CS-301 High Performance Computing
Lab 5

Pratvi Shah 201801407
Arkaprabha Banerjee 201801408

Experiments and the corresponding observations:

1. **Understanding your CPU architecture using *lscpu***
    - Architecture: **x86_64**
    - Byte Order: Little-endian
    - CPU(s): 16 x 4
    - Model Name: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
    - L1d cache: 32K
    - L1i cache: 32K
    - L2 cache: 256K
    - L3 cache: 20480K

1 Introduction

This experiment shall aim to compare various performance metrics for trapezoidal integration to find the value of π. We shall compare parallel implementations via OpenMPI as well as the standard OpenMP implementations to the traditional serial implementation.

Question 1: Calculate value of π using Trapezoidal Integration

Implementation Details:

1. Description about the Serial implementation

In serial implementation a single loop runs for all the N(problem size) steps and gives the value of the integral using trapezoidal method. The output of the code gave us the desired value of π.

2. Description about the Parallel implementation via OpenMP

- We first defined a simple *pragma omp parallel* directive to define the parallel region
- We then used the **for** directive to implicitly divide the computations of the entire loop between all the available threads.
- Reduction clause was used to collect the sum of all the value into a global variable.

3. Description about the Parallel implementation via OpenMPI

- Two variations have been taken for this implementation
    1. Point to Point Communication with blocking : Each core shall compute a local integral and send it to the master core (rank 0) to compute the net integral by using the default blocked send clause.
    2. Unblocked Point to Point Communication : Each core shall compute a local integral and send it to the master core (rank 0) to compute the net integral by using a unblocked send clause.

3. Broadcast Communication with reduce clause : Master core shall broadcast the relevant information to all cores. The local integrals are evaluated into the global net value via reduction clause.

- Furthermore, we shall vary the number of cores taken (4,8,16,32) , the Problem Size (N=1024 and N=131072) as well as the location of the cores.
- We shall attempt to run our code concurrently via OpenMPI on a distributed system with 4 compute nodes each having 16 cores.

Complexity

We consider complexity in terms of the number of computations done by each thread or each core.

Complexity of serial code:

In serial code total number of computations are same as the problem size so, complexity of serial code is $O(N)$.

Complexity of parallel code:

In methods which explicitly define the for loops the number of computations are equally divided amongst all the threads. For the implicit methods we assume that similar condition will arise. Hence, the complexity of any parallelisation will be $O(N/P)$ where P is the number of threads in case of OpenMP and number of cores in case of OpenMPI.

Curve Related Analysis

In this section all the plots are Parameter v/s Number of cores in case of OpenMPI and Number of Threads in case of OpenMP. For the sake of simplicity x-axis can we used for both OpenMP and OpenMPI with corresponding x-label. All the measurements of time are in milliseconds unless otherwise mentioned.

Time Curve Related analysis:

- One can observe that the parallel implementations via OpenMP and OpenMPI take less time than the serial implementation in general. However, the time increases for the OpenMP implementation after a certain amount of threads. This is because the overhead increases significantly
- For Large Problem sizes (larger number of trapezoids) OpenMPI takes less time than OpenMP however for smaller problem sizes, the reverse happens. This is because for small problem sizes the overhead associated in OpenMPI is significant as compared to OpenMP. However for large problem sizes, we are able to take the full benefit of the individual computation speed of a distributed system. For both problem sizes , we obtain an increase in execution time for OpenMP after a certain threshold on account of the overhead involved.

- Unblocked Point to Point communication takes less time than its Blocked Counterpart. This is because in unblocked communication, the send statement does not wait for the receive statement to receive the information and instead moves on to the next instruction.However, the accuracy in unblocked send is less as compared to blocked send.
- As we increase the number of cores the communication overhead in terms of the percentage of the total end to end time increase even though the algorithmic speedup does not increase much. This can be observed clearly from the (d) figure. Communication time has been approximated as the total wall time minus the algorithmic time. The overhead ratio is the ratio of the Communication time to the total time.
- If we take all cores from the same node then the overhead is less as compared to taking them from different compute nodes.

