Offloading Halo Exchange Applications - MPI Manual Offloading

DPU Offloading

Nandhana Sakthivel

March 2023
AccelCom Group
Barcelona SuperComputing Center

Contents

1 Introduction

The main objective is to offload the halo exchange applications with the help of MPI by leveraging **NVIDIA's Bluefield2 card** which is known for its networking and data movement performances when compared to a standard **Intel server host**. The halo exchange application used for this purpose is **MiniMD**. MiniMD is a simple, parallel molecular dynamics (MD) code. It is an MD micro application in the Mantevo project at Sandia National Laboratories (http://www.mantevo.org).

2 Structure of the miniMD application

The basic structure of a halo exchange application is **a) Communication and b) Computation.** The halo exchange application we are studying for the purpose of offloading here is miniMD application. The basic elements of miniMD application are

- **Atom**
- **Neighbor**
- **Comm**
- **Force**
- **Thermo**

In the **force-calc-on-BF-with-MPI-and-OpenMP** branch, the parts offloaded to the BlueField are the border elements i.e. ghost cells and neighbor cells. A **neighbor list** is an object containing information about which atoms are located near each other, taking the boundary conditions into account.The main distinction is between half and full neighbor list. In a **full neighbor list**, if atoms A and B are neighbors, A appears on B's list and B appears on A's list. In a half neighbor list, either A appears on B's list or B appears on A's list, but not both. A full neighbor list is thus useful when analysing the neighborhood of a given atom, whereas a **half neighbor list** is useful when looping over all bonds between atoms, since each bond appears only once. In most cases, the neighbor list associated with a potential will be a half neighbor list. **Ghost cells** also called as guard cells are the cells which are around the edges of the domain that contains boundary values.

3 MPI Manual Offloading

Offloading halo exchange application involves two sections:

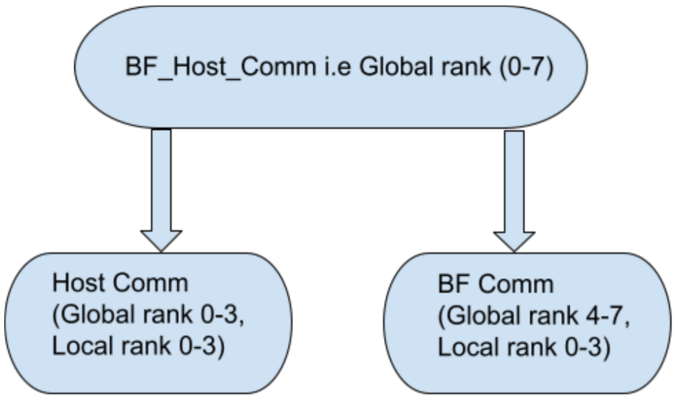
- Offloading Communication
- Offloading Computation

3.1 Offloading

Offloading refers to the data transfer from one processor to another. It is a solution where computations are migrated to the resourceful processors in order to increase the performance. Currently, the offloading is carried out manually with MPI by running the code on both Host and Bluefield simultaneously through the following command as an example:

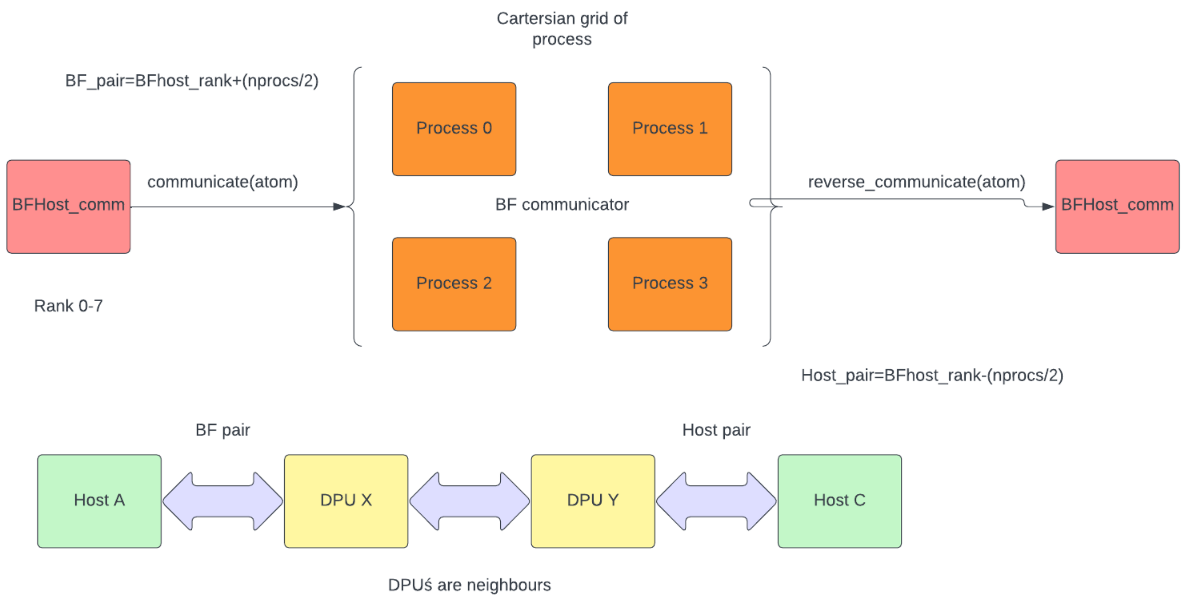
mpirun -np 4 -hostfile hostfile -bind-to cpu-list -map-by node miniMD-openmpi-x86-64 : -np 4 -hostfile BFfile -bind-to cpu-list -map-by node miniMD-openmpi-aarch64

