Hierarchical Task Mapping on Dragonfly topology for scaling Molecular Dynamics

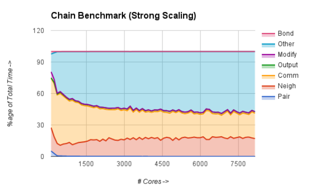
Sabyasachi Sahoo, Sathish S. Vadhiyar
Department of Computational and Data Sciences
Indian Institute of Science, Bangalore, India
sabyasachi@grads.cds.iisc.ac.in, vss@cds.iisc.ac.in

***Abstract*—Molecular Dynamics (MD) performs non-uniform computations and communications that vary over space and time. LAMMPS is one of the leading MD packages used extensively by molecular dynamics community. With the advent of exascale systems, it becomes important to ensure good scalability on large number of cores. Dragonfly topology is one of the most promising network topologies for the exascale era due to their scalability and cost. Despite having multiple advantages like non-minimal routing in Cray, communication is a major bottleneck for the scalability of LAMMPS. In this work, we propose strategies for efficient communications in LAMMPS MD simulations on dragonfly network topology. In particular, we propose partitioning-based task mapping algorithm, which along with considering the hierarchical nature of topology, also tries to account for non-minimal routing, non-contiguous job allocations and possible contentions at intra-node or intra-socket level. We compare our mapping with open source mapping software HierTopoMap and NUMA mapping from LAMMPS and show that our mapping algorithm reduces communication time by more than 15% in general and maximum of 44% on 8016 cores. While we had applied our strategy to dragonfly topology, our work is also applicable to general hierarchical topologies.**

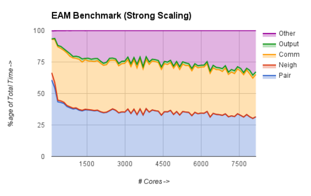
I. Introduction

The scientific community is interested in a variety of experiments that requires analysis at molecular level. Molecular Dynamics (MD) simulation provides a platform to conduct these experiments which are used for prediction of various complex properties of materials like density, heat of vaporisation [1] and also to understand the interactions between proteins and ligands [2]. LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and coarse-grained systems using a variety of force fields and boundary conditions. The major phases of any MD simulation are as follows [3]:

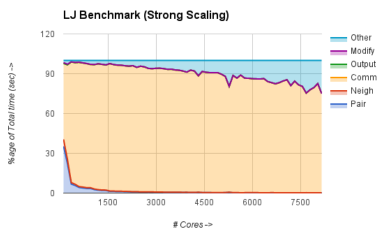
- **Pair:** all non-bonded force computation
- **Bond:** bonded interactions: bonds, angles, dihedrals, impropers
- **Kspace:** reciprocal space interactions: Ewald, PPPM, MSM
- **Neigh:** neighbor-list construction
- **Comm:** communicating atoms and their properties
- **Output:** writing dumps and thermo output
- **Modify:** fixes and computes called by them



(a) Chain benchmark



(b) EAM benchmark



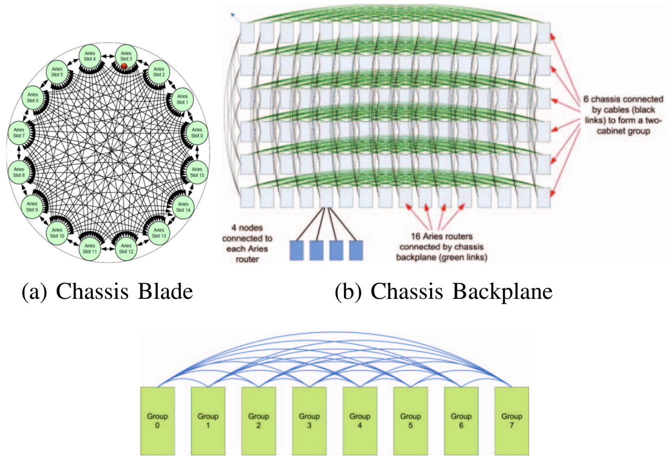
(c) LJ benchmark

Fig. 1: Percentage execution times for varying cores for LAMMPS

- **Other:** is the remaining time

Communication is usually one of the least scaling phases in MD whic also has been confirmed by our experiments on LAMMPS. We conducted scalability tests of LAMMPS for Chain, EAM and LJ benchmarks whose results are shown in figure 1. These plots show percentage execution time of LAMMPS with varying number of cores. For EAM and LJ benchmark, communication time clearly accounts for most of the total simulation time.

Network topology specifies the node layout in a supercomputer and plays a major role in communication time of an application. Dragonfly [4] is one such hierarchical network topology design which provides scalable global bandwidth while minimizing the number of expensive optical links. In dragonfly network, a group of routers, acting together as a single very-high radix router, pool their links, so that all the groups can maintain a network diameter of one. There are three chassis within a cabinet, each containing 16 blades. A 2D all-to-all structure is used within every two cabinets as shown in figure 2b. The chassis backplane provides connectivity in the first dimension (Green) and the backplane connects each Aries to each of the other 15 Aries within that chassis as shown in figure 2a. The second dimension (Black) consists of three links from each Aries within a chassis to its peer in each of the other chassis within that group. Groups connected via optical links are shown in figure 2c.