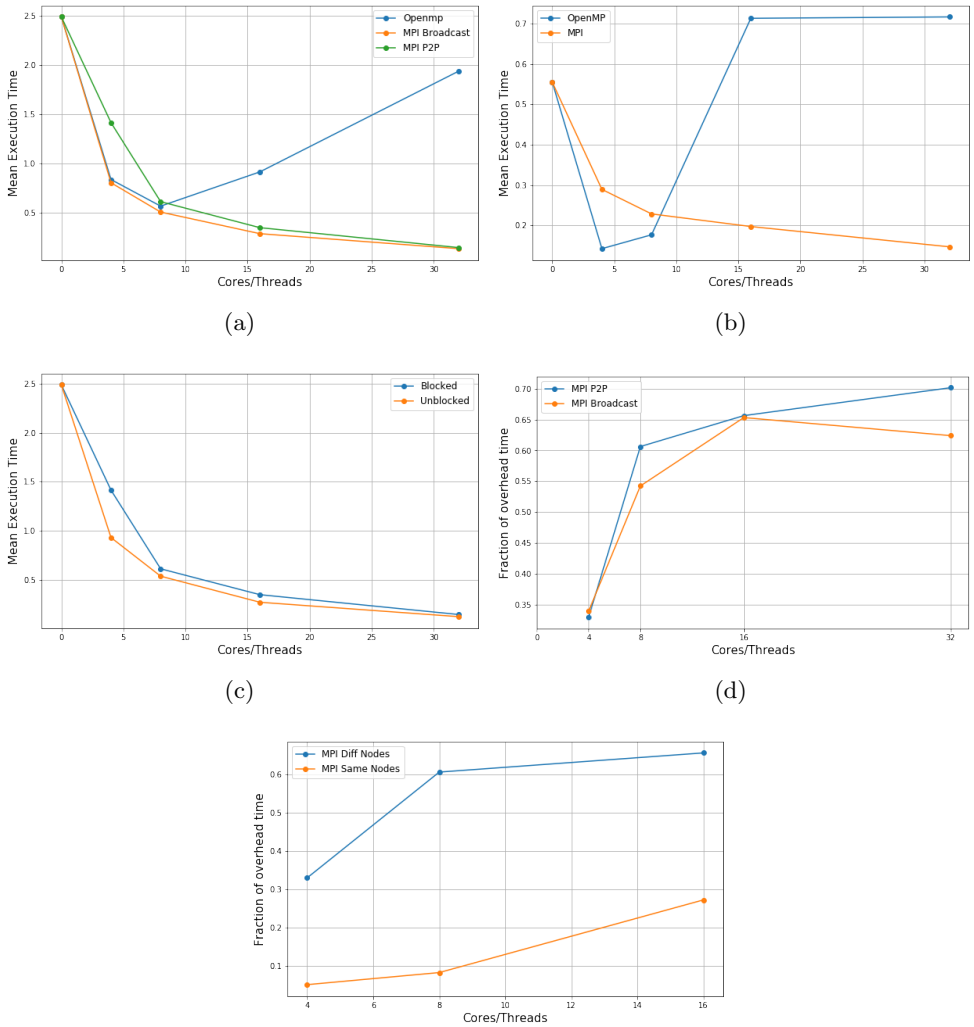

Figure 1: Time in milliseconds
(a) OpenMP and MPI(P2P & Broadcast) for N=131072
(b) OpenMP and MPI(P2P) for N=1024
(c) Blocked and Unblocked Calls of Recv and Send in MPI
(d) Overhead time ratio P2P & Broadcast
(e) Overhead time ratio for MPI P2P in same node and different nodes

Speedup Curve Related analysis

- The above observations directly relate to the speedup. We observe maximum speedup for N=131072 with OpenMPI for higher number of cores and we can see that similar implementation for N=1024 does not produce comparable speedup in case of OpenMPI due to the overhead involved in communication for smaller problem size.
- OpenMPI broadcast method with reduce gives better result than Blocked Point to Point Communication due to the inherent nature of the reduction algorithm.
- Unblocked calls correspond to more speedup than Blocked calls.