This command creates a MPI communicator of size 8 i.e. 4 processes for Hosts and 4 processes for DPUs.


BF_Host_Comm i.e Global rank (0-7)
Host Comm (Global rank 0-3, Local rank 0-3)
BF Comm (Global rank 4-7, Local rank 0-3)

With the help of MPI, we can create two intra communicators ie one for host and one for DPU. It is the same case for multiple hosts and multiple DPUs. With the help of processor's name, we can identify which rank belongs to which host/ DPU, based on that intra communicators are formed.

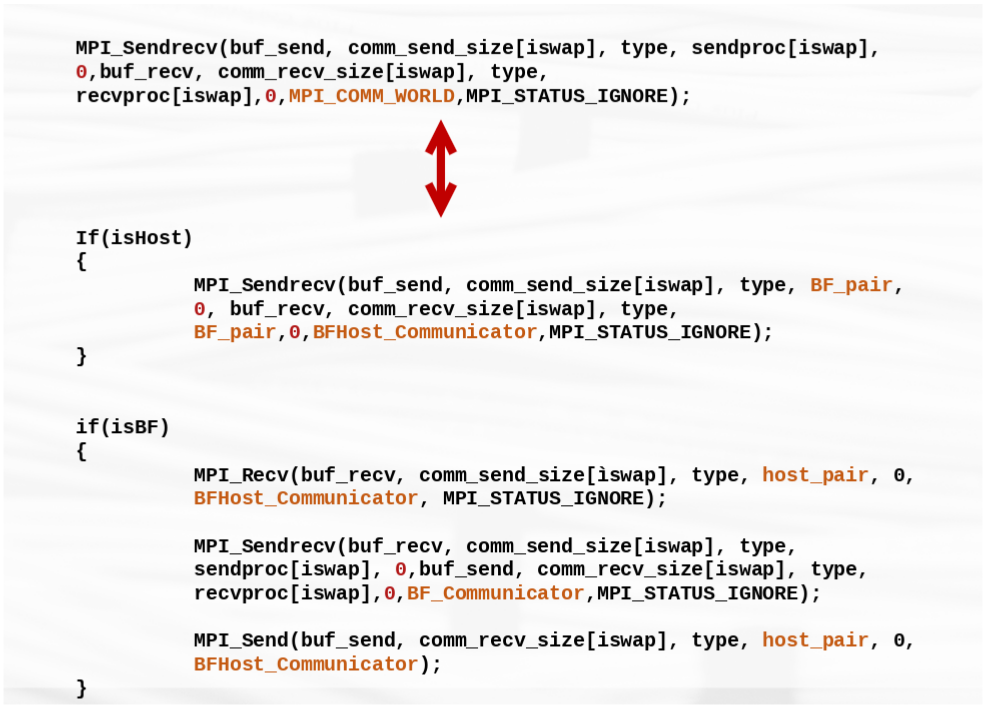
3.2 Offloading Communication



The figure above shows the communication schema for MPI offloading using DPUs. The main advantage of **DPU ie NVIDIA's BlueField2 card** is high networking and data movement performance when compared to the standard **Intel** server host. We are trying to leverage this in the above schema.

