Hierarchical Task Mapping on Dragonfly topology for scaling Molecular Dynamics

A Thesis

Submitted for the Degree of

Master of Technology

in the **Faculty of Engineering**

by

Sabyasachi Sahoo

Computational and Data Sciences
Indian Institute of Science
Bangalore – 560 012 (INDIA)

JUNE 2016

DEDICATED TO

The Almighty

and the research fraternity

Signature of the Author :	....
	Sabyasachi Sahoo
	Dept. of Computational and Data Sciences
	Indian Institute of Science, Bangalore

Signature of the Thesis Supervisor :	....
	Dr. Sathish S. Vadhiyar
	Associate Professor
	Dept. of Computational and Data Sciences
	Indian Institute of Science, Bangalore

Acknowledgements

First of all, I would like to thank The Almighty for being with me in all times and guiding me om the right path.

I would like to express my sincere gratitude to Dr. Sathish Vadhiyar for his continuous support of my research work, without whom this work would not have been possible. Right from the literature survey phase to thesis writing, he has been instrumental in every part of the project, minutely analysing and rectifying every single step of progress in the process. I look up to many of his qualities at professional and personal front, which I wish to imbibe.

I would like to thank IISc, especially CDS (or SERC) department, for having given me this opportunity to experience one of the greatest academic environment and bring out another person in me. I have learnt a lot from my faculties, and not all of it had to do with the topics they taught. I would like to thank the system admins at SERC, Vinod and Anoop, for their continuous help for Sahasrat usage. I am thankful to all my classmates (who also happen to be the first batch '16 of CDS department to graduate) and all the members of MARS Lab for creating an environment favourable for free thinking.

I would like to thank Ashirbad Mishra for solving most of my errors and problems, big or small. My discussions with Vineetha Kondameedi have often led me to important conclusions for my project, for which I am very grateful to her. I would like to acknowledge the contributions of Surriennddrrahh Varma and Rintu Panja for giving me a sneak peek into mapping algorithms and graph partitioning algorithms. I would like to thank Kartik Kharbanda, Nikhil Ranjanikar and Manjunath Hegde for supporting me in different ways during the course of the project. I would also like to thank Amlesh Kashyap for educating me on the qualitative aspects of research and life.

Back home, I would like to thank my father for making me stronger each day and and my mother for supporting me at all levels. I would also like to thank my sister for all the love and support she gave me.

Abstract

Molecular Dynamics (MD) simulation provides a platform to conduct and analyse chemical and biological experiments at molecular levels. It 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. Communication is one the major bottleneck for LAMMPS for various molecular systems. It limits the scalability of LAMMPS to higher number of cores. With the advent of exascale systems, it becomes even more 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. It is an hierarchical network topology design constituting of various levels and at each level, all the processors are one hop away from each other. Despite having multiple advantages like non-minimal routing in Cray, communication is still a major bottleneck.

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. Our experiments have shown that for any given message size, communication at an higher level (like Inter-Group) can have a lot of communication overhead w.r.t. communications at lower levels (like Inter-Node communications). Hence, our mapping algorithm follows a top down approach, unlike any previously proposed mapping algorithm. As a result, we are able to successfully prioritise communications taking place at higher levels of communication w.r.t. communications taking place at lower levels. However, we also notice that pushing all large message communications to lower levels may not be beneficial and hence propose a dynamic mapping algorithm which can predict the correct mapping depth (depth or network level upto which mapping should be applied) and generate the mapping accordingly.

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 a maximum of 44% on 8016 cores. While we had applied our strategy to dragonfly topology, our work is also applicable to general hierarchical topologies. In future, we would like to develop a mapping algorithm that can account for probabilistic routing function, congestion function, variation of communication time at a specific level, network noise function, contention function, etc. Using these functions can also help us develop better dynamic algorithms, that can accurately predict the mapping depth that will give us optimal performance. Since the communication volume of molecular dynamics is dynamic in nature and unpredictable, we propose to perform dynamic remapping in future, to best account for this uncertainty. We can also account for non-hierarchical topologies by using neighbor joining algorithm to automatically predict the network topology. The resulting mapping algorithms can be put together as an open source generic mapping tool.

Contents

List of Figures

List of Tables

LIST OF TABLES

Chapter 1

Introduction

In this chapter, we discuss about the scalability bottlenecks of LAMMPS (a popular MD package), network design of dragonfly topology and importance of mapping for MD.

1.1 Molecular Dynamics and its Scalability

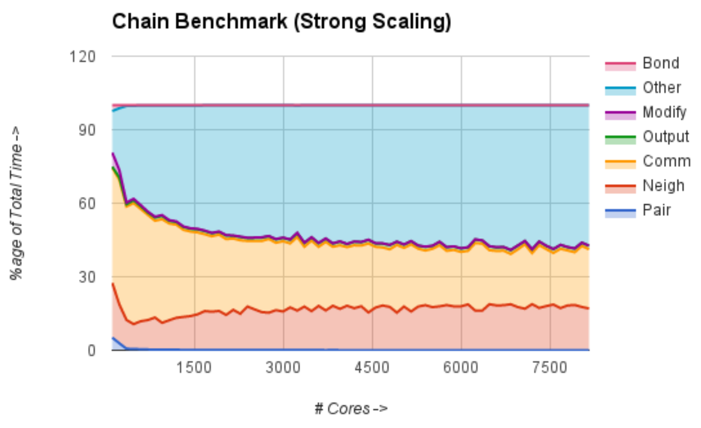
The scientific community is interested in a variety of experiments that requires analysis at molecular level. Molecular Dynamics (MD) simulation provides a platform to conduct these experiments which not only help determine various properties of the material being simulated, but also helps predict, understand, compare and interpret real life experimental observations. Ideally, MD requires solving Schrodinger's wave equation for determining the properties and end results of an experimental simulation, called as *ab-initio* MD. *ab-initio* MD helps to simulate chemical reactions quite accurately, but solving the wave equation is not easy in general. Analytical solution for the wave equation exists only for one electron atom, and other multi-electron atomic systems are solved using numerical methods, which are very computationally expensive. Hence, without loss of much accuracy, these material properties can be simulated by applying classical Newton's equations of motion to molecular systems, subjected to constraints.

MD simulations are used for prediction of various complex properties of materials like density, heat of vaporisation[27]. MD simulations are used for a better understanding of the interactions between proteins and ligands and also in prediction of stable protein structure[20]. It has become a standard tool in various industries like pharmacy, aerospace, nuclear, etc. However, high computational cost of MD has hindered its acceptance among researchers as a platform for simulating large or real life simulations. It is still mostly used to simulate nano-scopic sized molecular systems for few microseconds. 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.

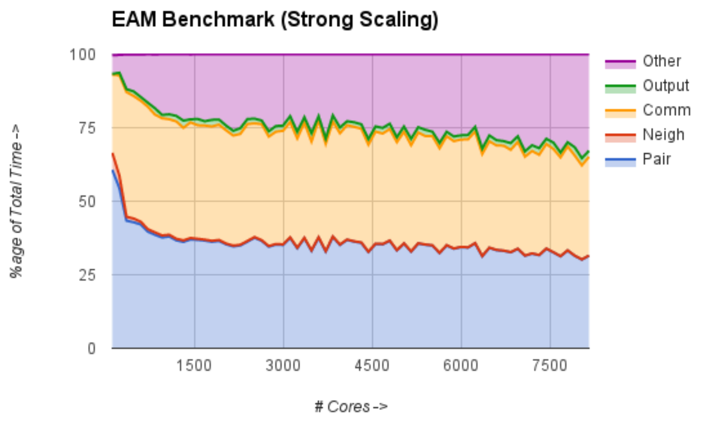
Few popular parallelization strategies[21] that have been proposed in the past are atom decomposition, force decomposition, spatial (or domain) decomposition or any strategy which combines them. It has been shown [14] that domain decomposition methods usually scale well and most packages use it as their fundamental parallelization strategy. In domain decomposition, we partition total simulation volume into numerous 3D blocks or cells among the processors. The force computations can be reduced by using Newton's third law of motion, using which pairwise force computation is necessary only once. The major phases of any MD simulation which forms the mains structure of single time step are as follows:

- **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
- **Other:** is the remaining time

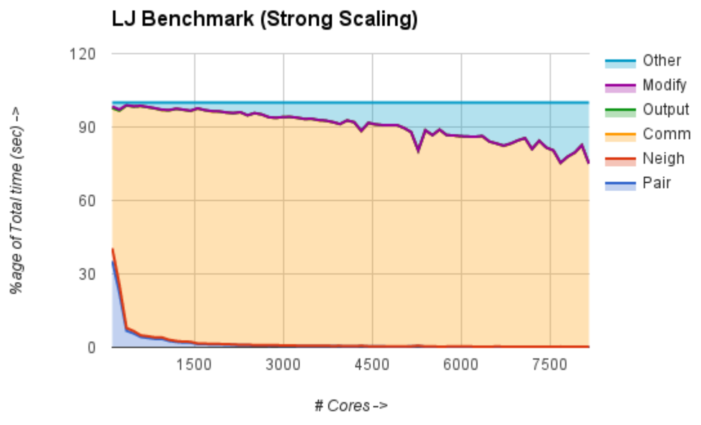
Communication is usually one of the least scaling phases in MD which also has been confirmed by our experiments on LAMMPS. We conducted scalability tests of LAMMPS for Chain, EAM and LJ benchmarks whose results are shown in figure 1.1. These area plots show percentage execution time of different phases of LAMMPS with varying number of cores. For EAM and LJ benchmark, communication time clearly accounts for most of the total simulation time and is the main bottleneck phase limiting the scalability of LAMMPS.



