Q1.

(a)

• Explain the terms *concurrency* and *parallelism*.

• What are the similarities and differences between the two concepts? Explain clearly through examples.

(5)

(b)

• Explain clearly what is meant by *Principle of referential locality*.

• How does an operating system take advantage of this principle while executing a process?

(5)

(c)

• Explain why the performance of a multi-threaded program depends on good cache management.

• What is *Data parallelism*? How do OpenMP threads make use of data parallelism? Explain through an example.

(5)
Q2.

(a) Explain clearly the events that occur in this program, starting from the `omp_set_num_threads(4)` statement.

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

int main (int argc, char *argv[])
{
    int nthreads, tid;
    #pragma omp parallel private(nthreads, tid)
    {
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);

        if (tid == 0)
        {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    }
}
```
Q2. continued ..

(b) Explain the terms `private`, `shared`, `firstprivate` and `lastprivate` in relation to an OpenMP program.

(c) Write a C program using OpenMP that finds the maximum and minimum of 100 integers using 4 threads. Use an array that holds 100 integers. Distribute this array evenly among the four threads. Each thread should find its local maximum and minimum in its own part of the array. Finally the global maximum and minimum should be printed by the master thread.

**Note:** You need not write code for reading the input integers in the array. I am interested only in the OpenMP part of the code.
Q3.
(a) Explain each of the parameters in the following function calls:

\[
\text{MPI\_Send}(&\text{taskid}, 1, \text{MPI\_INT}, \text{partner}, 1, \text{MPI\_COMM\_WORLD});
\]

\[
\text{MPI\_Recv}(&\text{message}, 1, \text{MPI\_INT}, \text{partner}, 1, \text{MPI\_COMM\_WORLD}, &\text{status});
\]

(b) • What is an \textit{MPI communicator}? What is \texttt{MPI\_COMM\_WORLD}?

• Explain the purpose of the two functions \texttt{MPI\_Comm\_size()} and \texttt{MPI\_Comm\_rank()}.

(c) What will happen if the following code segments are executed by these two processes? Do you think it needs a correction? If you think so, explain your correction.

• Process 0 executes:

\[
\text{MPI\_Recv}(&\text{yourdata}, 1, \text{MPI\_FLOAT}, 1, \text{tag}, \text{MPI\_COMM\_WORLD}, &\text{status});
\]

\[
\text{MPI\_Send}(&\text{mydata}, 1, \text{MPI\_FLOAT}, 1, \text{tag}, \text{MPI\_COMM\_WORLD});
\]

Process 1 executes:

\[
\text{MPI\_Recv}(&\text{yourdata}, 1, \text{MPI\_FLOAT}, 0, \text{tag}, \text{MPI\_COMM\_WORLD}, &\text{status});
\]

\[
\text{MPI\_Send}(&\text{mydata}, 1, \text{MPI\_FLOAT}, 0, \text{tag}, \text{MPI\_COMM\_WORLD});
\]

• What is the difference between \textit{blocking} and \textit{non-blocking} communication in MPI?

(d) Explain why a combined MPI and OpenMP implementation framework is a good model for high performance computing. Given a compute-intensive problem, how would you exploit both MPI and OpenMP for designing a solution that will result in performance improvement?