6 chassis connected by cables (black links) to form a two-cabinet group
4 nodes connected to each Aries router
16 Aries routers connected by chassis backplane (green links)
Group 0
Group 1
Group 2
Group 3
Group 4
Group 5
Group 6
Group 7

(a) Chassis Blade (b) Chassis Backplane

(c) DragonFly Groups

Fig. 2: DragonFly Network [4]

With the advent of exascale era, as the number of compute nodes increase, so does the size of interconnect network. Though dragonfly shows that it can minimize communication cost by efficient network topology, the communication performance can be pushed further by optimal placement of application processes to processors. Owing to sparse communication pattern in MD, intelligent placement of processes can result in larger performance gains. This optimal placement is called topology-aware task mapping, or *mapping*. Mapping is a NP-complete [5] problem and a number of heuristics have been proposed.

In this project, we optimize the communication times by intelligent mapping of the MD interactions in LAMMPS to Cray's dragonfly topology. To the best of our knowledge, we are the first group to show improvement in communication times by mapping on dragonfly topology. We also account for non-contiguous job allocation policies of Cray, and have shown improvement in all possible allocations. We use partitioning based task mapping and try to dynamically determine the depth of mapping (level in network hierarchy after which processes are randomly placed) for which contention should be minimum. We obtain an improvement of more than 15% in general over the default mapping of LAMMPS.

The rest of the report is divided as follows: Section II describes the recent work on mapping algorithms. Section III gives the mathematical and experimental reasons for the choice of our mapping algorithm and is explained in great detail. Section IV contains all the results obtained during the course of this project. Finally, in Section V we conclude with discussions on future work that can be performed on similar lines.

II. Related Work

MD simulations have regularly been used by scientists since 1950s. With the advent of vector machines and supercomputers, people have developed many MD software packages to bring down the total time to perform these simulations. There are more than 50 MD softwares, that are used for a variety of applications. In particular, LAMMPS, GROMACS and NAMD are leading packages used extensively by molecular dynamics community.

The most common mapping strategy in most MPI implementations, and also followed as default mapping in LAMMPS, was implemented in CHACO [6]. Baba et al. [7] implemented topology independent allocation system by evaluating several functions that represented properties of the graph. Agarwal et al. [8] demonstrated a process mapping strategy that minimizes total number of hop bytes communicated for torus and grid networks. Hoefler et al. [9] have discussed about several techniques and algorithms that efficiently address the issue of mapping. Deveci et al. [10] have developed geometric partitioning algorithm, where they recursively partition the communication and topology graph using multi-dimensional jagged algorithm to map nearby heavily communicating processes to nearby cores. Their algorithm also accounts for non-contiguous job allocation of Cray scheduler, but their work is applicable only for torus networks. They propose another algorithm [11] similar to greedy graph growing algorithm, which seeks to minimize weighted hops and maximum congestion in the network.

Abdel et al. [12] leverage the knowledge of routing algorithm to develop hierarchical mapping using linear programming, to limit the number of evaluations for possible mappings. Wu et al. [13] exploit the performance gap between inter- and intra-node communication, as well as between inter- and intra-socket communications to develop a hierarchical task mapping for cell based AMR cosmology simulations on 3D torus and fat tree topology. They later [14] incorporate neighbor joining algorithm [15] to detect fat tree topology. Jain et al. [16] analyze the behavior of dragonfly network for various routing strategies, job placement policies, and application communication patterns using network simulations. Prisacari et al. [17] have proposed a theoretical performance analysis framework that takes network specifications and traffic demand matrix as input, and predict bottlenecks and their impacts on bandwidth. Bhatele et al. [18] explore intelligent topology aware mappings of different communication patterns to physical topology to minimize link utilization on packet level network simulations of two-level direct networks like IBM's PERCS network, which is similar to dragonfly network. However, their mappings do not account for non-contiguous job allocations and mostly compare the effects of minimal and non-minimal routing on such networks. Their mapping follows a bottom-up approach wherein, they first cluster processes to form nodes, then further up, upto supernodes. With the motivation to reduce congestion in such networks, they bring randomisation at node levels to increase link utilization at higher levels, but still minimize inter-node communication. Due to this randomisation, their mapping reduces communication volume at node and to some extent at supernode level, but not at intermediate levels. They later show that use of non-minimal routing on default mapping gives similar performance as their mapping with randomisation.

Almost all of the works developed for Dragonfly networks have been performed as network simulations and there is no detailed discussion on any particular mapping strategy that could be useful on this network topology with non-minimal routing. Deveci et al. [10] have developed mapping algorithm that accounts for non-contiguous allocations of Cray systems, but their mapping algorithm and further optimizations are very much tailored for torus-based networks. In their another work [11], they suggest a mapping algorithm to minimize inter-node communication, irrespective of network hierarchy. They focus on minimizing metrics like total hops travelled by a message or reducing maximum congestion in a network, both of which have lower implications for Cray's Dragonfly network, owing to heterogeneity of the network and non-minimal routing respectively. Since dragonfly topology is hierarchical in nature, for better data locality, it is intuitive to implement hierarchical task mappings. Most hierarchical mappings have been developed and tested on non hierarchical topologies (like mesh or torus), which limits the extent to which the mapping algorithm can make use of hierarchical nature of dragonfly network.

Wu et al. [14] implement hierarchical mapping on fat tree topology which detects the hierarchical nature of topology and then performs the mapping based on a k-way unweighted graph partitioning by using METIS [19]. However, owing to the unweighted partitioning nature of METIS, low quality under- and over-balanced partitions are obtained for non-contiguous job allocations. As a result, they need to greedily move vertices to balance out. Also, using neighbor joining algorithm [15] they approximate the topology, whereas for Cray systems, accurate topology can be determined by considering coordinates of each node in the network. Another important point to note is that, they partition the node graph but not the application process graph. Owing to this coarseness, some processes will always be together in a node, irrespective of the hierarchy above them, which limits the optimal performance that could have been leveraged by efficient task to processor mapping. Moreover, their work pushes most of the communications to intra-node or intra-socket levels. This can result in unwanted message contention at lower levels of hierarchy. We account for these limitations in our mapping algorithm.

