Create a shared variable and two private variable for each threads b and c. Change value of `b` for thread 2 and perform operation `c = a + b + tid` and print the result of `c` for each threads.

7.1. code

```c
#include<stdio.h>
#include<omp.h>         //for openmp support
void task1(int a, int b, int tid){
    int c = a + b + tid;
    printf("Value of C for thread %d is %d\n", tid, c);
}
int main(){
    int a = 6;
    #pragma omp parallel num_threads(4)
    {
        int tid = omp_get_thread_num();
        int b = 6;
        if(tid == 2) b = 7;
        task1(a, b, tid);
    }
    return 0;
}
```

7.2. compile

```
gcc task1.c -fopenmp -o task1.out
```

7.3. run

```
./task1.out
```

```
Value of C for thread 3 is 15
Value of C for thread 1 is 13
Value of C for thread 0 is 12
Value of C for thread 2 is 15
```

Author: Abhishek Raj
Created: 2024-12-18 Wed 17:08