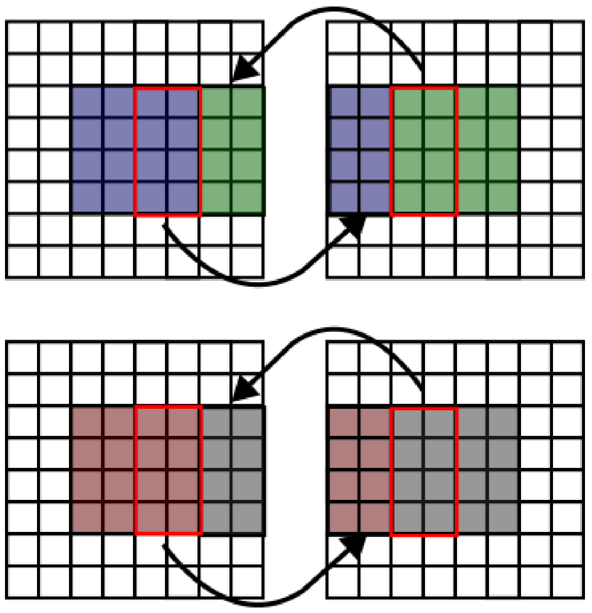
In this schema, one host is paired with one DPU based on its rank. The communicator **BFHostcomm** is an inter communicator between hosts and DPUs. With its help, the data is offloaded to it DPU pair. A cartesian grid is created on the communicator **BFcomm** which consists of processes that run on DPUs through which we can find the neighbouring process to which we need to communicate the **ghost/boundary cells** in the case of halo exchange. Once the exchange across neighbouring DPUs are done, the DPUs will send the received data back to its host pair. Here the communication across host neighbours happen through **DPU tunnel**.

3.2.1 Generic pattern for MPI Offloading Communication



```
MPI_Sendrecv(buf_send, comm_send_size[iswap], type, sendproc[iswap],
0,buf_recv, comm_recv_size[iswap], type,
recvproc[iswap],0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);



If(isHost)
{
          MPI_Sendrecv(buf_send, comm_send_size[iswap], type, BF_pair,
          0, buf_recv, comm_recv_size[iswap], type,
          BF_pair,0,BFHost_Communicator,MPI_STATUS_IGNORE);
}


if(isBF)
{
          MPI_Recv(buf_recv, comm_send_size[iswap], type, host_pair, 0,
          BFHost_Communicator, MPI_STATUS_IGNORE);

          MPI_Sendrecv(buf_recv, comm_send_size[iswap], type,
          sendproc[iswap], 0,buf_send, comm_recv_size[iswap], type,
          recvproc[iswap],0,BF_Communicator,MPI_STATUS_IGNORE);

          MPI_Send(buf_send, comm_recv_size[iswap], type, host_pair, 0,
          BFHost_Communicator);
}
```