III. Methodology

Mapping problem can be formally stated as matching vertices of communication graph to topology graph so as to reduce communication time by minimizing some metrics like total hop counts, hop-bytes, etc. These metrics directly correlate with communication time for network topologies like mesh, torus or fat-tree networks, but this may not be true for a multi-level direct network like dragonfly. Table I shows the heterogeneity in communication for different levels of communication link in dragonfly topology. It implies that placement of processes needs to account for hops of different types, which wasn't the case for previously popular network topologies. Also, unlike previous static routing algorithms, Cray uses non-minimal routing algorithm which is also real-time congestion aware. Hence, metrics like maximum congestion [10], [11], [18] are also not applicable for these topologies. Unlike IBM's BlueGene systems, Cray allocates parallel MPI jobs in non-contiguous (sparse) fashion, where nodes from any portion of the machine can be allocated for a job without any regard to the allocated job's shape or locality. This means that MPI jobs requesting only 240 cores can end up performing at all levels of network hierarchy. Our mapping algorithm needs to account for all these factors.

Level	Latency ($\times 10^{-6} sec$)	Bandwidth (Gbps)
InterGroup (IG)	15.9322	0.2116
InterChassis (IC)	12.3422	1.5378
InterBlade (IB)	9.3519	1.43
InterNode (IN)	4.2312	3.8707
InterSocket(IS)	0.8242	6.1979
IntraSocket(IaS)	0.3632	6.382

TABLE I: Average Latency and Bandwidth constants

It is quite intuitive to develop a mapping algorithm that performs smaller message size communications at higher level of hierarchy and larger message sizes at lower levels. Let this intuition based mapping be called as $mapHier$. It is clear that $mapHier$ will perform better than any other strategy that sends larger message sizes at higher levels of hierarchy. Hence, we propose partitioning-based hierarchical mapping, where we partition the communication graph (where each vertex represents a process and edge-weight represents communication volume between any two processes) at each level, recursively. Partitioning at any level gives us the minimum edgeweight-cut which ensures that the messages of smaller sizes are communicated in between the partitions. This is done hierarchically in a recursive fashion (top-down approach) to ensure that the higher levels are given more priority than the lower levels.

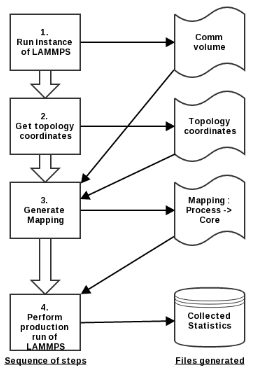
A. *Weighted-Partitioning based Hierarchical Task mapping*

An entity in a dragonfly network can be a collection of cores, nodes, blades, etc. Network diameter of one is maintained at each level, but same is not true across the levels. As a consequence, our experiments (figure 5) show that there is as much as 1.2-1.5 times communication time gap between any two subsequent levels of dragonfly topology. Hence, we want to minimize amount of communication volume at each level, starting from the top level. Reducing this metric also results in link de-congestion at higher levels (as for same higher message sizes, it is mapped to lower levels of network hierarchy), resulting in overall faster communication.

Our mapping algorithm shown in algorithm 1 takes two inputs, namely, network topology coordinates, corresponding to each MPI rank and communication volume among processes. Each process in LAMMPS communicates with six neighboring processes in a sequential fashion, such that each process ends up sending (and receiving) data from adjacent twenty-six processes of the cube. The communication volume among the processes is collected by running an instance of LAMMPS simulation with same number of atoms and iterations and default parameters. By directly reading the topology coordinates, we inherently account for non-contiguous job allocation of MPI processes. Figure 3 shows a flow chart of the job script, that contains the flow of the various steps corresponding to a single job submission. Our mapping algorithm works in three phases as follows:

1) First, it reads the topology coordinates and forms the topology tree. The topology tree maybe skewed or balanced, based on the job allocation by Cray ALPS.

2) Next, it reads the communication volume among the processes, and accordingly forms communication graph of processes with communication volume as the edge weight among the processes.
3) Finally, we recursively partition the graph based on the topology to obtain the mapping of processes to cores.



1. Run instance of LAMMPS
Comm volume
2. Get topology coordinates
Topology coordinates
3. Generate Mapping
Mapping : Process -> Core
4. Perform production run of LAMMPS
Collected Statistics
Sequence of steps
Files generated

Fig. 3: Job script for generating mapping

Algorithm 1 Hierarchical Task Mapping Algorithm

Input: Communication volume file ($cvFile$), Topology coordinates file ($tFile$), mapping Depth (d)
Output: Mapping file for LAMMPS ($mapFile$)

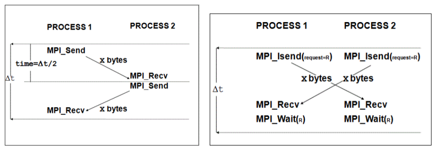
```
1: procedure GenerateMapping($cvFile, tFile, d$)
2:     $G \leftarrow formCommunicationGraph(cvFile)$
3:     $tHead \leftarrow formTopologyTree(tFile)$
4:     $M \leftarrow RecursiveMapping(G, tHead, d)$
5:     $mapFile \leftarrow generateOutput(M)$
6:     return $mapFile$
7: end procedure
8: procedure RecursiveMapping($G', \xi, d$)
9:     if $d = currentLevel$ then
10:        return mapping $m : v \in G' \rightarrow \xi \in T$
11:    end if
12:    for each sibling $s$ at level of entity $\xi$ in topology tree $T$ do
13:        $ratio[s] = |cores$ in $s|$ / $|cores$ in $\xi|$
14:    end for
15:    $P = Patoh(G', ratio)$
16:    for each partition $p \in P$ corresponding to entity $s$ do
17:        Form new comm graph $g' \leftarrow p$
18:        $RecursiveMapping(g', s, d)$
19:    end for
20: end procedure
```

