Name: Bhavikk Patel
Register No.: 24MCS0047

Computer system architecture and organization

Computer system architecture is a foundational area of study that addresses the conceptual design and operational structure of computer systems. It involves the configuration of hardware components, including the CPU, memory units, and I/O devices, and defines how these elements collaborate to perform computational tasks effectively. The architecture includes aspects such as instruction set design, addressing modes, and data types, which are crucial for the efficient functioning of a computer system.Computer organization involves the study of mechanisms like pipelining, cache memory, and bus systems that directly impact the performance and efficiency of a computer system.

Understanding the connection between computer architecture and organization is important for optimizing system performance, enhancing computational efficiency, and ensuring the seamless execution of software applications. This forms the backbone of advancements in areas such as parallel processing, distributed systems and more.

Parallel computing and parallel programming languages

Parallel computing focuses on executing multiple computations simultaneously to solve complex problems more efficiently. It involves dividing a large problem into smaller, independent tasks that can be processed concurrently across multiple processors. This approach significantly reduces computation time and enhances performance, making it essential for applications in scientific research, large-scale simulations, data analysis, and real-time processing. Key concepts in parallel computing include data, task parallelism, and the design of parallel algorithms that can effectively use the maximum of the underlying hardware architecture.

Parallel programming languages are designed to facilitate the development of parallel applications by providing constructs that explicitly express parallelism. These languages offer abstractions for managing multiple threads or processes, coordinating communication and synchronization between them, and optimizing resource utilization. OpenMP is one of the examples of parallel programming languages and CUDA for GPU programming.

Introduction to OpenMP

OpenMP is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C/C++. It is a standard for parallel programming that enables the development of concurrent programs by adding simple compiler directives to sequential code. These directives allow the programmer to define parallel regions where the code will be executed by multiple threads concurrently, taking advantage of modern multi-core processors to improve performance.

OpenMP is crucial for efficiently solving computationally intensive problems in fields such as scientific computing, data analysis, and engineering simulations. OpenMP's ease of use and compatibility with existing codebases make it an excellent tool for quickly parallelizing applications without the need for extensive code rewrites. With OpenMP, we can leverage the full potential of multi-core processors, optimize resource usage, and significantly reduce computation times.

2. Write a C/C++ program using OpenMP To print hello world. To print the following environment detail: Number of threads, Thread number, Number of processors and Maximum threads with sample message.

```cpp
1   // BHAVIKK D PATEL
2   // 24MCS0047
3
4   #include <iostream>
5   #include <omp.h>
6
7   using namespace std;
8
9   int main() {
10      omp_set_num_threads(4);
11
12      cout << "Number of threads = " << omp_get_num_threads() << "\n";
13      cout << "Number of processors = " << omp_get_num_procs() << "\n";
14      cout << "Max threads = " << omp_get_max_threads() << "\n";
15
16      #pragma omp parallel
17      {
18          cout << "Hello = " << omp_get_thread_num() << "\n";
19      }
20  }
```

```
get_wtime();

t_wtime();
ication Thread = " << omp_get_thread

get_wtime();

t_wtime();
n Thread = " << omp_get_thread_num()

get_wtime();

t_wtime();
 AND Thread num = " << omp_get_threa

get_wtime();
```

Windows PowerShell   ×   +   ∨

```
PS D:\Vit\COA\openMP\openMP> hostname
BhavikkPatel
PS D:\Vit\COA\openMP\openMP>
```
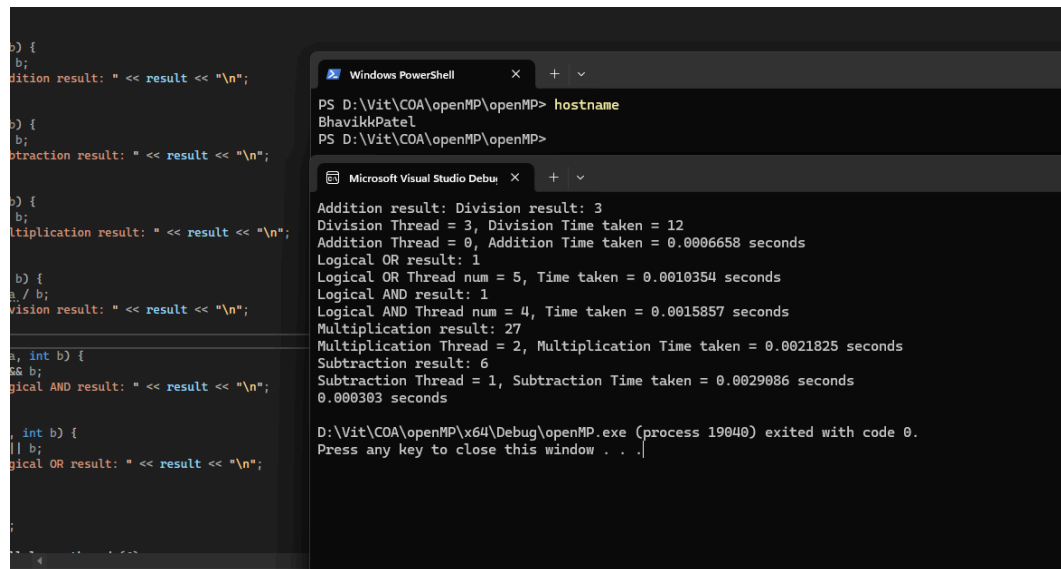
Microsoft Visual Studio Debug   ×   +   ∨

```
Number of threads = 1
Number of processors = 12
Max threads = 4
Hello = 0Hello = 3Hello = 1

Hello = 2
```

