EXPERIMENT 5

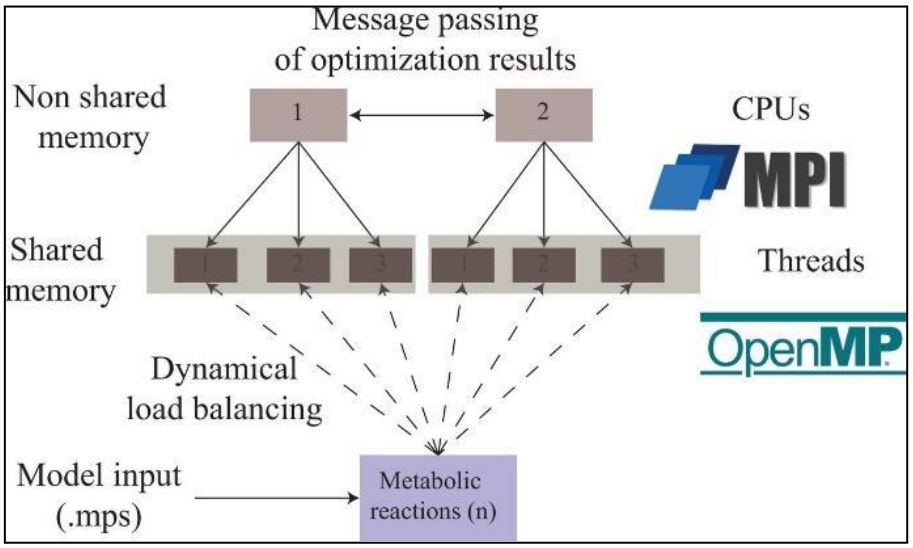
AIM: To implement a program to demonstrate balancing of workload on MPI Platform.

THEORY:
Load balancing is the process of distributing a set of tasks over a set of resources (computing units), with the aim of making their overall processing more efficient. Load balancing can optimize the response time and avoid unevenly overloading some compute nodes while other compute nodes are left idle. Load balancing is the subject of research in the field of parallel computers.

Two main approaches exist: static algorithms, which do not consider the state of the different machines, and dynamic algorithms, which are usually more general and more efficient but require exchanges of information between the different computing units, at the risk of a loss of efficiency.

When splitting the workload across multiple processors, the speedup describes the achieved reduction of the runtime compared to the serial version. Obviously, the higher the speedup, the better. However, the effect of increasing the number of processors usually goes down at a certain point when they cannot be utilized optimally anymore. This parallel efficiency is defined as the achieved speedup divided by the number of processors used. It usually is not possible to achieve a speedup equal to the number of processors used as most applications have strictly serial parts (Amdahl's Law).

Load balancing is of great importance when utilizing multiple processors as efficiently as possible. Adding more processors creates a noticeable amount of synchronization overhead. Therefore, it is only beneficial if there is enough work present to keep all processors busy at the same time. This means tasks should not be assigned randomly as this might lead to serial behavior where only one processor is working while the others have already finished their tasks, e.g., at synchronization points (e.g., barriers) or at the end of the program.



CODE:

```
#include <iostream>
#include <ctime>
#include <mpi.h>
using namespace std;

double int_theta(double E) {
   double result = 0;
   for (int k = 0; k < 20000; k++)
      result += E * k;
   return result;
}

int main() {
   int n = 3500000;
   int counter = 0;
   time_t timer;
   int start_time = time(&timer);
   int myid, numprocs, k;
   double integrate, result;
   double end = 0.5;
```

```
   double start = -2.;
   double E;
   double factor = (end - start) / (n * 1.);
   integrate = 0;
   MPI_Init(NULL, NULL);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   for (k = myid; k < n + 1; k += numprocs) {
      E = start + k * (end - start) / n;
      if ((k == 0) || (k == n))
         integrate += 0.5 * factor * int_theta(E);
      else
         integrate += factor * int_theta(E);
      counter++;
   }
   cout << "Process " << myid << " took " << time(&timer) - start_time << " s" << endl;
   cout << "Process " << myid << " performed " << counter << " computations" <<
endl;
   MPI_Reduce(&integrate, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
   if (myid == 0)
      cout << "The Final Result is " << result << endl;
   MPI_Finalize();
   return 0;
}
```

OUTPUT:

```
C:\Users\JARVIS\source\repos\Experiment 5\x64\Debug>mpiexec -n 4 "Experiment
5.exe"
Process 2 took 87 s
Process 2 performed 875000 computations
Process 3 took 87 s
Process 3 performed 875000 computations
Process 1 took 87 s
Process 1 performed 875000 computations
```

```
Process 0 took 87 s
Process 0 performed 875001 computations
The Final Result is -3.74981e+08

C:\Users\JARVIS\source\repos\Experiment 5\x64\Debug>mpiexec -n 8 "Experiment
5.exe"
Process 2 took 40 s
Process 2 performed 437500 computations
Process 7 took 40 s
Process 7 performed 437500 computations
Process 3 took 40 s
Process 3 performed 437500 computations
Process 1 took 40 s
Process 1 performed 437500 computations
Process 4 took 40 s
Process 4 performed 437500 computations
Process 6 took 40 s
Process 6 performed 437500 computations
Process 0 took 40 s
Process 0 performed 437501 computations
Process 5 took 40 s
Process 5 performed 437500 computations
The Final Result is -3.74981e+08

C:\Users\JARVIS\source\repos\Experiment 5\x64\Debug>mpiexec -n 2 "Experiment
5.exe"
Process 0 took 173 s
Process 0 performed 1750001 computations
Process 1 took 173 s
Process 1 performed 1750000 computations
The Final Result is -3.74981e+08
```

CONCLUSION: Load balancing is the process of spreading tasks between a set of nodes to have equal resource utilization across all. With MPI it is possible to distribute task subsets among a collection of nodes in a communication domain. Each node is thus executing the same operation but on different data. This ensures consistent response times for requests which scale downward as the number of nodes increases.