(a) Chain benchmark



(b) EAM benchmark

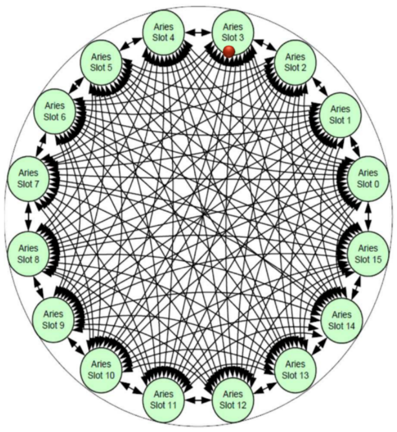


(c) LJ benchmark

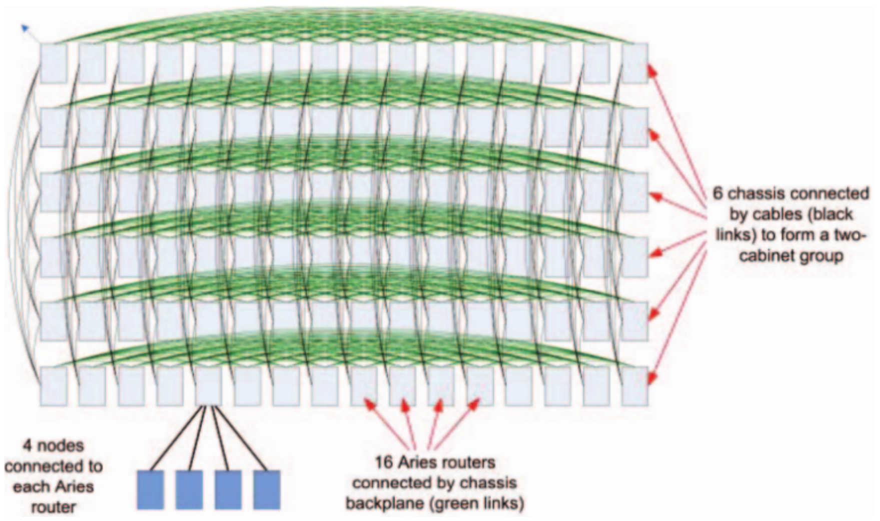
Figure 1.1: Percentage execution times for varying cores for LAMMPS

1.2 Dragonfly network topology

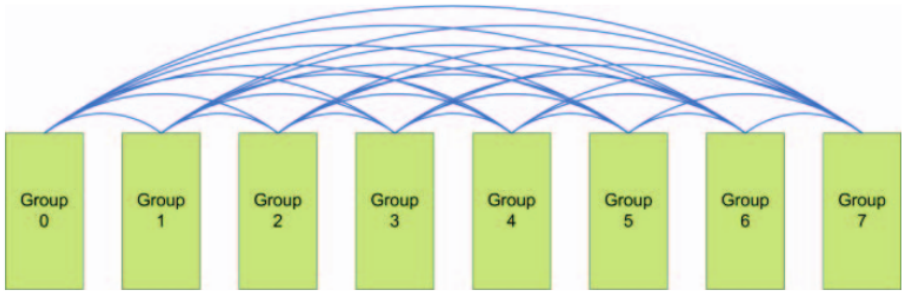
Network topology specifies the node layout in a supercomputer and plays a major role in communication time of an application. Dragonfly [12] is one such hierarchical network topology design which provides scalable global bandwidth while minimizing the number of expensive optical links. In dragonfly network, 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 1.2b. 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 1.2a. 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 1.2c.



(a) Chassis Blade


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)

(b) Chassis Backplane



(c) DragonFly Groups

Figure 1.2: DragonFly Network [12]

Therefore, the different levels in a dragonfly topology in a top down manner are as follows:

- Group : Each group contains of six chassis or 9216 processors.
- Chassis : Each chassis contains fifteen blades or 1536 processors.
- Blade : Each blade contains four nodes or 96 processors.
- Node : Each node contains of two sockets or 24 processors.
- Socket : Each socket contains of twelve processors.

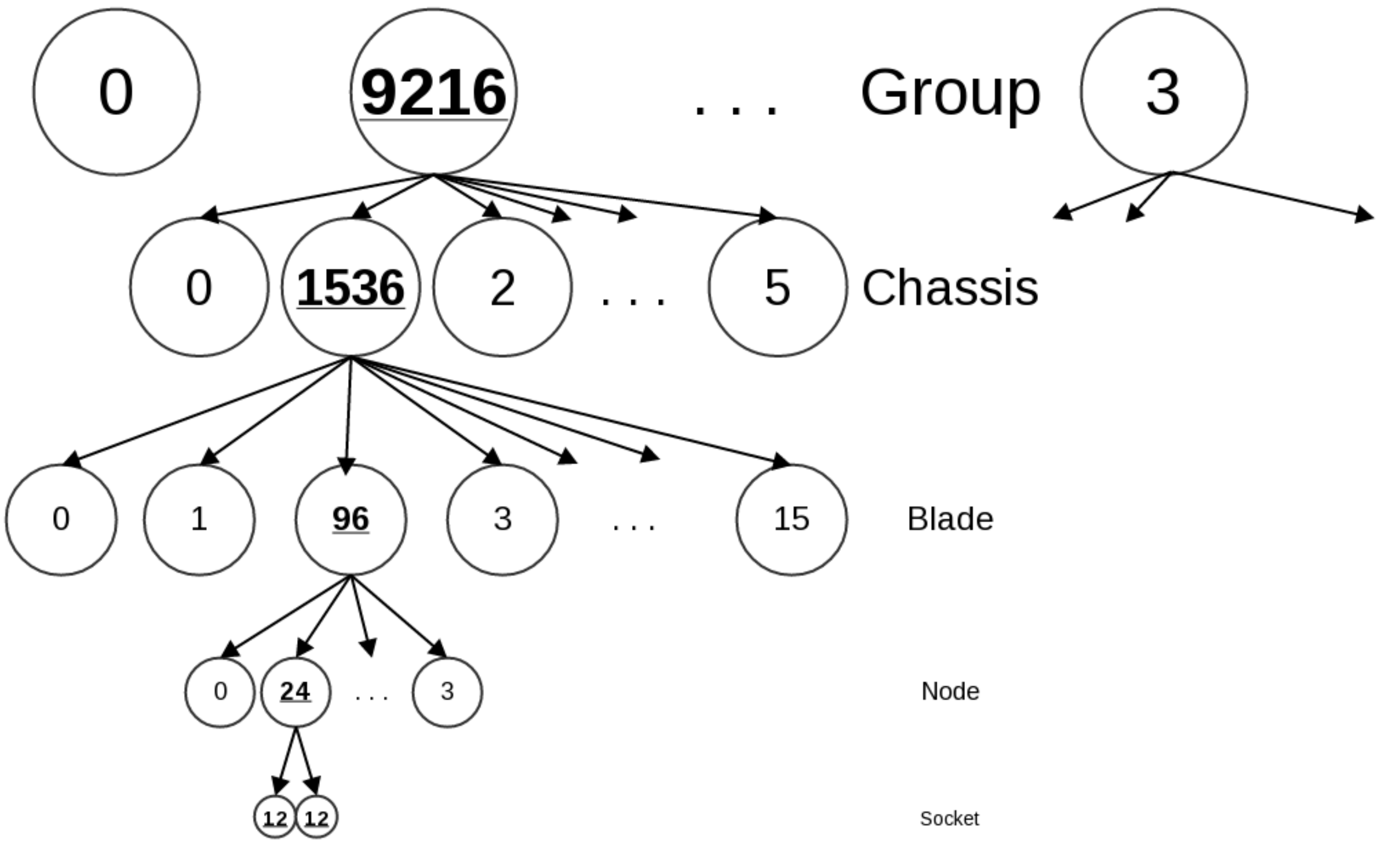


Figure 1.3: Topology Tree corresponding to Dragonfly network topology

Figure 1.3 shows the topology tree of the complete dragonfly network topology. At each level of topology, the entities are connected in an all-to-all fashion. An entity can be collection of cores, nodes, blades, etc. Network diameter of one is maintained at each level, but same is not true across the levels. Hence, this also brings out the heterogeneous nature of the topology.

1.3 Motivation

With the advent of exascale era, as the number of compute nodes increase, so will 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[16] problem and a number of heuristics have been proposed.

Mapping for MD on Cray's dragonfly topology is a non-trivial problem. Despite a structured communication pattern of MD, the communication among the processes is completely dynamic and varies with time and space. The communication graph (with processes as vertices in graph and communication volume among them as the edge weights) formed for the MD is completely unpredictable for different iterations in a single simulation and across different simulations. The mapping algorithm will need to account for some degree of similarity of communication graph across the simulations for forming the communication graph. Also, Cray follows non-minimal routing algorithm for routing of messages, which seeks to minimize communication overload for most of the applications. This implies that no application running in Cray is free from external network noise and as a result, even near-optimal mapping algorithms will not be able to achieve great performance improvements. Developing a mapping algorithm that takes care of all of this is definitely a big challenge.

1.4 Contributions

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 molecular dynamics application on dragonfly topology. We also account for non-contiguous job allocation policies of Cray, and have shown improvement in all possible allocations. We propose partitioning based task mapping which exhibits fine-grained flexibility for mapping processes to processors in a top-down manner, prioritizing higher levels of network. We also propose an algorithm to dynamically determine the depth of mapping (level in network hierarchy after which processes are randomly placed) for which suspected contention should be minimum. We show that our method is the most efficient in reducing the communication volume at higher levels of the network, but also carefully increasing it at lower levels. We obtain an improvement of more than 15% in general over the default mapping of LAMMPS.

1.5 Outline of the thesis

The rest of the thesis is divided as follows: chapter 2 describes the recent work on mapping algorithms. It contains literature survey of important mapping algorithms that were proposed in the past and many recent works that are relevant to mapping on dragonfly network. Chapter 3 gives the mathematical and experimental reasons for the choice of our mapping algorithm and is explained in great detail. We propose partitioning-based hierarchical task mapping algorithm which tries to make best use of hierarchical network topology for intelligent placement of processes to processors. We show how communication performance can be improved by restricting the mapping to certain mapping depth. Chapter 4 contains all the results obtained during the course of this project. It shows comparisons of our work with two other mapping algorithms, that we believe are the most suitable for MD on dragonfly topology. Finally, in chapter 5 we conclude with the discussions on future work that can be performed on similar lines.