The third phase is the most important phase, where we perform recursive partitioning of the communication graph using Patoh [20] graph partitioning tool. Patoh can generate weighted partitions for any given hyper-graph. Most popular mapping algorithms [13], [14] first form the node graph and then partition it with the objective of minimizing inter-node communication. As a result, processes in a node vertex will always be together, irrespective of the hierarchy in network topology. But, since we partition the process graph based on different levels of hierarchy, we ensure finer granularity of the mapping generated and provide more room for generation of flexible and optimal mapping. Also, we can perform partitioning till certain level of dragonfly topology and arrange the processes in the order of their ranks for later levels.

The steps performed at each level of recursion have been shown in detail in procedure $RecursiveMapping$. At each recursion level, we first find the ratio in which the communication graph is to be partitioned, based on the number of entities present at that level and the number of processes present per entity, for each of those entities. The partitions obtained from Patoh can be unbalanced sometimes (due to round off errors in ratio), and hence balancing is performed by greedily moving vertices from overloaded partitions to underloaded partitions. Next, we form communication graph from each of the partitions of the communication graph and recurse on the newly formed graph for the entity at next lower level. The recursion stops if we reach the lowest level of the network topology or the level till which the partitioning is desired.

B. Contention Aware Dynamic Mapping Algorithm

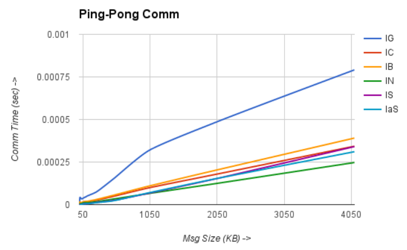
Figure 5 shows average communication time for a process communicating at a certain level, for varying message sizes. In Ping-Pong communication pattern, the communications take place in serial involving two communication processes, whereas in Ping-Ping communication pattern, two processes communicate in parallel with each other, as shown in figure 4. These plots were generated from the code developed in-house (similar to [21]), performing these communication patterns. It brings out two important characteristics of dragonfly network:



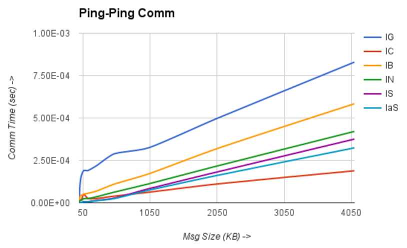
(a) Ping Pong (b) Ping Ping

Fig. 4: Communication Pattern [21]

- **Heterogeneity** : Unlike most popular networks, a single hop from one process to another in a dragonfly network can have different communication times (also clear from table I). Therefore, mapping algorithms trying to minimize hop-count, hop-bytes, etc., fail for such networks.
- **Contention effects** : Communication time in a lower level link may be more than a higher level link for message sizes above a threshold. The threshold can depend on a number of factors like node architecture, instantaneous network congestion, etc. The graph helps us understand that pushing all the messages to lower levels can result in increased communication time primarily due to socket-level contentions. Previous works have shown [13] that this can be due to smaller memory bandwidth per core or inefficient implementation of intra-node communication algorithms in mvapich library. Since LAMMPS follows more of a Ping-Ping fashioned communication pattern, the threshold will also be lower (from figure 5b), implying even severe contentions.



(a) Ping Pong



(b) Ping Ping

Fig. 5: Communication Time for varying message sizes

We have taken care of heterogeneity by making our algorithm take into account the hierarchical nature of network. We can reduce contention by restricting partitioning to a certain level from top, also called as mapping depth. Figure 6 shows that mapping depth of 2 (applying mapping till blade level and then map the processes according their MPI rank, below it) gives maximum percentage improvement on maximum communication time. It shows that taking care of contention can result in higher performance. But, it will be difficult for any user to determine the correct mapping depth for his set of experiments, as it may depend on a number of factors discussed above. Hence, we propose algorithm 2 to dynamically determine mapping depth by approximately simulating the communication pattern of LAMMPS, which yields correct mapping depth in most of the cases.

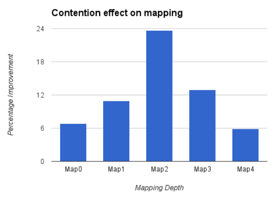


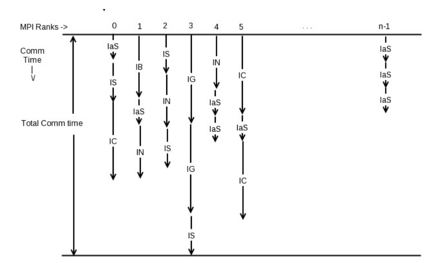
Fig. 6: Percentage improvement on communication time for varying mapping depth for 240 cores

The processes in LAMMPS communicate with six of its adjacent neighbors and all these six steps of communication take place in parallel in each iteration of simulation. The simulation of LAMMPS communication (in algorithm 2) is not exactly as performed in LAMMPS, but an approximation of actual communication. In LAMMPS, communication takes place in parallel for the given number of iterations. But, we consider the average communication volume transferred between any two processes and simulate it in parallel in a single iteration of communication (which constitutes communicating to six of its neighbors).