Output: (2)

3. Write a C/C++ program using open MP to perform the arithmetic operations and logical operations between two integers using multiple threads and measure the time?

```cpp
// BHAVIKK D PATEL
// 24MCS0047

#include <iostream>
#include <omp.h>

void add(int a, int b) {
    int result = a + b;
    std::cout << "Addition result: " << result << "\n";
}
void sub(int a, int b) {
    int result = a - b;
    std::cout << "Subtraction result: " << result << "\n";
}
void mul(int a, int b) {
    int result = a * b;
    std::cout << "Multiplication result: " << result << "\n";
}
void divi(int a, int b) {
    double result = a / b;
    std::cout << "Division result: " << result << "\n";
}
void logicalAND(int a, int b) {
    bool result = a && b;
    std::cout << "Logical AND result: " << result << "\n";
}
void logicalOR(int a, int b) {
    bool result = a || b;
    std::cout << "Logical OR result: " << result << "\n";
}

int main() {
    int a = 4, b = 5;
    #pragma omp parallel num_threads(6)
    {
        double startTime, endTime;
        switch (omp_get_thread_num()) {
        case 0:
            startTime = omp_get_wtime();
            add(a, b);
            endTime = omp_get_wtime();
            std::cout << "Addition Thread = " << omp_get_thread_num() << ", Addition Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        case 1:
            startTime = omp_get_wtime();
            sub(a, b);
            endTime = omp_get_wtime();
            std::cout << "Subtraction Thread = " << omp_get_thread_num() << ", Subtraction Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        case 2:
            startTime = omp_get_wtime();
            mul(a, b);
            endTime = omp_get_wtime();
            std::cout << "Multiplication Thread = " << omp_get_thread_num() << ", Multiplication Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        case 3:
            startTime = omp_get_wtime();
            divi(a, b);
            endTime = omp_get_wtime();
            std::cout << "Division Thread = " << omp_get_thread_num() << ", Division Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        case 4:
            startTime = omp_get_wtime();
            logicalAND(a, b);
            endTime = omp_get_wtime();
            std::cout << "Logical AND Thread num = " << omp_get_thread_num() << ", Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        case 5:
            startTime = omp_get_wtime();
            logicalOR(a, b);
            endTime = omp_get_wtime();
            std::cout << "Logical OR Thread num = " << omp_get_thread_num() << ", Time taken = " << (endTime - startTime) << " seconds\n";
            break;
        }
    }
    return 0;
}
```
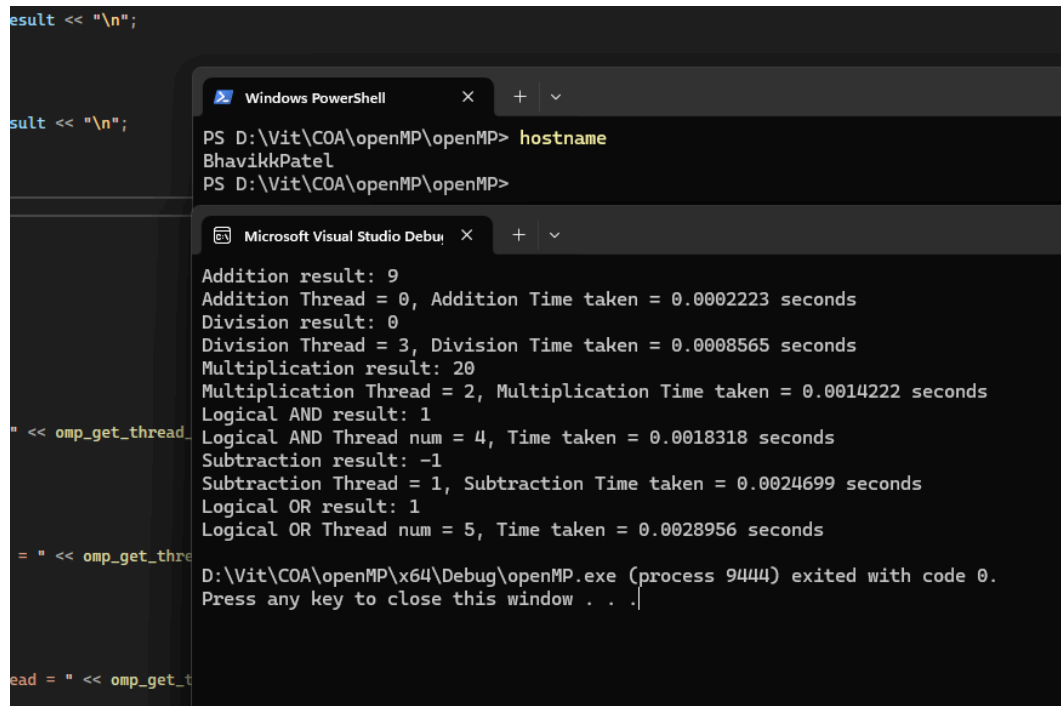
OUTPUT 1