Chapter 2

Review of Related Literature

In this chapter, we briefly discuss about few popular MD packages, some of the initial important mapping algorithms and algorithms that are relevant to the mapping of LAMMPS on dragonfly network.

2.1 Few popular MD packages

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 [21], GROMACS [6], RedMD [13], Desmond [8], NWChem [26], NAMD [19], AMBER [23] are leading packages used extensively by molecular dynamics community.

2.2 Various mapping algorithms

The most common mapping strategy in most MPI implementations, and also followed as default mapping in LAMMPS, was implemented in CHACO[15]. The processes are mapped to processors in the order of their MPI rank and processor (or node) ids. Baba et al. [5] implemented topology independent allocation system by evaluating several functions that represented properties of the graph. Agarwal et al. [4] demonstrated a process mapping strategy that minimizes total number of hop bytes communicated for torus and grid networks.

Hoefler et al. [17] 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. It has been shown in the figure 2.1 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.

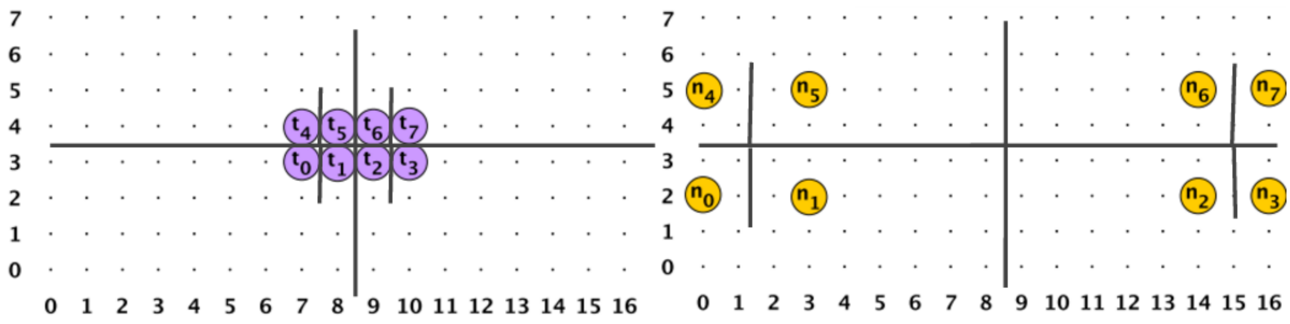


Figure 2.1: Deveci et al. [10] 's mapping algorithm

Abdel et al. [3] 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. [28] 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 [29] incorporate neighbor joining algorithm [24] to detect fat tree topology. Jain et al. [18] analyze the behavior of dragonfly network for various routing strategies, job placement policies, and application communication patterns using network simulations. Prisacari et al. [22] 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. [7] 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 algorithm has been shown in figure 2.2. 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.

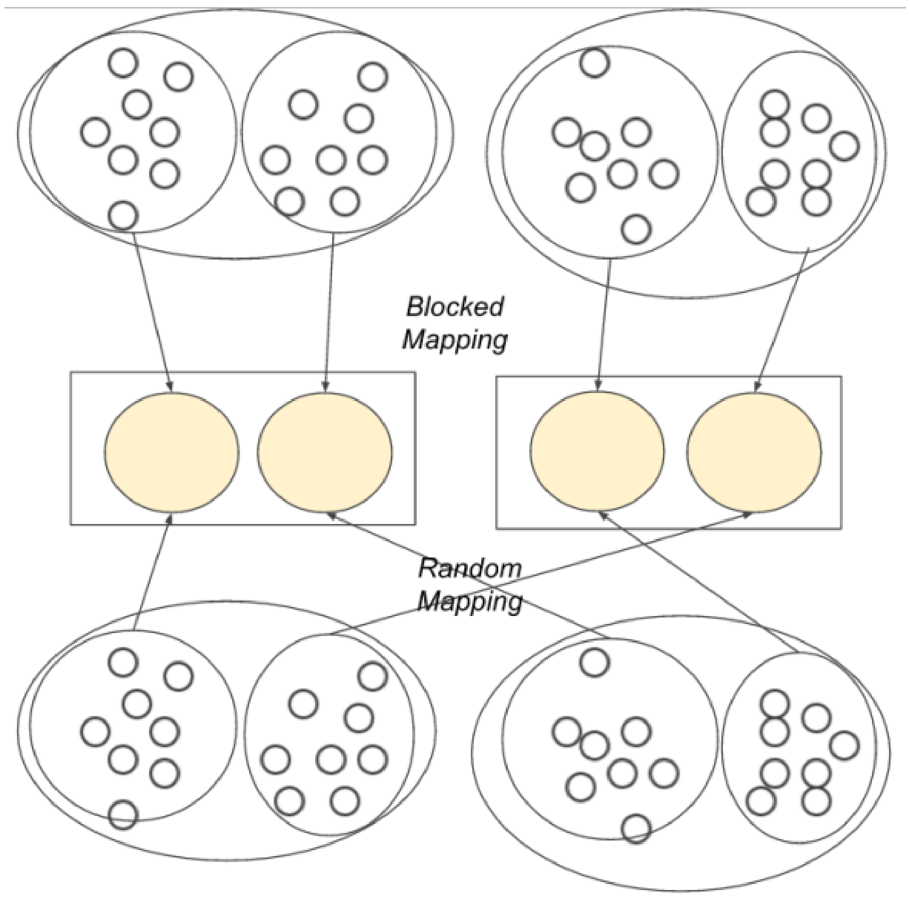
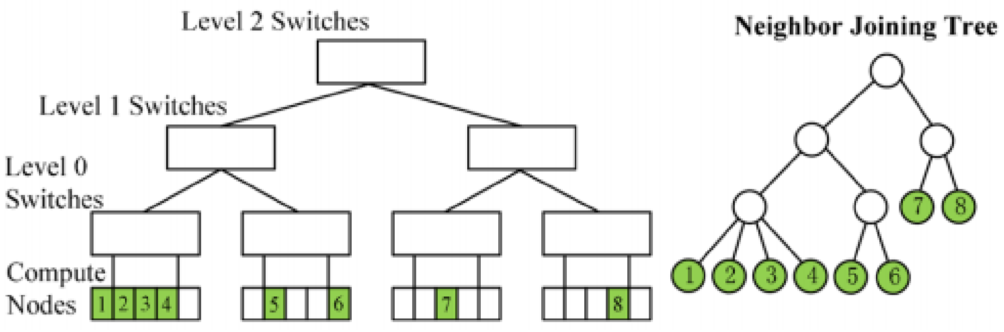


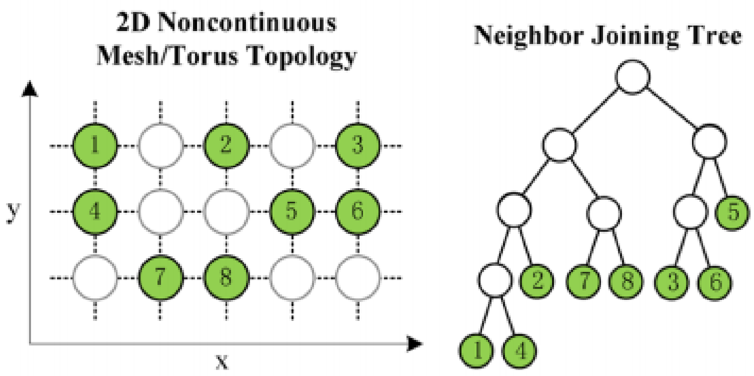
Figure 2.2: Bhatele et al. [7] 's mapping algorithm

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. [29] 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 [2]. Their mapping algorithm has been shown in figure 2.3. 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 [24] 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 without trying to factor the mapping depth(level upto which mapping applied produces near-optimal results).



(a) For fat tree topology



(b) For mesh/torus based topology

Figure 2.3: Wu et al. [29] 's mapping algorithm

Table 2.1 compares various mapping algorithms proposed by various works and how far they account for each of the factors important for task mapping on dragonfly networks. We see that some earlier works [10, 11] have accounted for sparse node allocations, but do not account for heterogeneity

Mapping Work	Heterogeneity	Hierarchical	Non Minimal Routing	Sparse allocation	Fine-grained flexibility	Mapping Depth	Topology applied
Deveci et al.'14	-	-	-	✓	✓	-	Torus
Deveci et al.'15	partly ✓	-	-	✓	✓	-	Torus
Wu et al.'12	-	partly ✓	-	-	-	-	Fat tree & Mesh/Torus
Wu et al.'15	-	✓	-	✓	-	-	Fat tree & Mesh/Torus
Bhatele et al.'11	✓	✓	partly ✓	-	-	-	IBM PERC (simulated)
Jain et al.'14	✓	✓	partly ✓	-	-	-	Dragonfly (simulated)
Our work	✓	✓	✓	✓	✓	✓	Dragonfly

Table 2.1: Factors accounted by various works

of the network. Recent works [7, 18] account for heterogeneity and hierarchical nature of topology, but do not factor the sparse node allocation. Also, factors like non-minimal routing, fine-grained flexibility of mapping and mapping depth have either been considered partially or are oblivious to it. In this work, we try to account for all these factors.

Chapter 3

Methodology

In this chapter, we discuss about important factors to account for while developing mapping algorithm, our first weighted partitioning based mapping algorithm and dynamic task mapping algorithm.

3.1 Theoretical Analysis

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.

Level	Bandwidth(Gbps)
Inter-Group (IG)	12.5
Inter-Chassis (IC)	14
Inter-Blade (IB)	14
Inter-Node (IN)	16

Table 3.1: Ideal Bandwidth values of Cray

Table 3.1 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, 7] are also not applicable for these topologies. Also, 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 regard to the allocation's shape or locality. This means that MPI jobs requesting only 240 cores can end up performing all types of communication. Our mapping algorithm also needs to account for these factors.