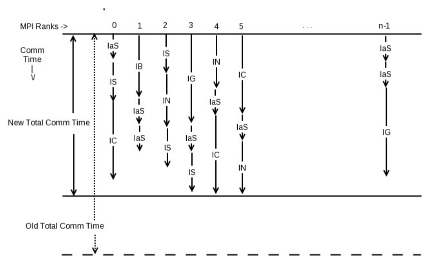
Algorithm 2 Dynamic Hierarchical Task Mapping Algorithm

Input: Communication graph G, Topology Tree T
Output: Mapping M : Process $P \rightarrow$ Core C

```
1:  procedure ContentionAwareMapping($G, T$)
2:      $t = DBL\_MAX$
3:      for each mappingDepth $d$ do
4:          $M' = RecursiveMapping(G, T, d)$
5:          $t' = SimulateCommTime(M')$
6:          // choose $M'$ corresponding to min time $t'$
7:          if $t' < t$ then
8:              $t = t'$
9:              $M \leftarrow M'$
10:         end if
11:     end for
12:     return $M$
13: end procedure
```



(a) Default

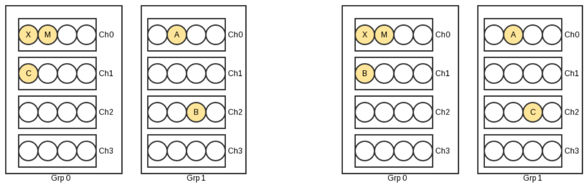


(b) Load Balanced

Fig. 7: Parallel communications in MD

C. Communication Load Balancing based Hierarchical Task mapping

Figure 7a (considering every process performs only three communications) shows pictorially what affects the total communication time. It might seem that another method of reducing communication time would be to minimize number of inter-partition(or entity) communications (at higher levels) per core, by swapping higher and lower levels of communications for those processes, that perform more than one high level communications (like IG or IC). For example, if a communication-load balancing mapping algorithm, say



(a) $mapHier$ mapping (b) $mapLB$ mapping

Fig. 8: Two different types of mapping for proof

$mapLB$, generates a mapping such that IG communication from rank 3 is moved to rank $n-1$ and IaS communication from rank $n-1$ to 3. Figure 7b illustrates new total communication time, due to mapping $mapLB$.

$$Minimize\ (\max_i \sum_j (l_j + (m/b_j))) \tag{1}$$

Mapping problem can be formalized as a minimization problem as shown in equation 1. l_j and b_j are average latency and bandwidth constants for a link at level j and we perform maximization over all processes. Let process X communicate with processes A and B, with message sizes m_a and m_b, respectively, such that $m_a \leq m_b$. Figure 8a and 8b shows network topology (depicting only Group (Grp), Chassis(Ch) and Core levels(as circles) for simplicity) for a given job allocation (allocated nodes shown in yellow), with corresponding mapping of processes by $mapHier$ and $mapLB$ algorithms.

In $mapHier$ mapping, X will communicate at level 0 to process A and B and say message m_c is the smallest message to be communicated at level 1 by some process M to C. Since message m_c is communicating at level 1 in $mapHier$ so, $m_a \leq m_b \leq m_c$. In fact, for most practical purposes $m_a < m_b << m_c$. Mapping $mapLB$ will first make process X communicate to A at level 0. But as it favors minimizing number of inter-partition communications per core and since the number of communications is already at a minimum (due to partitioning) in $mapHier$, the placement of processes B and C will be interchanged. As a result, process X ends up doing only one IG communication and message m_c is communicated at level 0 instead of level 1. Let t_{ml} and t_{mh} be total communication time for $mapLB$ and $mapHier$, respectively. So, total communication time for both types of mapping will be

$$\begin{aligned}
t_{mh} &= max\ (t_{mh}^{X}, \ldots, t_{mh}^{C}, \ldots) \\
t_{mh} &= max\ (t_{mh}^{XA} + t_{mh}^{XB}, \ldots, t_{mh}^{MC}, \ldots) \\
t_{mh} &= max\ (l_0 + \frac{m_a}{b_0} + l_0 + \frac{m_b}{b_0}, \ldots, l_1 + \frac{m_c}{b_1}, \ldots) \\
&and \\
t_{ml} &= max\ (t_{ml}^{X}, \ldots, t_{ml}^{C}, \ldots) \\
t_{ml} &= max\ (t_{ml}^{XA} + t_{ml}^{XB}, \ldots, t_{ml}^{XC}, \ldots) \\
t_{ml} &= max\ (l_0 + \frac{m_a}{b_0} + l_0 + \frac{m_c}{b_0}, \ldots, l_1 + \frac{m_b}{b_1}, \ldots)
\end{aligned}$$

Communication time for both mapping can be compared by

$$t_{ml} - t_{mh} = max\ (\frac{m_c - m_b}{b_0}, \frac{m_b - m_c}{b_1})$$

Now, $b_0 < b_1$ and $m_c >> m_b$. Therefore, for process X, $t_{ml} >> t_{mh}$ and vice-versa for process M. It implies that $mapLB$ considered process X to be one of the candidate for slowest process in $mapHier$ mapping and tried to load balance its communications. But, in the process, $mapLB$ is increased the communication time of process X. If process X was indeed the slowest process, then $mapLB$ has infact increased the total communication time. Hence, we have proven by contradiction that, $mapLB$ fails in these scenarios w.r.t. $mapHier$. Thus, a strategy that just considers load balancing the communication levels across processes may not generate optimal mapping in some cases.