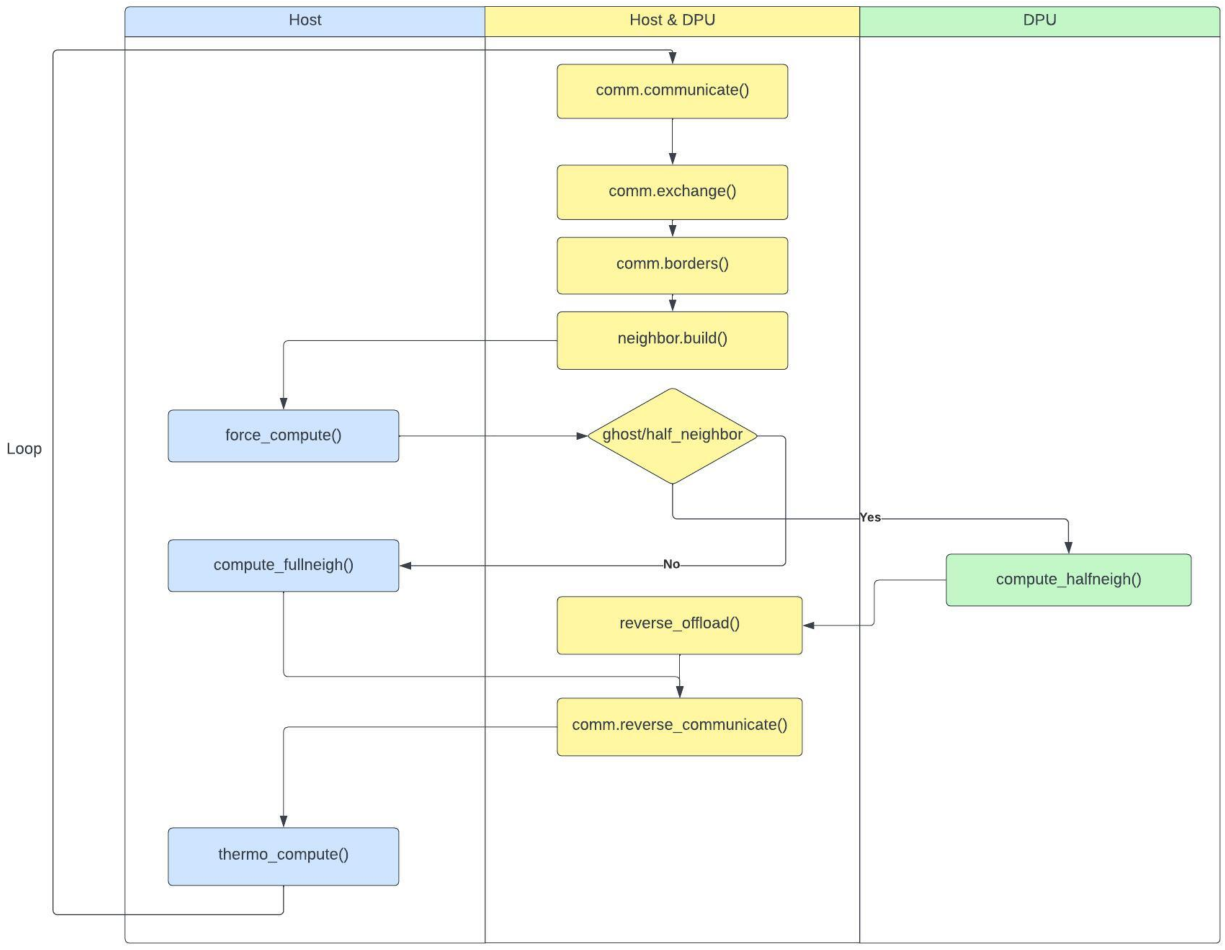
3.3 Offloading Computation



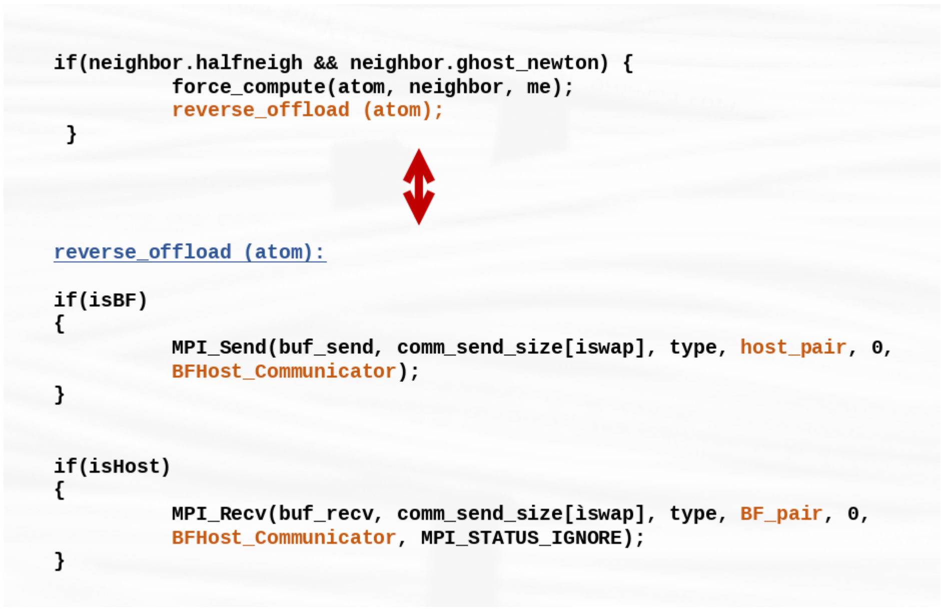
The computation can be divided into two sections:

- Halo / Boundary cells computation
- Interior cells computation

In the schema described below, we are offloading the computation of boundary/ghost cells to the DPU and the computation of interior cells are carried out in the host.