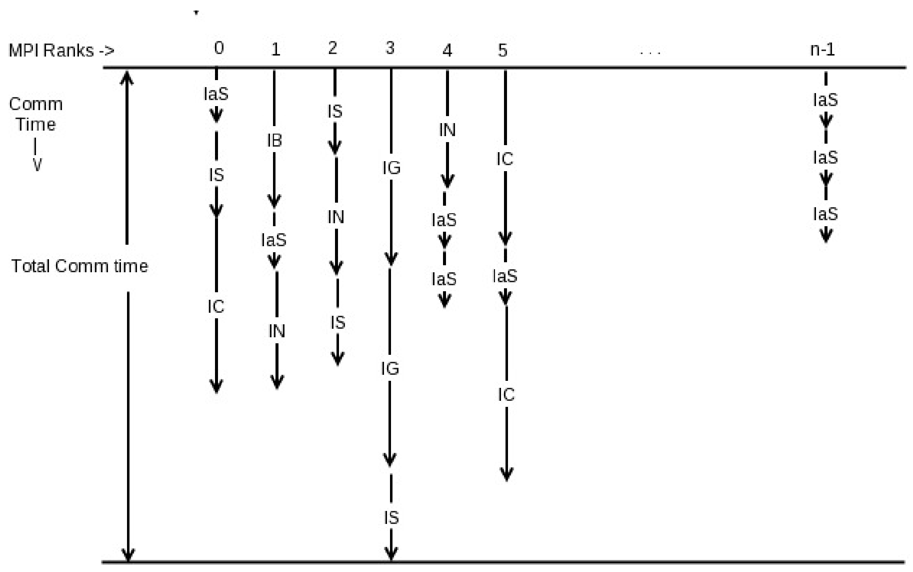


Figure 3.1: Parallel communications in MD

Mapping problem can be formalized as a minimization problem as follows:

$$Minimize\ (t_{total})$$

where t_{total} is the total communication time. We will henceforth limit our discussion on communication time for molecular dynamics only. Domain decomposition is the most popularly chosen strategy in molecular dynamics, where each process needs to communicate its data with twenty-six other processes in a 3D cube (or eight processes in 2D square). By intelligently carrying out communications in a sequential fashion, it is brought down to at-most six processes. As a result, communication pattern is quite sparse in MD and it stands a greater chance to benefit from intelligent mapping algorithms. Figure 3.1 (considering every process performs only three communications) shows pictorially what affects the total communication time. In MD, each of the six processes, communicating in parallel with each other, can be communicating through any one or all of the types of links of the network. So total communication time for MD is governed by slowest communicating process. Hence, mapping problem can be rephrased as :

$$Minimize\ (\max_i\ t_i)$$

where i is rank of a process. Communication time through a communication link at level j of network hierarchy characterised by latency constant l_j and bandwidth constant b_j for a message of size m can be theoretically expressed as

$$t_{i,j} = l_j + (m/b_j)$$

where $t_{i,j}$ is communication time for process i, through a communicating link at level j. So, mapping problem becomes

$$Minimize\ (\max_i \sum_j l_{i,j} + (m/b_{i,j}))$$

where $l_{i,j}$ is latency constant seen by process i, while communicating through link at level j.

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

Table 3.2: Average Experimentally observed Latency and Bandwidth constants for medium messages of size 1Kb - 8KB

Table 3.2 shows average communication latency and bandwidth constant. It was generated using Ping-Pong test among all processes which is later explained in section 3.3. At each level of hierarchy of Dragonfly topology, all entities are all-to-all connected. So, we assume the latency and bandwidth constant as seen by communicating processes at any particular level to be the same. Hence mapping problem for dragonfly topology can be safely be modified to

$$Minimize\ (\max_i \sum_j l_j + (m/b_j))$$

where, l_j and b_j are average latency and bandwidth constants for a link at level j.

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 us test this idea.

Let our intuition based mapping be called as $mapHier$ and any other default mapping be called as $mapDef$. Let us assume a process X communicating to two other processes, A and B in the application communication graph. The size of message communicated from $X \to A$ is m_a and $X \to B$ is m_b, such that $m_a < m_b$. For simplicity let us assume, that process X has been mapped to some processor χ, and we have two other processors α and β for placement of processes, A and B. Let the communication $\chi \to \alpha$ be a level 0 communication characterised by latency and bandwidth constants, l_0 and b_0 respectively and $\chi \to \beta$ be a level 1 communication characterised by latency and bandwidth constants, l_1 and b_1 respectively, such that level 0 is at higher level (eg: Inter-Group) of hierarchy w.r.t. level 1 (eg: Inter-Chassis). Hence, from table 3.1 and table 3.2 we can assume that $l_0 > l_1$ and $b_0 < b_1$. The process A is mapped to processor α and process B is mapped to processor β in $mapHier$ mapping and vice-versa in $mapDef$ mapping. If t_{mh} and t_{md} be communication time for process X in $mapHier$ and $mapDef$, respectively, then

$$\begin{aligned}
t_{mh} &= t_{mh}^{XA} + t_{mh}^{XB} \\
t_{mh} &= l_0 + \frac{m_a}{b_0} + l_1 + \frac{m_b}{b_1} \\
& and \\
t_{md} &= t_{md}^{XA} + t_{md}^{XB} \\
t_{md} &= l_0 + \frac{m_b}{b_0} + l_1 + \frac{m_a}{b_1}
\end{aligned}$$

Communication time for both the mapping can be compared as follows:

$$t_{mh} - t_{md} = (\frac{1}{b_0} - \frac{1}{b_1}).(m_a - m_b)$$

Now, since $b_0 < b_1$ and $m_a < m_b$. Therefore,

$$\begin{aligned}
t_{mh} - t_{md} &< 0 \\
t_{mh} &< t_{md}
\end{aligned}$$

which implies that communication time for any process X due to $mapHier$ will be lower than time from any other mapping like $mapDef$. Since it is true for any process X, hence it will also be true for the slowest process in the application. Hence, we have proven by contradiction that total communication time of mapping generated $mapHier$ will be lower than that generated by $mapDef$.

So, we need to devise an algorithm that not only tries to ensure that smaller message sizes are passed through higher levels and vice-versa, but also account for every level of the network. 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 passed through any one process to other) at each level, recursively. Partitioning at any level gives us the minimum edgeweight-cut which ensures that smaller message sizes are passed 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.

3.2 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 3.4) 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 3.2 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.

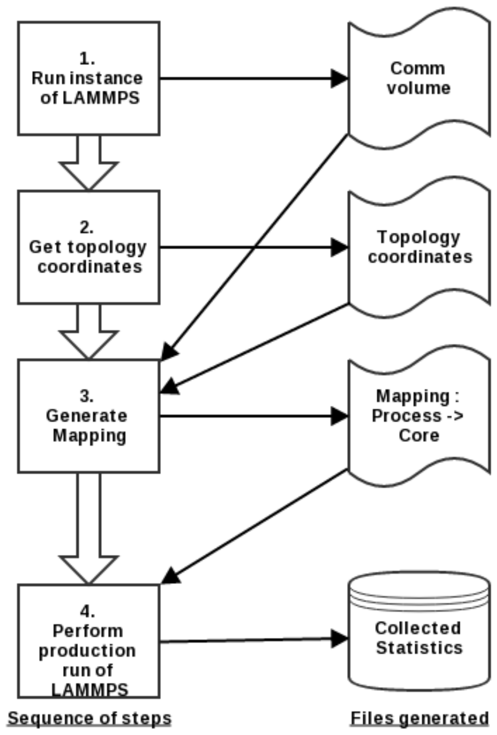


Figure 3.2: Job script for generating mapping

3. Finally, we recursively partition the graph based on the topology to obtain the mapping of processes to cores.

The third phase is the most important phase, where we perform recursive partitioning of the communication graph using Patoh [9] graph partitioning tool. Patoh can generate weighted partitions for any given hyper-graph. Most popular mapping algorithms [28, 29] 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 $RecursiveMapping$ function. 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

Algorithm 1 Hierarchical Task Mapping Algorithm

Input: Communication volume file ($cvFile$), Topology coordinates file ($tFile$), mapping Depth (d)

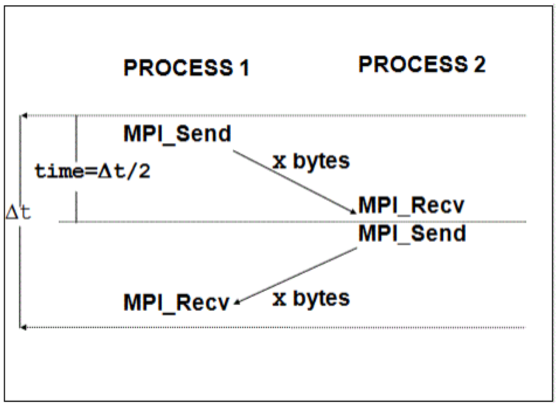
Output: Mapping file for LAMMPS ($mapFile$)

```
 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
```

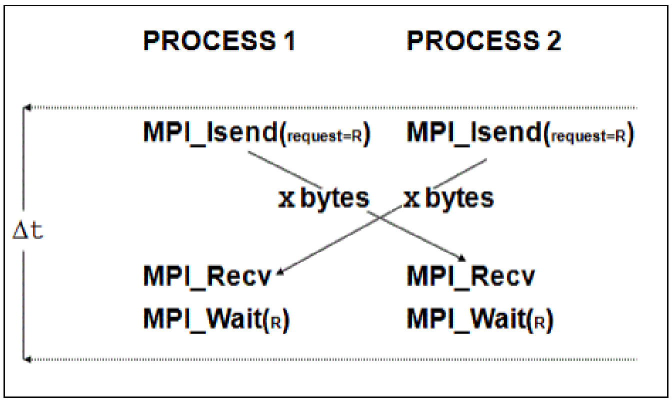
partitions to underloaded partitions. Next, we form communication graph from each of the partitions of the communication graph and recurse on the newly formed graph for the entity at next lower level. The recursion stops if we reach the lowest level of the network topology or the level till which the partitioning is desired.