OUTPUT 2

4. Write a C/C++ program using open MP to find the largest and smallest among three numbers using thread approach. [Analyze the time between serial and parallel approach]

```cpp
1   // BHAVIKK D PATEL
2   // 24MCS0047
3
4   #include <iostream>
5   #include <omp.h>
6
7   int main() {
8       int n1 = 10, n2 = 20, n3 = 5;
9       int largest, smallest;
10      double startTime = omp_get_wtime();
11      #pragma omp parallel sections
12      {
13          #pragma omp section
14          {
15              largest = n1;
16              if (n2 > largest) largest = n2;
17              if (n3 > largest) largest = n3;
18              std::cout << "Thread = " << omp_get_thread_num() << " found largest: " << largest << "\n";
19          }
20
21          #pragma omp section
22          {
23              smallest = n1;
24              if (n2 < smallest) smallest = n2;
25              if (n3 < smallest) smallest = n3;
26              std::cout << "Thread = " << omp_get_thread_num() << " found smallest: " << smallest << "\n";
27          }
28      }
29      double endTime = omp_get_wtime();
30      std::cout << "Largest = " << largest << "\n";
31      std::cout << "Smallest = " << smallest << "\n";
32      std::cout << "Parallel Execution time = " << (endTime - startTime) << "\n";
33
34      // sequential execution for analysis
35      std::cout << "Sequential \n";
36      startTime = omp_get_wtime();
37      largest = n1;
38      if (n2 > largest) largest = n2;
39      if (n3 > largest) largest = n3;
40      smallest = n1;
41      if (n2 < smallest) smallest = n2;
42      if (n3 < smallest) smallest = n3;
43      endTime = omp_get_wtime();
44      std::cout << "Sequential found largest = " << largest << "\n";
45      std::cout << "Sequential found smallest = " << smallest << "\n";
46      std::cout << "Sequential Execution time = " << (endTime - startTime) << "\n";
47      return 0;
48  }
```

OUTPUT 1


```
ead_num() << " found largest: " << largest << "\n";
```
Windows PowerShell
```
PS D:\Vit\COA\openMP\openMP> hostname
BhavikkPatel
PS D:\Vit\COA\openMP\openMP>
```
Microsoft Visual Studio Debug
```
Thread = 0 found largest: 9
Thread = 2 found smallest: 3
Largest = 9
Smallest = 3
Parallel Execution time = 0.0011679
Sequential
Sequential found largest = 9
Sequential found smallest = 3
Sequential Execution time = 1.00001e-07

D:\Vit\COA\openMP\x64\Debug\openMP.exe (process 17712) exited with code 0.
Press any key to close this window . . .
```

OUTPUT 2

Windows PowerShell
```
PS D:\Vit\COA\openMP\openMP> hostname
BhavikkPatel
PS D:\Vit\COA\openMP\openMP>
```
Microsoft Visual Studio Debug
```
Thread = 2 found largest: 20
Thread = 0 found smallest: 5
Largest = 20
Smallest = 5
Parallel Execution time = 0.0006938
Sequential
Sequential found largest = 20
Sequential found smallest = 5
Sequential Execution time = 0

D:\Vit\COA\openMP\x64\Debug\openMP.exe (process 2228) exited with code 0.
Press any key to close this window . . .
```

5. Write a C/C++ program using OpenMP to demonstrate the shared, private, first private, last private and thread private concepts.