3.3.1 Generic pattern for MPI Offloading Computation



```
if(neighbor.halfneigh && neighbor.ghost_newton) {
          force_compute(atom, neighbor, me);
          reverse_offload (atom);
 }

reverse_offload (atom):

if(isBF)
{
          MPI_Send(buf_send, comm_send_size[iswap], type, host_pair, 0,
          BFHost_Communicator);
}


if(isHost)
{
          MPI_Recv(buf_recv, comm_send_size[iswap], type, BF_pair, 0,
          BFHost_Communicator, MPI_STATUS_IGNORE);
}
```

4 Analysis

The changed MiniMD code for MPI offloading is analysed with different baselines to get an overview of how the communication and computation time varies for different number of nodes and different input size. The time improvement is calculated by the following formula:

$$Time\,improvement = \frac{Total\ time\ of\ miniMD\ old\ code - Total\ time\ of\ miniMD\ offloading\ code}{Total\ time\ of miniMD\ old\ code}$$

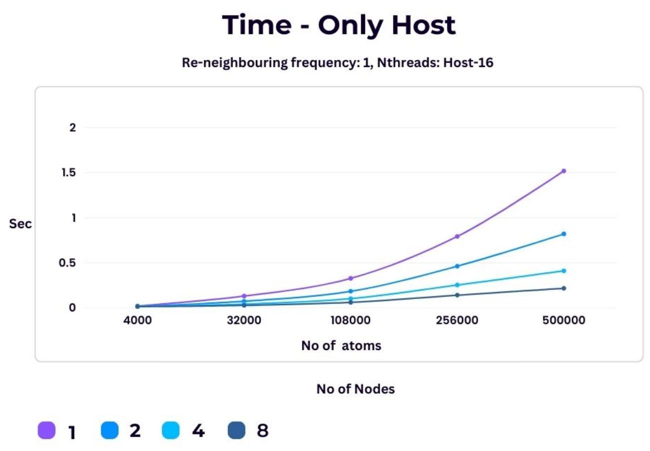
Parameters:

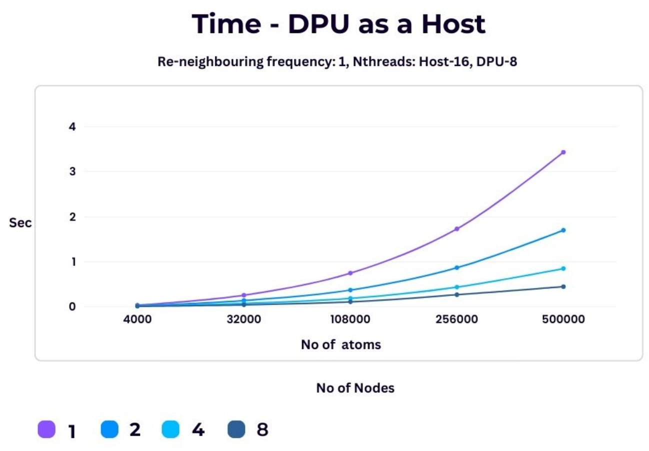
- **Only Host:** Old MiniMD code run on only CPUs
- **DPU as a Host:** Old MiniMD code run on both CPUs and DPUs
- **MPI manual offloading:** Modified MiniMD code as per the new schema
- **Input size:** 4000 to 500000 atoms
- **No of Host-DPU pairs:** 1,2,4,8
- **No of threads:** Host - 16, DPU - 8 (No of threads = No of cores per socket)
- **Re-neighboring frequency:** 1

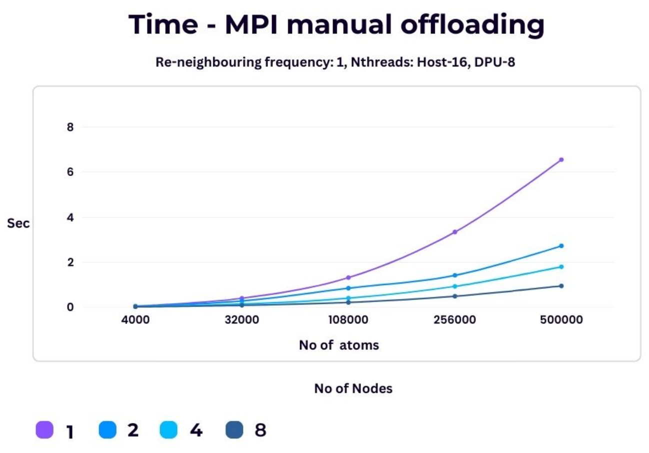
- **Nsteps:** 20
- **Machine:** Thor
- **ppn:**1