IV. Experiments and Results

A. Experiment Setup

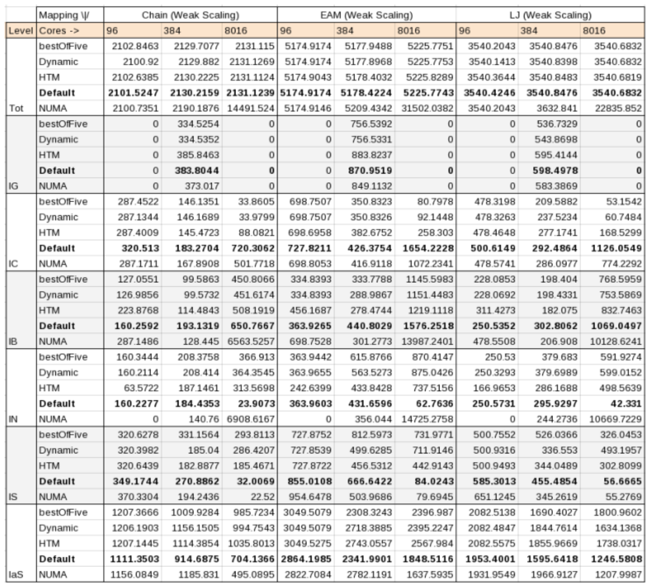
We are working on our Cray XC40 (also called as Sahasrat) at SERC IISc with following specifications:

- Intel Haswell 2.5 GHz based CPU cluster with 1376 nodes
- Each node has 2 CPU sockets with 12 cores each (i.e. 24 cores per node), 128GB RAM
- Connected with Aries high speed interconnect on a dragonfly topology

B. Results

We ran weak scaling experiments with 24-8016 processors (1-334 nodes) on Sahasrat for all the three benchmarks, namely, Chain (bead-spring polymer melt of 100-mer chains), LJ (atomic fluid with Lennard-Jones potential) and EAM (metallic solid). The number of atoms varied from 768K in 24-core run to 25.65M in 8016-core run. For each experiment, we obtained a node allocation of the requested size, and ran all mapping methods with their corresponding production run of LAMMPS simulation, for all the three benchmarks within that allocation. We repeated each production run five times, for each mapping, to average over different initial point of LAMMPS simulation. However, we have conducted our experiment only once, as a result of which, effects of different topology and instantaneous network congestion on percentage improvement haven't been nullified fully. The results for best out of the five mapping depth is represented by $bestOfFive$, which is generated to show how closely our dynamic algorithm (represented by $Dynamic$) predicts the correct mapping depth. We compare our $bestOfFive$ and $dynamic$ with HierTopoMap [14] (represented by HTM) and $NUMA$ mapping of LAMMPS. $NUMA$ mapping insures that contiguous subsections of the 3d grid of tasks are assigned to all the cores of a node by using MPI cartesian routines. In some sense, comparing with $NUMA$ is equivalent to comparing with the default mapping methods of MPI.

At the beginning of this report, we developed a partitioning-based mapping algorithm (algorithm 1) that seeks



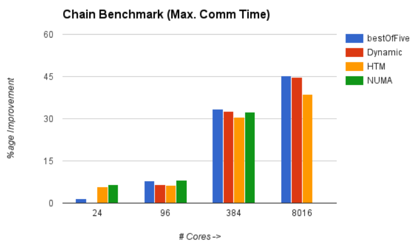
	Mapping \\|/	Chain (Weak Scaling)			EAM (Weak Scaling)			LJ (Weak Scaling)		
Level	Cores ->	96	384	8016	96	384	8016	96	384	8016
	bestOfFive	2102.8463	2129.7077	2131.115	5174.9174	5177.9488	5225.7751	3540.2043	3540.8476	3540.6832
	Dynamic	2100.92	2129.882	2131.1269	5174.9174	5177.8968	5225.7753	3540.1413	3540.8398	3540.6832
	HTM	2102.6385	2130.2225	2131.1124	5174.9043	5178.4032	5225.8289	3540.3644	3540.8483	3540.6819
	Default	**2101.5247**	**2130.2159**	**2131.1239**	**5174.9174**	**5178.4224**	**5225.7743**	**3540.4246**	**3540.8476**	**3540.6832**
Tot	NUMA	2100.7351	2190.1876	14491.524	5174.9146	5209.4342	31502.0382	3540.2043	3632.841	22835.852
	bestOfFive	0	334.5254	0	0	756.5392	0	0	536.7329	0
	Dynamic	0	334.5352	0	0	756.5331	0	0	543.8698	0
	HTM	0	385.8463	0	0	883.8237	0	0	595.4144	0
	Default	0	**383.8044**	**0**	0	**870.9519**	**0**	0	**598.4978**	**0**
IG	NUMA	0	373.017	0	0	849.1132	0	0	583.3869	0
	bestOfFive	287.4522	146.1351	33.8605	698.7507	350.8323	80.7978	478.3198	209.5882	53.1542
	Dynamic	287.1344	146.1689	33.9799	698.7507	350.8326	92.1448	478.3263	237.5234	60.7484
	HTM	287.4009	145.4723	88.0821	698.6958	382.6752	258.303	478.4648	277.1741	168.5299
	Default	**320.513**	**183.2704**	**720.3062**	**727.8211**	**426.3754**	**1654.2228**	**500.6149**	**292.4864**	**1126.0549**
IC	NUMA	287.1711	167.8908	501.7718	698.8053	416.9118	1072.2341	478.5741	286.0977	774.2292
	bestOfFive	127.0551	99.5863	450.8066	334.8393	333.7788	1145.5983	228.0853	198.404	768.5959
	Dynamic	126.9856	99.5732	451.6174	334.8393	288.9867	1151.4483	228.0692	198.4331	753.5869
	HTM	223.8768	114.4843	508.1919	456.1687	278.4744	1219.1118	311.4273	182.075	832.7463
	Default	**160.2592**	**193.1319**	**650.7667**	**363.9265**	**440.8029**	**1576.2518**	**250.5352**	**302.8062**	**1069.0497**
IB	NUMA	287.1486	128.445	6563.5257	698.7528	301.2773	13987.2401	478.5508	206.908	10128.6241
	bestOfFive	160.3444	208.3758	366.913	363.9442	615.8766	870.4147	250.53	379.683	591.9274
	Dynamic	160.2114	208.414	364.3545	363.9655	563.5273	875.0426	250.3293	379.6989	599.0152
	HTM	63.5722	187.1461	313.5698	242.6399	433.8428	737.5156	166.9653	286.1688	498.5639
	Default	**160.2277**	**184.4353**	**23.9073**	**363.9603**	**431.6596**	**62.7636**	**250.5731**	**295.9297**	**42.331**
IN	NUMA	0	140.76	6908.6167	0	356.044	14725.2758	0	244.2736	10669.7229
	bestOfFive	320.6278	331.1564	293.8113	727.8752	812.5973	731.9771	500.7552	526.0366	326.0453
	Dynamic	320.3982	185.04	286.4207	727.8539	499.6285	711.9146	500.9316	336.553	493.1957
	HTM	320.6439	182.8877	185.4671	727.8722	456.5312	442.9143	500.9493	344.0489	302.8099
	Default	**349.1744**	**270.8862**	**32.0069**	**855.0108**	**666.6422**	**84.0243**	**585.3013**	**455.4854**	**56.6665**
IS	NUMA	370.3304	194.2436	22.52	954.6478	503.9686	79.6945	651.1245	345.2619	55.2769
	bestOfFive	1207.3666	1009.9284	985.7234	3049.5079	2308.3243	2396.987	2082.5138	1690.4027	1800.9602
	Dynamic	1206.1903	1156.1505	994.7543	3049.5079	2718.3885	2395.2247	2082.4847	1844.7614	1634.1368
	HTM	1207.1445	1114.3854	1035.8013	3049.5275	2743.0557	2567.984	2082.5575	1855.9669	1738.0317
	Default	**1111.3503**	**914.6875**	**704.1366**	**2864.1985**	**2341.9901**	**1848.5116**	**1953.4001**	**1595.6418**	**1246.5808**
IaS	NUMA	1156.0849	1185.831	495.0895	2822.7084	2782.1191	1637.5935	1931.9549	1966.9127	1207.9987