1. **shared**:

    Variables that are shared among all threads. Each thread can read and write to the same memory location.

2. **Private**:

    Each thread has its own instance of the variable. The value is uninitialized i.e default value is 0 in case of int as shown in below example.

3. **Firstprivate**:

    Similar to private, but each thread's instance is initialized with the value of the variable before entering the parallel region.

4. **Lastprivate**:

    Similar to private, but the value of the variable from the last iteration of the loop is copied back to the original variable after the parallel region.

5. **Threadprivate**:

    Variables that are global but each thread has its own instance of the variable.

```cpp
// BHAVIKK D PATEL
// 24MCS0047

#include <iostream>
#include <omp.h>

int main() {
    int n1 = 10, n2 = 20, n3 = 5;
    int largest, smallest;
    double startTime = omp_get_wtime();
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            largest = n1;
            if (n2 > largest) largest = n2;
            if (n3 > largest) largest = n3;
            std::cout << "Thread = " << omp_get_thread_num() << " found largest: " << largest << "\n";
        }

        #pragma omp section
        {
            smallest = n1;
            if (n2 < smallest) smallest = n2;
            if (n3 < smallest) smallest = n3;
            std::cout << "Thread = " << omp_get_thread_num() << " found smallest: " << smallest << "\n";
        }
    }
    double endTime = omp_get_wtime();
    std::cout << "Largest = " << largest << "\n";
    std::cout << "Smallest = " << smallest << "\n";
    std::cout << "Parallel Execution time = " << (endTime - startTime) << "\n";

    // sequential execution for analysis
    std::cout << "Sequential \n";
    startTime = omp_get_wtime();
    largest = n1;
    if (n2 > largest) largest = n2;
    if (n3 > largest) largest = n3;
    smallest = n1;
    if (n2 < smallest) smallest = n2;
    if (n3 < smallest) smallest = n3;
    endTime = omp_get_wtime();
    std::cout << "Sequential found largest = " << largest << "\n";
    std::cout << "Sequential found smallest = " << smallest << "\n";
    std::cout << "Sequential Execution time = " << (endTime - startTime) << "\n";
    return 0;
}
```

OUTPUT 1:

```
Windows PowerShell                    ×    +  ∨                        —  □  ×

PS C:\Users\bhavi> hostname
BhavikkPatel

Microsoft Visual Studio Debu    ×    +  ∨                        —  □  ×

Variables Value before:
shared: 10
private: 20
firstprivate: 30
lastprivate: 40
threadPrivate: 50
Thread 0 ::: shared: 10
Thread 0 ::: private: 0
Thread 0 ::: firstprivate: 30
Thread 0 ::: lastprivate: 40
Thread 0 ::: threadPrivate: 50
Thread 1 ::: shared: 11
Thread 1 ::: private: 1
Thread 1 ::: firstprivate: 31
Thread 1 ::: lastprivate: 41
Thread 1 ::: threadPrivate: 1
Thread 2 ::: shared: 13
Thread 2 ::: private: 2
Thread 2 ::: firstprivate: 32
Thread 2 ::: lastprivate: 43
Thread 2 ::: threadPrivate: 2
Variables Value after:
shared: 13
private: 20
firstprivate: 30
lastprivate: 43
threadPrivate: 50

D:\Vit\COA\openMP\x64\Debug\openMP.exe (process 18500) exited with code 0.
Press any key to close this window . . .
```

OUTPUT 2:

```
Variables Value before:
shared: 1
private: 1
firstprivate: 1
lastprivate: 1
threadPrivate: 100
Thread 0 ::: shared: 1
Thread 0 ::: private: 0
Thread 0 ::: firstprivate: 1
Thread 0 ::: lastprivate: 1
Thread 0 ::: threadPrivate: 100
Thread 1 ::: shared: 2
Thread 1 ::: private: 1
Thread 1 ::: firstprivate: 2
Thread 1 ::: lastprivate: 2
Thread 1 ::: threadPrivate: 1
Thread 2 ::: shared: 4
Thread 2 ::: private: 2
Thread 2 ::: firstprivate: 3
Thread 2 ::: lastprivate: 4
Thread 2 ::: threadPrivate: 2
Variables Value after:
shared: 4
private: 1
firstprivate: 1
lastprivate: 4
threadPrivate: 100

D:\Vit\COA\openMP\x64\Debug\openMP.exe (process 6468) exited with code 0.
Press any key to close this window . . .
```