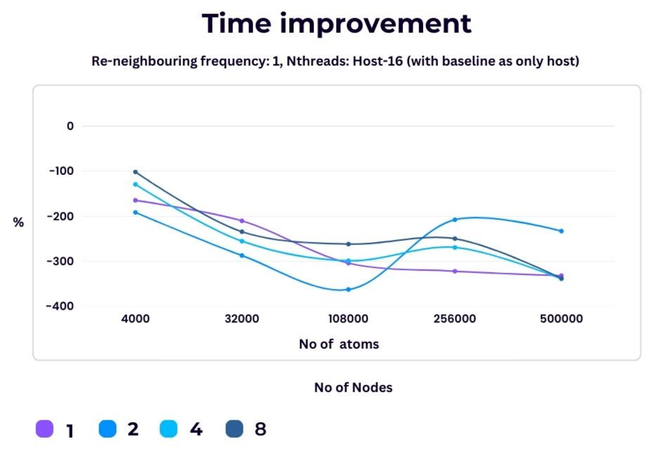
4.1 Analysis of total time

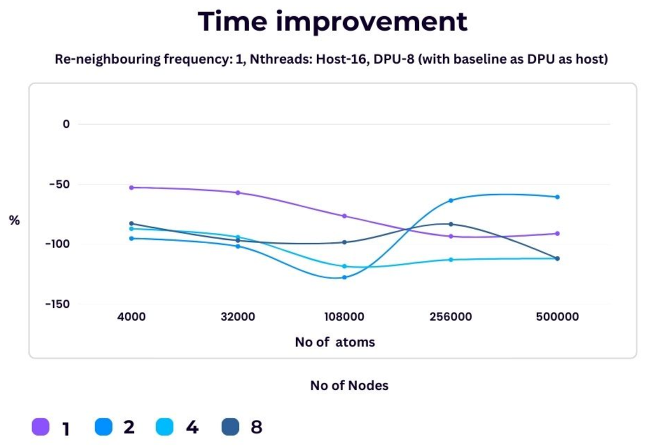
Total time: $Computation\ time + Communication\ time + Neighbor\ build\ time + Miscellaneous\ op\ time$





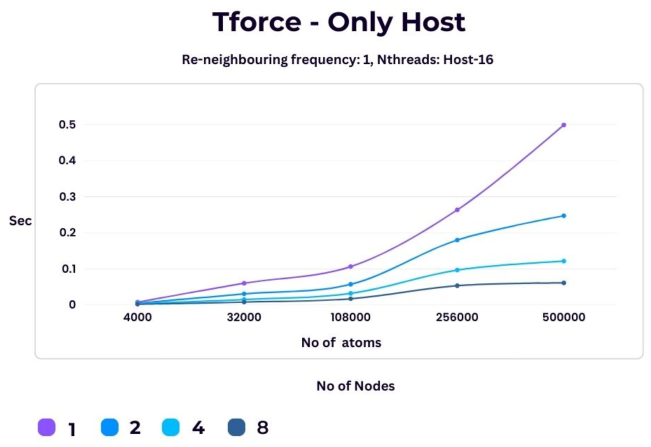

Time - MPI manual offloading
Re-neighbouring frequency: 1, Nthreads: Host-16, DPU-8
Sec
8
6
4
2
0
4000
32000
108000
256000
500000
No of atoms
No of Nodes
1
2
4
8

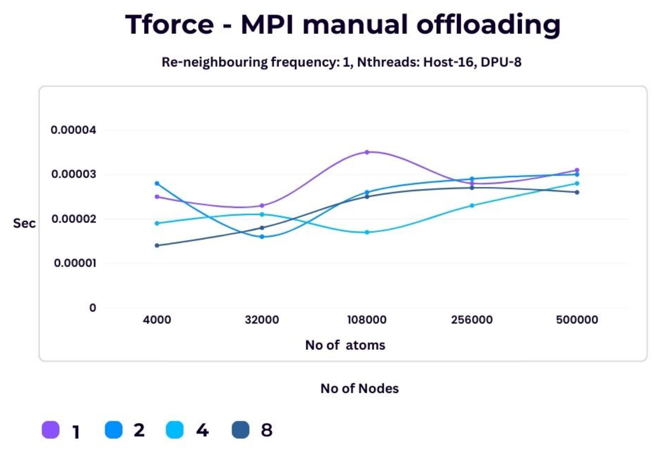

Time improvement
Re-neighbouring frequency: 1, Nthreads: Host-16 (with baseline as only host)
%
0
-100
-200
-300
-400
4000
32000
108000
256000
500000
No of atoms
No of Nodes
1
2
4
8