Fig. 9: Comm Volume (MB) for different mapping at different levels for varying cores

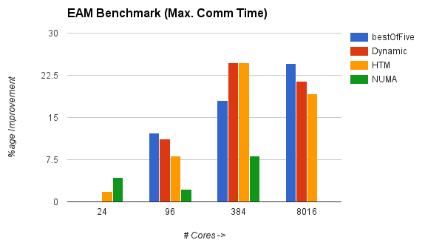
to minimize average communication volume at each level of network hierarchy, in a top down fashion, giving priority to higher levels. As a result, most volume of communication would be pushed to lower levels resulting in contention, as discussed in section III-B. So, we decided to stop the partitioning after a suitable mapping depth, which led to formulation of contention-aware dynamic algorithm (algorithm 2). As a result of this, most of the communication volume was shifted from higher levels to lower levels, till mapping depth. Hence, we expect to see average communication volume for the optimal mapping method at higher levels of network to be much lower than default mapping, but higher for lower levels upto certain extent (to account for both the factors discussed above). Same is also conveyed from table in figure 9.

Figure 9 shows a table of average communication volume (in MB) for different mappings, at each level of dragonfly network hierarchy and overall (represented by Tot in table), for all the three benchmarks for varying cores. Some values in IG are zero because no processes were doing inter-group communication due to node allocation. We can see that for higher levels of network (like IG and IC), our mapping algorithms, $bestOfFive$ and $Dynamic$, bring maximum reduction of communication volume over $Default$ mapping in all the cases and perform better than HTM and $NUMA$ in almost all the cases. At lower levels (like IS and IaS) too, our algorithms carefully increase the communication volume upto a certain point w.r.t. $Default$, to account for the effects of contention. Hence, theoretically, we generate better task-mapping than HTM and $NUMA$. Similar results were obtained from the experimental runs.

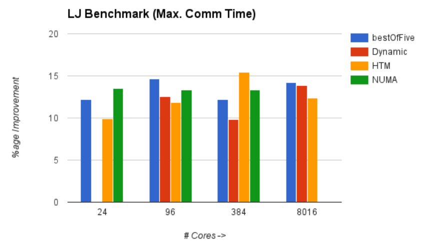
Figure 10 shows percentage improvement in weak scaling for maximum communication time of LAMMPS simulation. Compared to default mapping, our algorithm with the best mapping depth, $bestOfFive$ performs better than HTM and $NUMA$ mapping of LAMMPS in most core counts, getting a maximum performance improvement of 45% for maximum



(a) chain



(b) eam



(c) lj

Fig. 10: Percentage Improvement on Max. Communication Time for varying nodes for LAMMPS

communication time and 7.5% for total simulation time on 8016 cores. Our dynamic algorithm, $Dynamic$ also attained similar percentage improvement as $bestOfFive$, reducing maximum communication time by 44% and total simulation time by 7%. Theoretically, $bestOfFive$ should perform better than all others, at-least better than $Dynamic$ since dynamic algorithm tries to predict the best mapping depth and then generated appropriate mapping. But, it is found to be not true for some core counts (like for 384 cores). It is because we have collected these results from only a single set of experiment and as mentioned previously, our results currently are not quite independent of the effects of the instantaneous network congestion and the topology of cores allocated to our jobs. Repeating and averaging these set of experiments can help us mitigate such effects.