3.3 Dynamic Task Mapping Algorithm

Figure 3.4 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 3.3. These plots were generated from the code developed in-house (similar to [1]), performing these communication patterns. It brings out the heterogeneous nature of topology. 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 3.2). Therefore, mapping algorithms trying to minimize hop-count, hop-bytes, etc., fail for such networks.



(a) Ping Pong

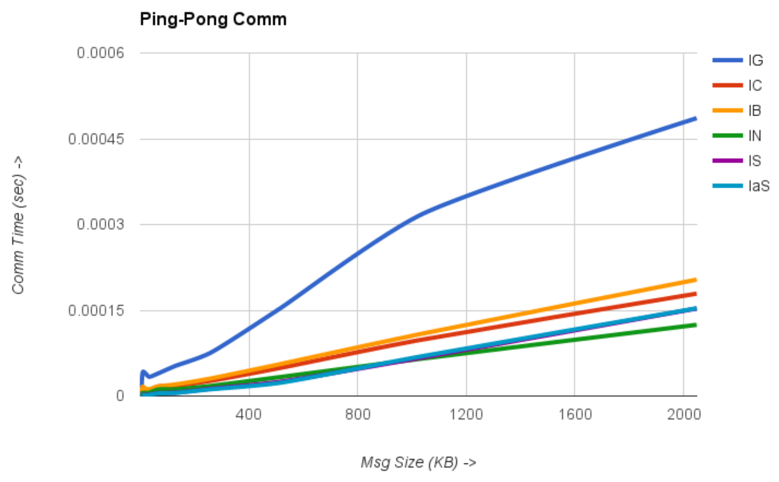


(b) Ping Ping

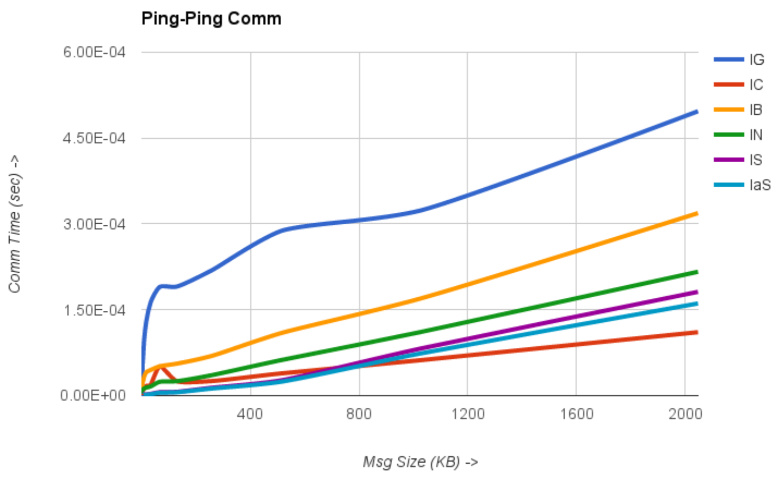
Figure 3.3: Communication Pattern [1]

As we can see from both these plots, 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. This is contrary to the previous assumption that communication time reduces as we go down the levels of topology, based on ideal bandwidth values from table 3.1. Previous works have shown [28] that this can be due to smaller memory bandwidth per core or inefficient implementation of intra-node communication algorithms in mvapich library. Therefore, for very higher message sizes, it might be preferable to not push them to lower levels of communication.

In a Ping-Pong communication, the same communication links are used one after the another whereas in a Ping-Ping communication, the probability of using the same communication links in parallel is higher. As a result, the probability of congestion increases due to possible congestion. Hence, we see the threshold calue is lower for Ping-Ping communication w.r.t. Ping-Pong communication. In LAMMPS communication pattern, we will see both the Ping-Pong and Ping-Ping type of communications taking place in the network. Due to suspected contentions and lower threshold value, it necessitates the need to develop a mapping that accounts for it accordingly.



(a) Ping Pong



(b) Ping Ping

Figure 3.4: 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 account for reduced threshold by restricting partitioning to a certain level from top, also called as mapping depth. We have generated results to support our claim in section 4.2. Figure 3.5 shows that mapping depth of 2 (applying mapping till blade level and then map the processes according to their MPI rank below it) gives maximum percentage improvement on maximum communication time. It shows that taking care of suspected 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.

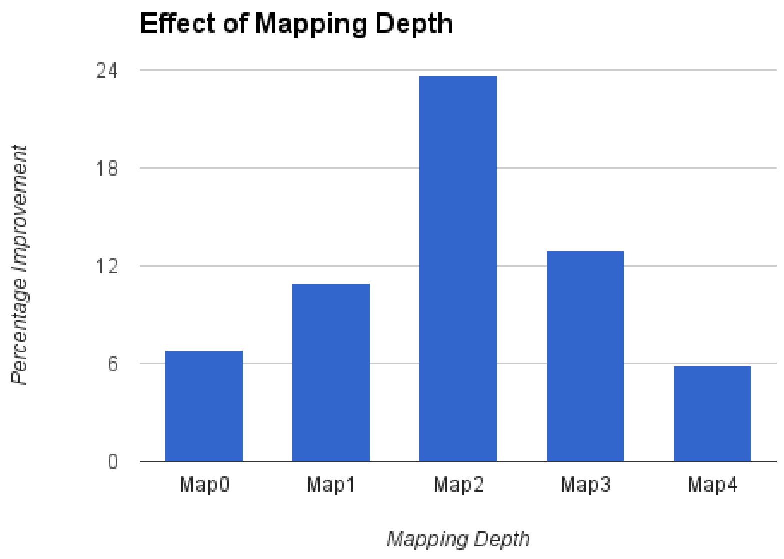


Figure 3.5: Percentage improvement on communication time for varying mapping depth for 240 cores

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).

To summarise, our mapping algorithm successfully accounts for the following factors:

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
```

- Heterogeneity : We treat communication at each level differently by partitioning at different levels. As a result, processes are mapped based on the type of communications performed by it.
- Hierarchical : We recursively partition the communication graph in a top-down approach to prioritise communications taking place at higher levels. Our mapping algorithm is the only algorithm known to us that follows a truly prioritises higher level communications.
- Non-Minimal routing : Metrics like maximum congestion fail for such systems because we can not pin-point the links that will be most congested. Instead, we focus on reducing congestion or communication volume at higher levels of network only.
- Sparse allocation : We perform weighted partitioning of communication graph to account for processors spread out across different groups or chassis.
- Fine-grained flexibility : We partition the communication process graph, rather than node graph. As a result, the mapping generated is flexible enough to place processes based on the topology tree above it. Applying any optimizations on node graph kills this fine-grained flexibility.
- Mapping Depth : We try to dynamically restrict mapping to a certain depth, to account for suspected contention and threshold effects.

Chapter 4

Experiments and Results

In this chapter, we show results obtained from our algorithms and two other mapping algorithms for reduction in average communication volume at higher levels and corresponding reduction in communication time of LAMMPS.

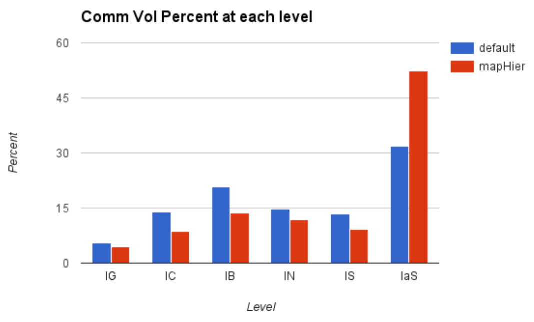
4.1 Experiment Setup

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

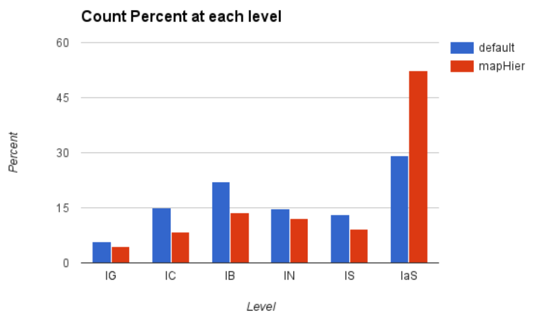
- Intel Haswell 2.5 GHz based CPU cluster with 1376 nodes.
- Each node has 2 CPU sockets with 12 cores each (i.e. 24 cores per node), 128GB RAM.
- Connected with Aries high speed interconnect on a dragonfly topology.
- High speed DDN storage unit supporting Crays parallel Lustre filesystem.

4.2 Bringing down threshold and contention effects

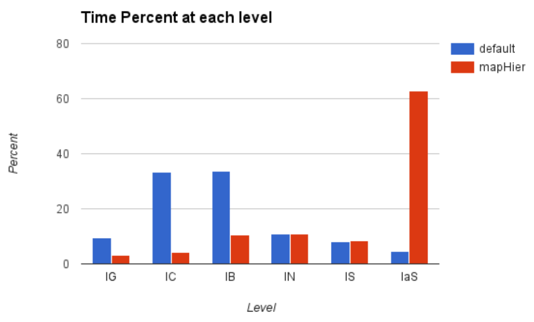
Figure 4.1 shows few statistics for 3072 core run at each level. These plots have been generated by considering for all the communicating processes. These plots show how these metrics vary w.r.t. default mapping due to our mapping algorithm discussed above. Our mapping algorithm tries to minimize inter-entity communication at each level of network topology, except for the lowest level (i.e. for intra-socket communications). Same is also confirmed from figure 4.1a where we see that percentage of communication volume at each level for our mapping method is significantly lower than the default mapping, except for intra-socket level. This corresponds to reduction in percentage of communications at all higher levels (as seen in figure 4.1b) and reduction in percentage of time spent at each level (as seen in figure 4.1c). Therefore, our mapping algorithm balances out average time taken per communication for all levels, w.r.t. default mapping, as shown in figure 4.1d.



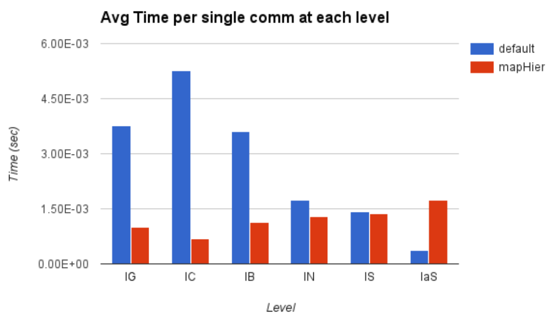
(a) Percentage of communication volume



(b) Percentage of count of communications taking place



(c) Percentage of time spent by processes



(d) Average time per single communication

Figure 4.1: Few statistics collected for default mapping ($default$) and our mapping method $mapHier$ at each level of network topology for 3072-core run

But we can see that average time taken now for intra-socket communication is more than that of any other level. This shows that in our mapping, not only more processes are doing intra-socket communication as shown in figure 4.1b, but each communication is taking a considerable time. Therefore, we proposed to restrict mapping upto certain level in section 3.3. Figure 4.2 shows average time taken per communication for different mapping depths. We see that mapping depth 3 best balances the communication, and this mapping depth gives the best improvement over all other mapping depths.

4.3 MD Setup

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-

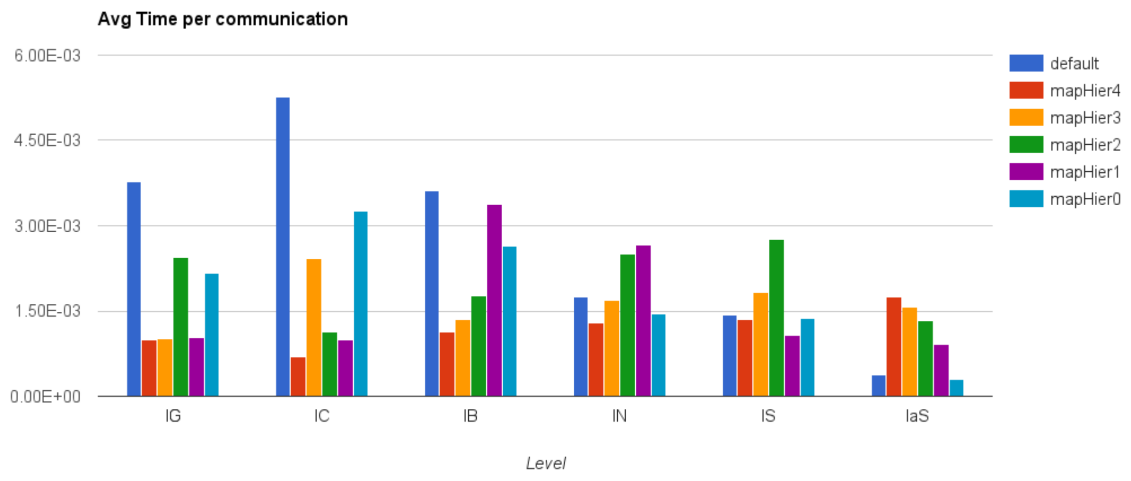


Figure 4.2: Average time per single communication for different mapping depth for different levels

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 [29] (represented by HTM) and $NUMA$ mapping of LAMMPS. $NUMA$ mapping ensures that contiguous sub-sections 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.

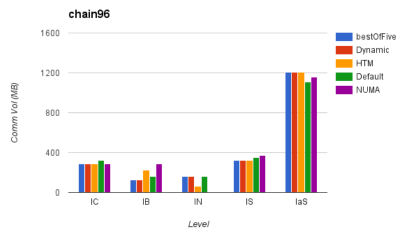
4.4 Mapping Experiments

At the beginning of this project, we developed a partitioning-based mapping algorithm (algorithm 1) which seeks 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 3.3. 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

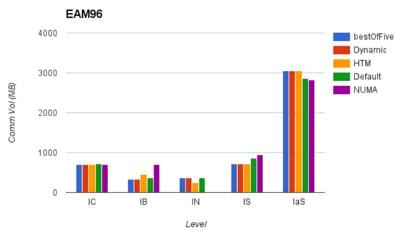
		Chain	Chain	Chain	EAM	EAM	EAM	LJ	LJ	LJ
Level	Cores ->	96	384	8016	96	384	8016	96	384	8016
Tot	bestOfFive	2102.8463	2129.7077	2131.115	5174.9174	5177.9488	5225.7751	3540.2043	3540.8476	3540.6832
	Dynamic	2100.92	2129.882	2131.1269	5174.9174	5177.8968	5225.7753	3540.1413	3540.8398	3540.6832
	HTM	2102.6385	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
	NUMA	2100.7351	2190.1876	14491.524	5174.9146	5209.4342	31502.0382	3540.2043	3632.841	22835.852
IG	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
	NUMA	0	373.017	0	0	849.1132	0	0	583.3869	0
IC	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
	NUMA	287.1711	167.8908	501.7718	698.8053	416.9118	1072.2341	478.5741	286.0977	774.2292
IB	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
	NUMA	287.1486	128.445	6563.5257	698.7528	301.2773	13987.2401	478.5508	206.908	10128.6241
IN	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
	NUMA	0	140.76	6908.6167	0	356.044	14725.2758	0	244.2736	10669.7229
IS	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
	NUMA	370.3304	194.2436	22.52	954.6478	503.9686	79.6945	651.1245	345.2619	55.2769
IaS	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
	NUMA	1156.0849	1185.831	495.0895	2822.7084	2782.1191	1637.5935	1931.9549	1966.9127	1207.9987

Table 4.1: Avg. Comm Volume (MB) for different mapping at different levels for varying cores

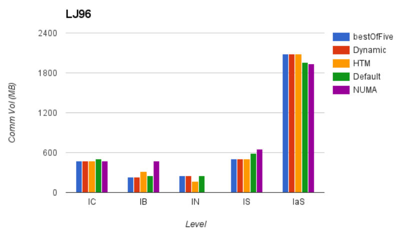
volume for the optimal mapping method at higher levels of network to be much lower than default mapping, but higher for lower levels upto certain extent (to account for both the factors discussed above). Same is also conveyed from table 4.1 and figure 4.3.



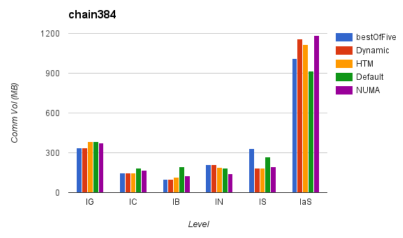
(a) Chain for 96 cores



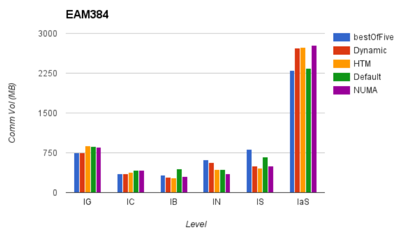
(b) EAM for 96 cores



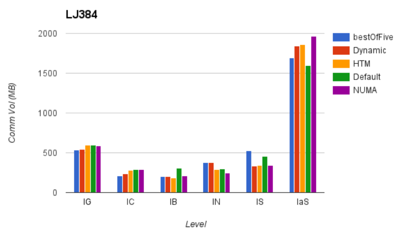
(c) LJ for 96 cores



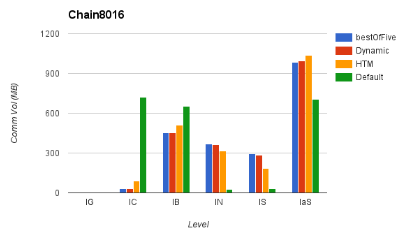
(d) Chain for 384 cores



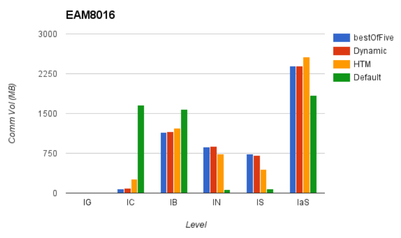
(e) EAM for 384 cores



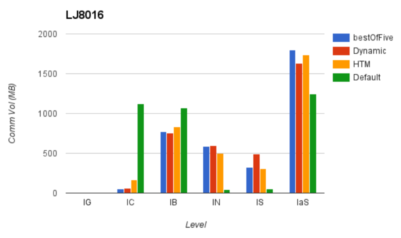
(f) LJ for 384 cores



(g) Chain for 8016 cores