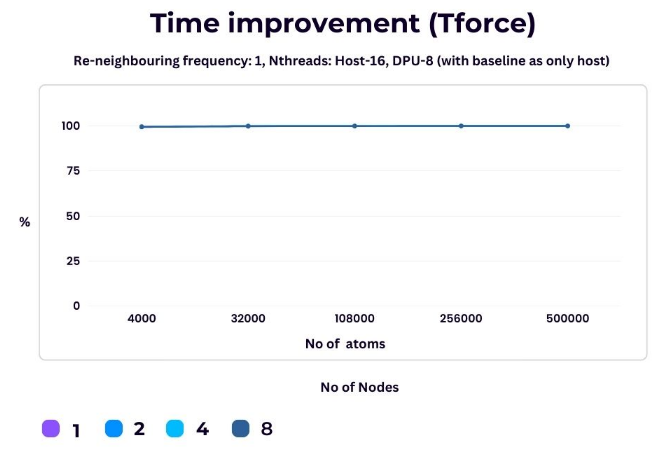


From the above given charts, we observe that the total time is scaling along with the input size and the performance is better in the case of more nodes. The total time in the case of only host and DPU as host performs better than the MPI manual offloading.

4.2 Analysis of Force computation time

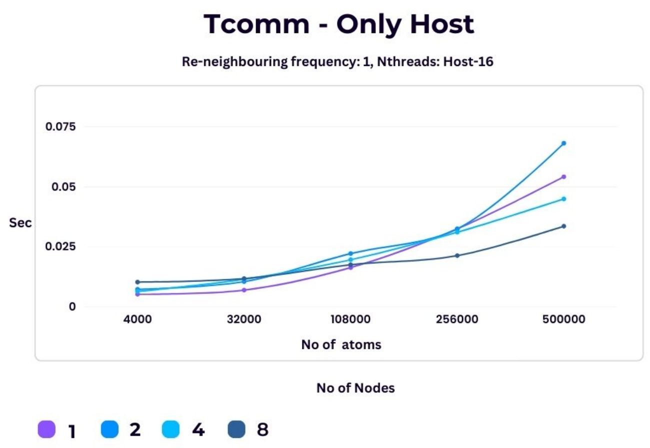


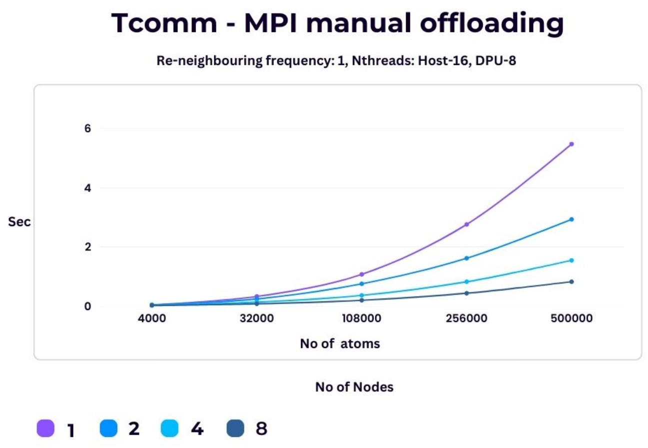


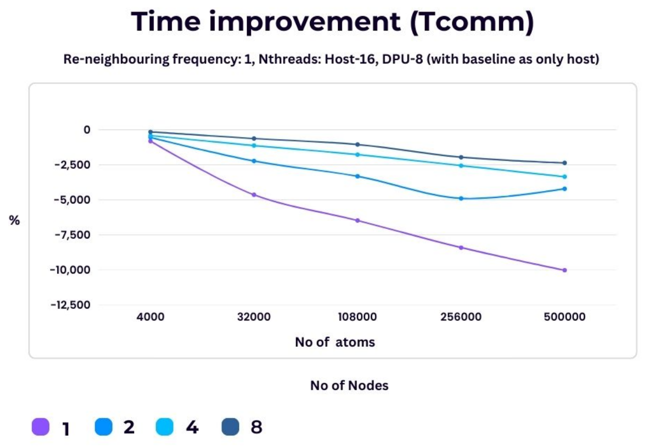


From the above given charts, we observe that the force computation time scales along with the input size in the case of only host and it remains within a particular range in the case of MPI manual offloading as the input size scales. The MPI manual offloading gives nearly 100 percent time improvement in the case of force computation.

4.3 Analysis of Communication time







From the above given charts, we observe that the communication time scales along with the input size for both the cases i.e. only host and MPI manual offloading. The communication time in the case of only host performs better than the communication time in the case of MPI manual offloading.