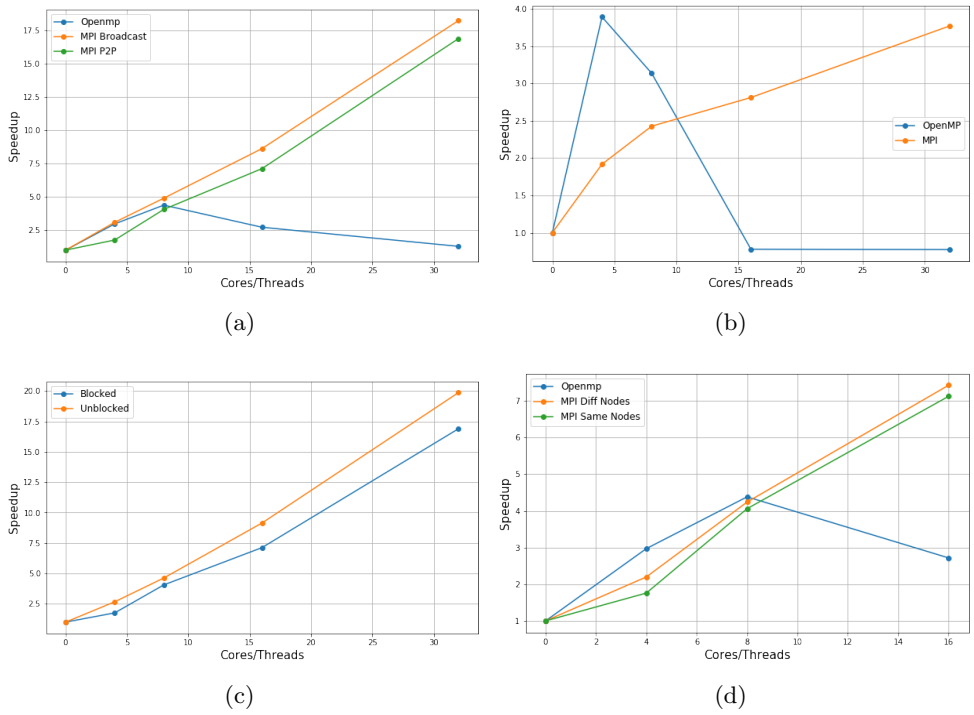

Figure 2: Speedup
(a) OpenMP and MPI(P2P & Broadcast) for N=131072
(b) OpenMP and MPI(P2P) for N=1024
(c) Blocked and Unblocked Calls of Recv and Send in MPI
(d) OpenMP and OpenMPI (P2P) on same and different nodes

Efficiency Curve Related analysis

- The efficiency of openmpi cores is worse for small problem sizes as compared to larger problems/steps.
- Efficiency in both OpenMP and OpenMPI decreases with increase in threads/cores
- If we select all cores from the same compute node then we achieve better efficiency in terms of communication overhead even though the algorithmic speedup remains the same for distributed for compute nodes.

- We may conclude that for higher number of cores or for bigger problem sizes, OpenMPI is more scalable as compared to OpenMP.

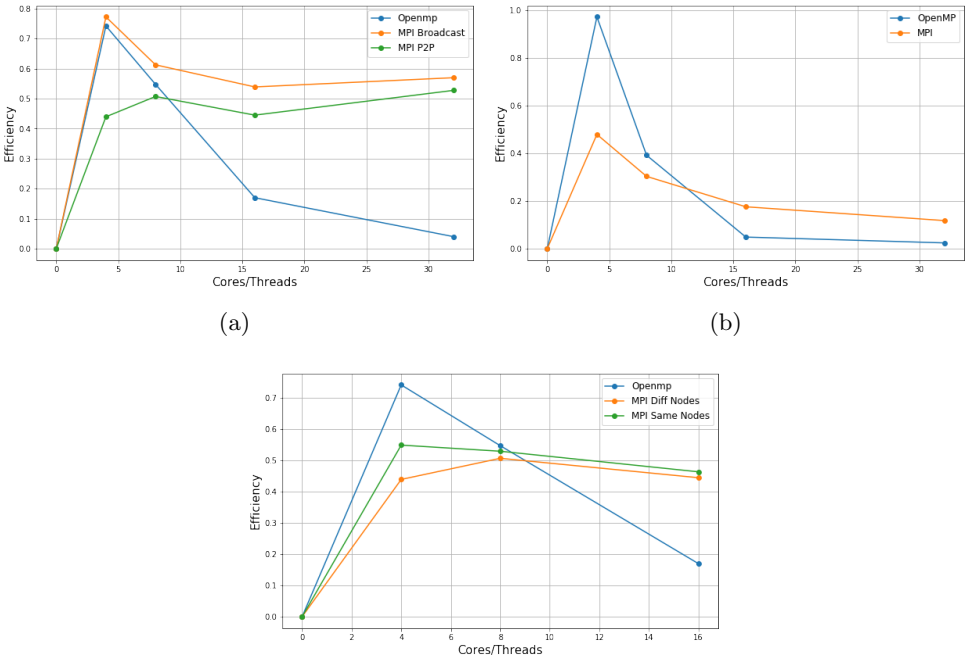

Figure 3: Efficiency
(a) OpenMP and MPI(P2P & Broadcast) for N=131072
(b) OpenMP and MPI(P2P) for N=1024
(c) OpenMP and OpenMPI (P2P) on same and different nodes

Conclusion

From the above observations we can conclude the following points:

- When there are less number of threads OpenMP performs better in terms of time complexity in comparison to the OpenMPI due to the eradication of communication amongst the nodes(communication overhead).
- When the problem size is large OpenMPI performs better due to the access to larger memory from each node as compared to single node and also due to the efficient parallelization by division of work amongst the nodes.
- In OpenMPI, parallelization using MPI_Bcast and MPI_Reduce performs better than the other one with basic MPI calls.