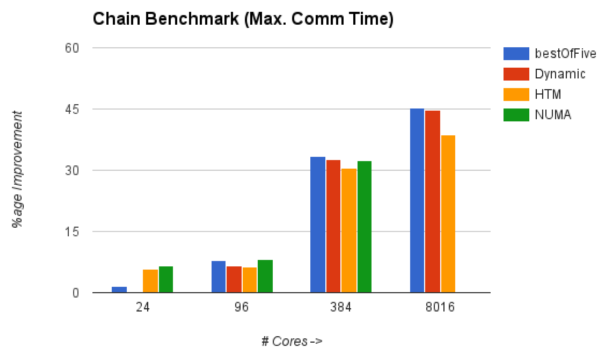
(h) EAM for 8016 cores



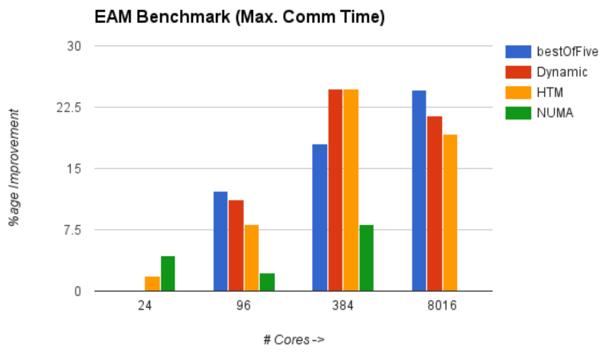
(i) LJ for 8016 cores

Figure 4.3: Avg. Comm Volume (MB) for different mapping at different levels for varying cores

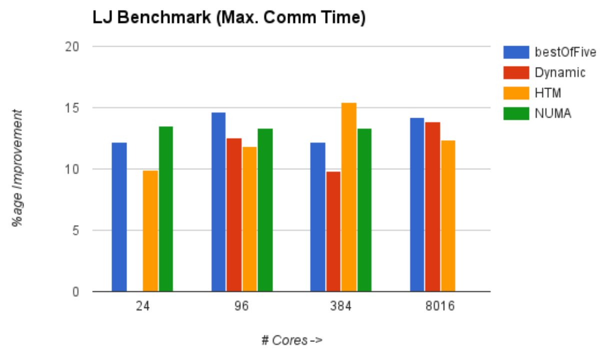
Table 4.1 shows a table of average communication volume (in MB) for different mappings, at each level of dragonfly network hierarchy and overall (represented by Tot in table), for all the three benchmarks for varying cores. Figure 4.3 shows table 4.1 graphically. Some values in IG are zero because no 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. In figure 4.3g, our mapping methods (shown by blue and red bars) brings a sharp reduction in average communication volume of about 95% over default mapping of LAMMPS (shown by green bar) at inter-chassis level. Similarly, at inter-blade level our methods bring a considerable amount of reduction. But for inter-node, inter-socket or intra-socket levels, our methods compensate for the reduction in higher levels by increasing average communication volume at these levels. However, for intra-socket levels we can see that our methods increase average communication volume above default mapping, but not more than methods like HTM, which are oblivious to threshold and contention effects. Hence, theoretically, we generate better task-mapping than HTM and $NUMA$. Similar results were obtained from the experimental runs.



(a) chain



(b) eam



(c) lj

Figure 4.4: Percentage Improvement on Max. Communication Time for varying nodes for LAMMPS

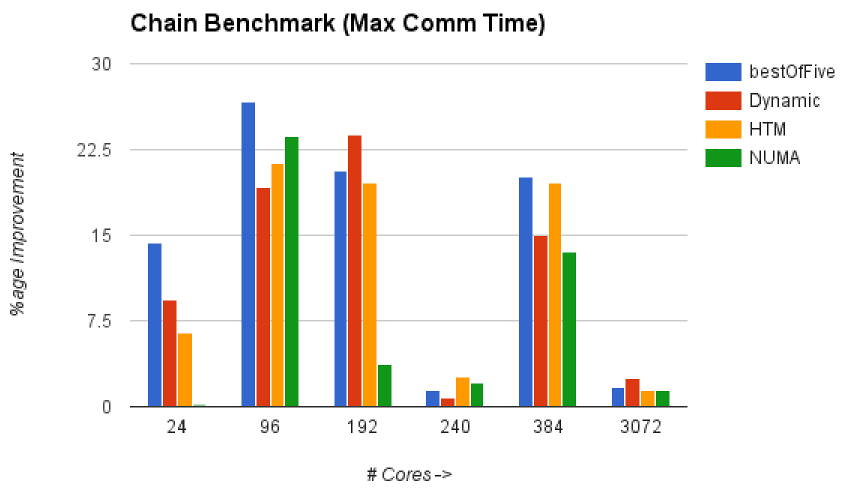
Figure 4.4 shows percentage improvement in weak scaling for maximum communication time of LAMMPS simulation. Compared to default mapping, our algorithm with the best mapping depth, $bestOfFive$ performs better than HTM and $NUMA$ mapping of LAMMPS in most core counts, getting a maximum performance improvement of 45% for maximum 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.

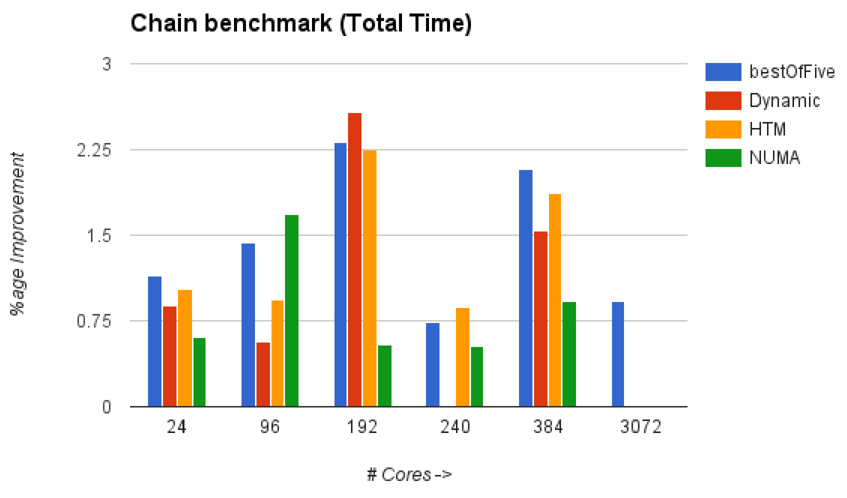
4.5 Realistic Experiments

For all the results shown above, we first run an instance of simulation of exactly the same number of iterations as that for the actual production run, to collect the communication volume amongst the processes. This is done because the communication pattern of molecular dynamics is dynamic in nature (i.e. communication varies w.r.t. time and space), unlike communication pattern of most other processes. Hence, it is quite difficult to predict the communication pattern of molecular dynamics statically, without actually running the simulation of the given molecular system. An important point to note is that most of the simulations of molecular dynamics require a pre-processing step. Therefore, we can use the pre-processing step to generate the communication volume amongst the processes. But, most of the pre-processing steps are run for fewer iteration count as compared to actual production run.

Using a smaller instance (i.e. number of iterations in the instance run is less than the number of iterations in the production run) can be detrimental to the mapping quality due to dynamic nature of communication pattern. For example, two processes that seem to be heavily communicating from the smaller instance communication graph may be actually be less communicating as the simulation advances or reaches near the equilibrium state. Still, we will like to see how our mapping methods perform under such realistic conditions. For the next set of experiments, we have considered the size of smaller instance at 1% of the actual production run. It is a practical assumption for most of the molecular systems that require a pre-processing step.



(a) Max Communication Time



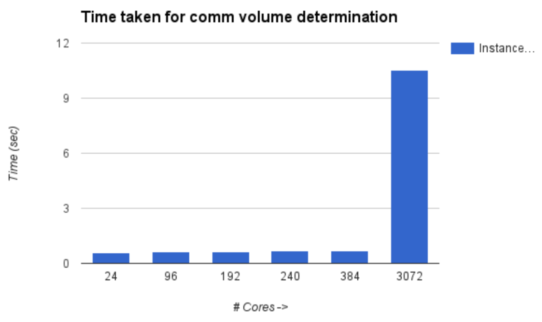
(b) Total Simulation Time

Figure 4.5: Percentage Improvement for Chain benchmark for varying nodes

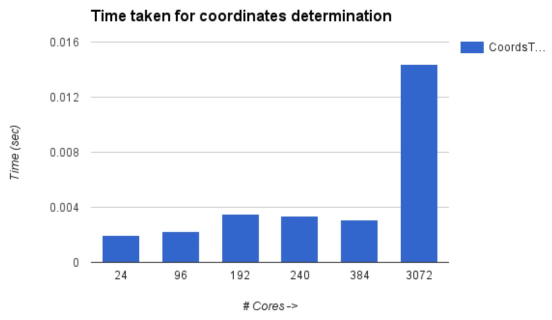
Figure 4.5 shows reduction in maximum communication time and total simulation time, considering smaller instance run for collecting communication volume. The weak scaling experiment was performed for chain benchmark. We see that percentage improvement of communication time reduces with increasing number of cores. This is because communication pattern varies w.r.t. Space. So more the number of processors, more will be the number of communications taking place (i.e. more surface area of communication face) and considering lower number of iterations for generating communication graph would result in a poor approximation of communication graph. Hence we see decreasing plot of percentage improvement for varying number of cores. This is also true for plots showing percentage improvement of total simulation time.

We see a maximum improvement of about 26.6% and 23.8% for $bestOfFive$ and $Dynamic$ respectively, compared to 21.3% and 23.7% for HTM and $NUMA$ respectively. Similarly, for total simulation time, we see an improvement of 2.31% and 2.57% for $bestOfFive$ and $Dynamic$ respectively, compared to 2.25% and 1.68% for HTM and $NUMA$ respectively. These percentages have been calculated w.r.t. $Default$ mapping. These values obtained are for lower core counts (like 192 cores). For 3072 cores, we see very minimal improvement for maximum communication time of about 2.48% for $Dynamic$ mapping.

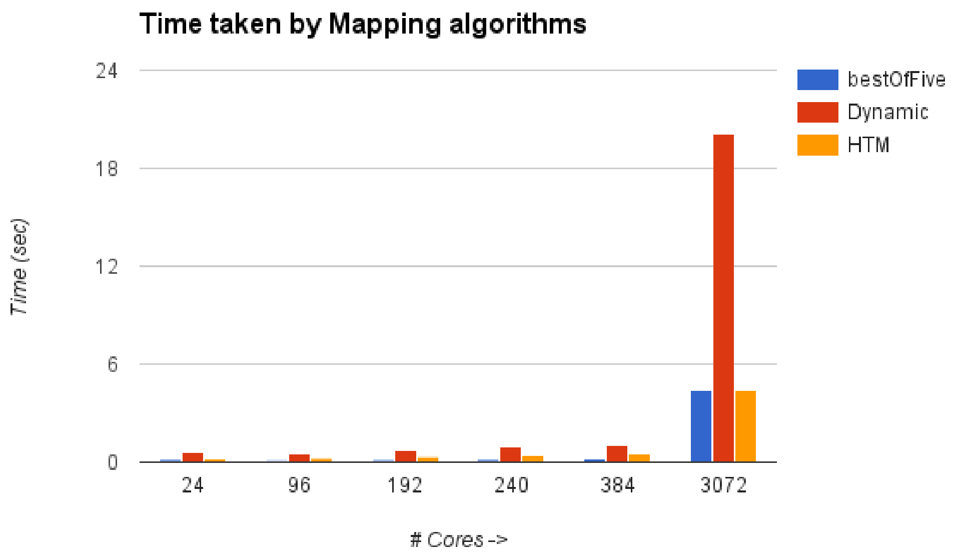
Figure 4.6 shows time taken by various pre-processing steps like running smaller instance run for communication volume generation, determination of coordinates for each processor and mapping time taken by individual mapping algorithm. However we have not considered thes pre-processing times for Default and NUMA mapping, because the molecular systems under consideration (chain benchmark here) do not need pre-processing externally. We see that time taken for determining coordinates of all the processors is fairly constant and this is quite expected since all the processes in the respective processors need to simply read a file ($/proc/cray_xt/cname$) and print the coordinates in a file. Due to poor scalability of LAMMPS, time taken for running smaller instance runs for generating communication volume amongst processes increases with increasing cores.



(a) Determine communication volume among processes



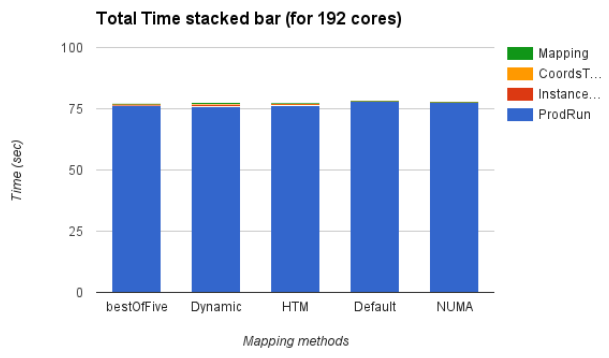
(b) Determine coordinates of each processor



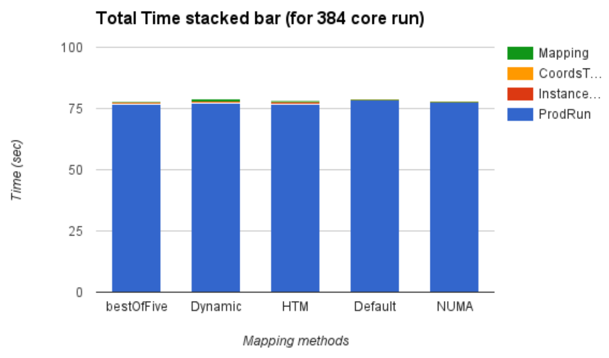
(c) Generating mapping for different mapping algorithms

Figure 4.6: Time taken for various pre-processing steps required for all the mapping algorithms (except NUMA) for varying number of cores

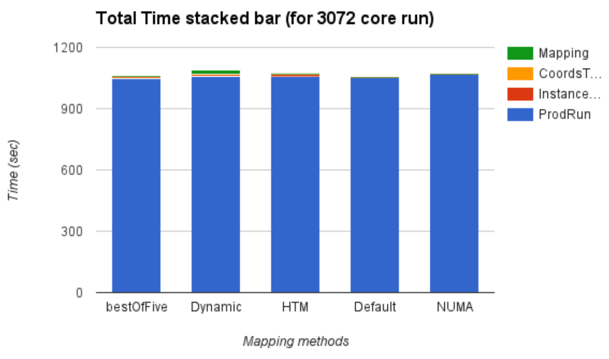