V. Conclusion and Future Work

Multi-level direct networks like Cray's dragonfly network and IBM's PERC network are some of the most important network design which will be used in building next generation of supercomputers in exa-scale era. Though these networks adopt better communication friendly designs and algorithms than its predecessor designs, communication may still become one of the major bottlenecks for many parallel applications with default MPI-rank ordered mapping. In this project, we have developed a dynamic hierarchical task mapping algorithm which gives a maximum improvement of 44% for reducing maximum communication time. Our mapping algorithm accounts for heterogeneous and hierarchical nature of network, sparse job allocations, non-minimal routing, and network noise to a certain extent. It is one of the first attempts of mapping (to the best of our knowledge) on such next generation networks, also laying the basic theoretical foundation for mapping on such networks. We show that conventional metrics like total hop count and maximum congestion have less relevance in such networks. Though we have considered LAMMPS as application and dragonfly network for our experiments, this mapping algorithm is also applicable for other popular MD packages and other hierarchical networks, respectively.

We have shown that theoretical models can also be used to theoretically test the mapping algorithms that may arise from some intuitions, before actually experimenting them. Hence, predictions based on the theoretical model is highly dependent on reliability of the model, or how exactly the model mimics the real network. As a future work, probabilistic routing function (Valiant's routing algorithm [22]), congestion prediction, variation of communication time at a specific level, network noise function, contention function, etc. can be modelled for even more practical model of dragonfly network. Using these methods can also help us develop better dynamic algorithms, that can accurately predict the mapping depth that will give us optimal performance.

The communication volume throughout the molecular dynamics simulation varies w.r.t. time and is unpredictable in nature. Sometimes, communication volume between two processes for two different simulations, for same molecular system can be very different (as it will depend on their respective initial points, which in practical system is usually random). Hence, mapping will have to be generated on the fly, as the simulation proceeds. This class of methods for mapping are called dynamic remapping, and is proposed as a future work. We can also use neighbor joining algorithm to automatically detect the underlying network and then accordingly generate required mapping even for non-hierarchical networks. The collective work can be put together in the form of open source library like LibTopoMap [5] and can be made even more generic, by taking into account recent developments in node architecture, network designs, routing functions and mapping algorithms. It can be used as a generic mapping tool on any network, for applications with any type of communication pattern and to drastically reduce total communication time.

References

[1] J. Wang and T. Hou, "Application of molecular dynamics simulations in molecular property prediction. 1. density and heat of vaporization," *Journal of chemical theory and computation*, vol. 7, no. 7, pp. 2151–2165, 2011.

[2] G. P. Pieffet, "The application of molecular dynamics simulation techniques and free energy," Ph.D. dissertation, University of Groningen, 2005.

[3] "2. Getting Started - LAMMPS documentation," http://lammps.sandia.gov/doc/Section_start.html#lammps-screen-output, accessed: 2016-5-20.

[4] G. Faanes, A. Bataineh, D. Roweth, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, J. Reinhard *et al.*, "Cray cascade: a scalable hpc system based on a dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 103.

[5] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the international conference on Supercomputing*. ACM, 2011, pp. 75–84.

[6] B. Hendrickson and R. Leland, "The chaco users guide: Version 2.0," Technical Report SAND95-2344, Sandia National Laboratories, Tech. Rep., 1995.

[7] T. Baba, Y. Iwamoto, and T. Yoshinaga, "A network-topology independent task allocation strategy for parallel computers," in *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 1990, pp. 878–887.

[8] T. Agarwal, A. Sharma, and L. V. Kalé, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[9] T. Hoefler, E. Jeannot, and G. Mercier, "An overview of process mapping techniques and algorithms in high-performance computing," *High Performance Computing on Complex Environments*, pp. 75–94, 2014.

[10] M. Deveci, S. Rajamanickam, V. J. Leung, K. Pedretti, S. L. Olivier, D. P. Bunde, U. V. Catalyurek, and K. Devine, "Exploiting geometric partitioning in task mapping for parallel computers," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 27–36.

[11] M. Deveci, K. Kaya, B. Uçar, and U. V. Catalyurek, "Fast and high quality topology-aware task mapping," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 197–206.

[12] A. H. Abdel-Gawad, M. Thottethodi, and A. Bhatele, "Rahtm: routing algorithm aware hierarchical task mapping," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 325–335.

[13] J. Wu, Z. Lan, X. Xiong, N. Y. Gnedin, and A. V. Kravtsov, "Hierarchical task mapping of cell-based amr cosmology simulations," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 75.

[14] J. Wu, X. Xiong, and Z. Lan, "Hierarchical task mapping for parallel applications on supercomputers," *The Journal of Supercomputing*, vol. 71, no. 5, pp. 1776–1802, 2015.

[15] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, and D. K. Panda, "Design of a scalable infiniband topology service to enable network-topology-aware placement of processes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 70.

[16] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 336–347.

[17] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 2014, pp. 129–140.

[18] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 76.

[19] "METIS: Graph Partitioning Tool," http://glaros.dtc.umn.edu/gkhome/views/metis.

[20] Ü. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1479–1487.

[21] "Intel MPI Benchmarks User Guide and Methodology Description," http://www.hpc.ut.ee/dokumendid/ips_xe_2015/imb_latest/doc/IMB_Users_Guide.pdf, accessed: 2016-6-4.

[22] L. G. Valiant, "A scheme for fast parallel communication," *SIAM journal on computing*, vol. 11, no. 2, pp. 350–361, 1982.