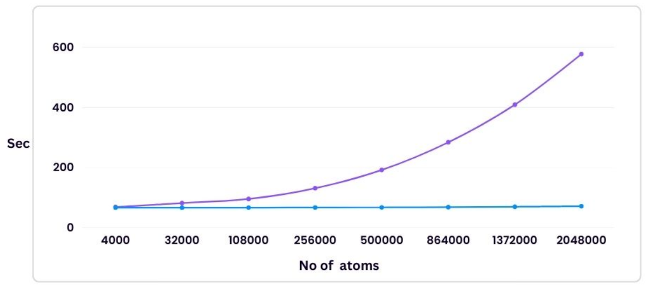
4.4 Analysis of time with increasing computational intensity

From the above analysis, we observe that the force computation time remains more less constant as the input size scales where as communication time scales along with the input size in the case of MPI manual offloading. We would be able to overlap the computation and communication time in the case of MPI manual offloading. If the application is computationally more intensive, the MPI manual offloading schema will perform better than the other two cases. The MiniMD application is not computationally intense to verify that. Hence, we are manually adding some additional time which is proportional to the input size in the case of force computation to observe the overlap we described above. The additional time is added in the following way:

```
double additional_time = 500000*timer.array[TIME_FORCE];
int t = 9999+additional_time;
std::this_thread::sleep_for(std::chrono::microseconds(t));
```

Ttotal

Re-neighbouring frequency: 1, Nthreads: Host-6, BF-6, Nodes: 4 with sleep()

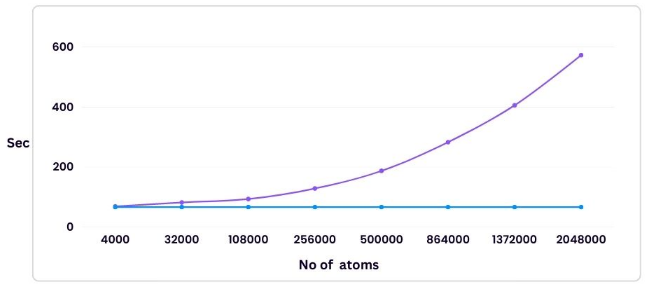


No of Nodes

● Host ● MPI Manual Offloading

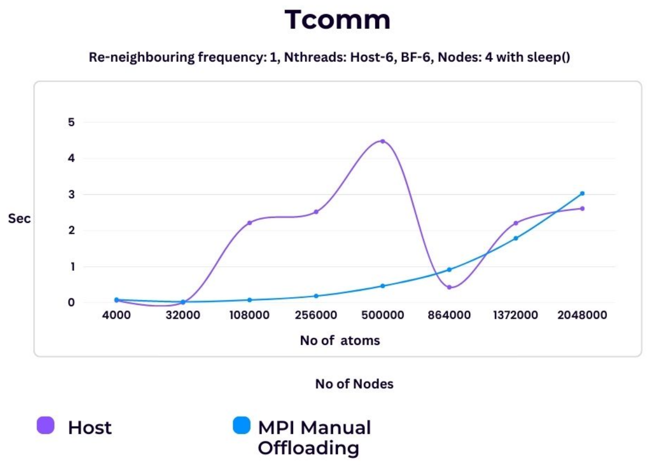
Tforce

Re-neighbouring frequency: 1, Nthreads: Host-6, BF-6, Nodes: 4 with sleep()



No of Nodes

● Host ● MPI Manual Offloading



We used **4 nodes on jupiter machine with ppn:1** for the above analysis. From the above given charts, the total time which is the combination of both computation and communication time performs better in the case of MPI manual offloading. Thus, we can observe the overlapping of communication and computation time in the case of MPI manual offloading.

5 Conclusion

From the analysis, we can observe that for the above described MPI manual offloading schema, there is a possibility of overlapping the communication and computation time. In the case of miniMD application, the computation is not heavy enough to overlap the communication time difference in comparison with the baseline i.e. miniMD code run just on host without any offloading. Hence, we manually added the time difference to computation time using sleep() function incrementally depending on the input size. It showed that the computation and communication time can be overlapped and the above explained MPI offloading schema made the miniMD application to perform better than the miniMD code running only on host without any offloading. **So in the case of computationally heavy applications, the above explained MPI offloading schema will perform better.**

6 Future Work

The future work involves using this MPI manual offloading schema as the baseline for the analysis of OpenMP pragma for DPU offloading that is currently under development involving LLVM and DOCA.

References

[1] Karamati, S., Hughes, C., Hemmert, K.S., Grant, R.E., Schonbein, W.W., Levy, S., Conte, T.M., Young, J. and Vuduc, R.W., 2022, May. “Smarter” NICs for faster molecular dynamics: a case study. In 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 583-594). IEEE.

[2] MiniMD base code github link

A Github links

- MiniMD DPU MPI manual offloading code
- MiniMD OpenMP DPU offloading code