Figure 4.6c shows time taken by various mapping algorithms for ($bestOfFive$, $Dynamic$, HTM) for varying number of cores. We see that for all the core counts, our mapping algorithm $bestOfFive$ takes the minimum time to generate mapping. However, our mapping algorithm $bestOfFive$ takes the maximum time to generate the mapping because it needs to approximate the communication time for mapping of all the mapping depths and then output the best mapping corresponding to mapping depth that resulted in minimum communication time. Parallelizing the mapping process can further reduce the mapping algorithm.



(a) 192 core run



(b) 384 core run



(c) 3072 core run

Figure 4.7: Plot showing total time taken for different mapping algorithms for different core count

Figure 4.7 and table 4.2 show total time, considering the pre-processing time required for all the mapping methods. Since $Default$ and $NUMA$ mapping do not need any pre-processing, hence no value from figure 4.6 have been considered for them. We see improvements for some cores over default mapping, with maximum improvement of about 1.3% for $bestOfFive$ and 0.7% for $Dynamic$.

Cores	bestOfFive	Dynamic	HTM	Default	NUMA
24	63.686604	64.363622	63.81817	63.6971	63.3103
96	71.166105	72.18208	71.676338	71.453966	70.2476
192	77.063118	77.50034	77.345718	78.086033	77.6608
240	82.978258	84.478502	83.138025	82.736866	82.303966
384	77.631952	78.960159	78.125419	78.413566	77.694733
3072	1060.247657	1086.934477	1072.774657	1054.982	1066.68

Table 4.2: Table showing total time taken for different mapping algorithms (including all pre-processing steps like collecting communication volume, determining coordinates for each processor and performing mapping)

Chapter 5

Conclusion and Future Work

In this chapter, we conclude with throwing light on future work on similar lines.

5.1 Conclusion

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.

5.2 Future Work

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[25]), 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 [16] 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.

Bibliography

[1] 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. vii, 20

[2] METIS: Graph Partitioning Tool. http://glaros.dtc.umn.edu/gkhome/views/metis. 11

[3] Ahmed H Abdel-Gawad, Mithuna Thottethodi, and Abhinav Bhatele. Rahtm: routing algorithm aware hierarchical task mapping. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 325–335. IEEE Press, 2014. 9

[4] Tarun Agarwal, Ashok Sharma, and Laxmikant 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*, pages 10–pp. IEEE, 2006. 8

[5] Takanobu Baba, Yoshifumi Iwamoto, and Tsutomu Yoshinaga. A network-topology independent task allocation strategy for parallel computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 878–887. IEEE Computer Society Press, 1990. 8

[6] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91 (1):43–56, 1995. 8

[7] Abhinav Bhatele, Nikhil Jain, William D Gropp, and Laxmikant V Kale. Avoiding hot-spots on two-level direct networks. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 76. ACM, 2011. vii, 9, 10, 12, 13

[8] Kevin J Bowers, Edmond Chow, Huafeng Xu, Ron O Dror, Michael P Eastwood, Brent Gregersen, John L Klepeis, Istvan Kolossvary, Mark Moraes, Federico D Sacerdoti, et al. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 43–43. IEEE, 2006. 8

[9] Ümit Çatalyürek and Cevdet Aykanat. Patoh (partitioning tool for hypergraphs). In *Encyclopedia of Parallel Computing*, pages 1479–1487. Springer, 2011. 18

[10] Mehmet Deveci, Sivasankaran Rajamanickam, Vitus J Leung, Kevin Pedretti, Stephen L Olivier, David P Bunde, Umit V Catalyurek, and Karen Devine. Exploiting geometric partitioning in task mapping for parallel computers. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 27–36. IEEE, 2014. vii, 8, 9, 10, 11, 13

[11] Mehmet Deveci, Kamer Kaya, Bora Uçar, and Umit V Catalyurek. Fast and high quality topology-aware task mapping. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 197–206. IEEE, 2015. 9, 10, 11, 13

[12] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, James 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*, page 103. IEEE Computer Society Press, 2012. vii, 4

[13] Adam Górecki, Marcin Szypowski, Maciej Długosz, and Joanna Trylska. RedMD reduced molecular dynamics package. *Journal of computational chemistry*, 30(14):2364–2373, 2009. 8

[14] Joachim Hein, Fiona Reid, Lorna Smith, Ian Bush, Martyn Guest, and Paul Sherwood. On the performance of molecular dynamics applications on current high-end systems. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 363(1833):1987–1998, 2005. 2

[15] Bruce Hendrickson and Robert Leland. The chaco users guide: Version 2.0. Technical report, Technical Report SAND95-2344, Sandia National Laboratories, 1995. 8

[16] Torsten Hoefler and Marc Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the international conference on Supercomputing*, pages 75–84. ACM, 2011. 6, 38

[17] Torsten Hoefler, Emmanuel Jeannot, and Guillaume Mercier. An overview of process mapping techniques and algorithms in high-performance computing. *High Performance Computing on Complex Environments*, pages 75–94, 2014. 8

[18] Nikhil Jain, Abhinav Bhatele, Xiang Ni, Nicholas J Wright, and Laxmikant V Kale. Maximizing throughput on a dragonfly network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 336–347. IEEE Press, 2014. 9, 12

[19] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of computational chemistry*, 26(16):1781–1802, 2005. 8

[20] Gilles Paul Pieffet. *The application of molecular dynamics simulation techniques and free energy*. PhD thesis, University of Groningen, 2005. 1

[21] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995. 2, 8

[22] Bogdan Prisacari, German Rodriguez, Philip Heidelberger, Dong Chen, Cyriel Minkenberg, and Torsten 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*, pages 129–140. ACM, 2014. 9

[23] Romelia Salomon-Ferrer, David A Case, and Ross C Walker. An overview of the amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013. 8

[24] Hari Subramoni, Sreeram Potluri, Krishna Kandalla, B Barth, Jérôme Vienne, Jeff Keasler, Karen Tomko, K Schulz, Adam Moody, and Dhabaleswar 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*, page 70. IEEE Computer Society Press, 2012. 9, 11

[25] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11 (2):350–361, 1982. 37

[26] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. NWChem:

a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010. 8

[27] Junmei Wang and Tingjun Hou. Application of molecular dynamics simulations in molecular property prediction. 1. density and heat of vaporization. *Journal of chemical theory and computation*, 7(7):2151–2165, 2011. 1

[28] Jingjin Wu, Zhiling Lan, Xuanxing Xiong, Nickolay Y Gnedin, and Andrey 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*, page 75. IEEE Computer Society Press, 2012. 9, 18, 20

[29] Jingjin Wu, Xuanxing Xiong, and Zhiling Lan. Hierarchical task mapping for parallel applications on supercomputers. *The Journal of Supercomputing*, 71(5):1776–1802, 2015. vii, 9, 10, 11, 18, 26