Optimizing Parallel Processes to Nodes Mapping in Contemporary High Performance Interconnection Topologies

Master Project

Natural Science Faculty of the University of Basel

Department of Mathematics and Computer Science

High Performance Computing Group

Submitted by:

Fatjon Lala

Matriculation No.:

18-063-412

Email:

fatjon.lala@stud.unibas.ch

Examiner:

Prof. Dr. Florina M. Ciorba

Supervisor:

Jonas H. Müller Korndörfer

Submission date, place:

31.12.2020, Basel

Contents

Summary

Parallel applications' performance is influenced by the mapping (kwown also as placement) of the processes onto the computing nodes, the frequency and volume of exchanges among the processing elements, the network capacity, and the routing protocol, among others. A poor mapping of application processes degrades performance and wastes computing resources. The mapping of an application is a well-known NP-hard problem, also known as task-to-nodes mapping problem. Mapping processes in an application- and topology-aware manner is expected to minimize application performance degradation and optimize system resource usage. Both goals are critical for advancing scientific discovery and the efficient use of high performance parallel and distributed computing systems. This project focuses on exploring LibTopoMap (`htor.inf.ethz.ch/research/mpitopo/libtopomap/`) [15][14], a generic mapping library and reproducing the work with different applications. That will help us find opportunities for improvements and extension of LibTopoMap with another library named MapLib (`github.com/unibas-dmi-hpc/MapLib`) [21], which is a library that provides algorithms for generating mapping of processing elements to processor unities. The extension of the libraries should result in supporting more mapping algorithms and more processor network topologies in the resulting mapping library.

1 Introduction

Every year the list of High Performance Computing (HPC) systems [9] changes due to different improvements done on these systems. One of the improvements that one might think of is increasing number of computing elements. The increasing number of the computing elements inside the HPC systems, increases the number of problems that are needed to be solved in these systems. These problems among researches are known as bottlenecks which interfere with the parallel applications' performance. One of the most important bottlenecks is the communication between processes executing on HPC systems.

A lot of research is being conducted to reduce the communication time between processes in a HPC system. To achieve a better performance, an efficient mapping of tasks to processes, of a parallel application, is needed. Reducing the communication time is a well known problem, also known as task to nodes mapping problem. It is shown in [15][14] to be a NP-hard problem. Therefore, to solve this problem, we need to find a sub-optimal solution. Different mapping libraries consider the problem as a graph problem and try to find the solution using different algorithms that work on graphs.

Regarding that the problem is still not solved, a lot of evaluation metrics are presented to measure possible sub-optimal solutions. The most important evaluation metrics are *congestion* and *dilation* as shown in [15] [17].

In this project we are focused on exploring a generic library named LibTopoMap [15]. We try to reproduce the work done presented by Hoefler et al., [15] with different applications. The purpose of this work is to evaluate the efficiency of different mapping strategies and network topologies. That will help us find opportunities for improvements and extension of LibTopoMap with another library named MapLib [3], which is a library that provides algorithms for generating mapping of processing elements to processor unities. The extension of the libraries should result in supporting more mapping algorithms and more processor network topologies in the resulting mapping library.

This work is organized as follows. In Section 2, the related work for our topic is carefully reviewed. Some background information about the mapping problem is described in Section 3. The workflow that is used for this project is presented in Section 4. Section 5 provides the information about the experimental setup, the results for the benchmarking experiments and a discussion regarding the performance based on evaluation metrics. Section 6 concludes the work and mentions the future work(s) on this topic.

2 Related Work

Topology mapping problem is an interesting topic among researchers. Different researches go beyond on-node mapping and consider also the in-node mapping as a second scale for performance gains. Therefore, the complexity of parallel application can be efficiently exploited with the right task mapping algorithm.

Hoefler et al., [15] present LibTopoMap, the generic mapping library. LibTopoMap considers different fast heuristics such as (*greedy heuristic algorithm*), bisection mapping (*recursive bisection mapping*) and mapping based on graph similarity (*Reverse Cuthill McKee (RCM)*[12]). Hoefler et al., also show that the benefit of topology mapping grows with the network size. Their mapping strategies have shown to reduce network congestion up to 80%, reduce average dilation up to 50%, and improve benchmarked communication performance by 18%.

Pilla et al. [19] present a load balancing algorithm named HwTopoLB. This algorithm is topology-aware and works on clustered multi-core machines. The goal of the algorithm is to reduce idle time and the communication delay between computation nodes. With this approach, based on their experimental results, the performance improvement was 23% when executing without load balancers and 19% when executing with the existing load balancing strategies on different clusters(systems).

Deveci et al., [13] discuss about the mapping problem in the context of geometric partitioning. They consider sparse node allocation in a parallel machine. Based on this consideration, their goal is to map tasks, that are more dependent, to computations nodes close to each other. With this approach the distance that the messages need to travel is lower and therefore also the congestion in the network is lower, which would end up in reduced cost for the overall communication. Their results show an improvement with 15% in MiniGhost application (mini application) and 10% in MiniMD (molecular dynamics mini application) compared to LibTopoMap. Even thought their results shown improvement, they have not tested their methods with large applications, and therefore we can not come to the conclusion that it will always end up in a performance improvement.

Jeannot et al., [16] introduce extended version of TreeMatch, named TopoMatch. It is a partially distributed algorithm that performs an optimized process placement in multi-core parallel machines. The goal of this algorithm is to improve performance of parallel applications by matching the patterns of communication with the hardware architecture that it runs on. As in LibTopoMap, this algorithm has some phases such as: getting the information about the application communication pattern, modeling the hardware architecture and

computing the process placement. After computing the re-ordering, the algorithm makes it possible to map the application based on the new order, to reduce communication time.

Wu et al., [22] take in consideration two types of mapping: inter-node mapping and intra-node mapping. They also consider two types of network topologies: fat tree and torus and two mapping algorithms: *generic recursive tree mapping algorithm* and *recursive bi-partitioning mapping algorithm* for torus topology. Their results show a significantly improvement of the communication performance up to 77%.

3 Background

We have explored LibTopoMap, a generic topology mapping library [15], in this project. This library addresses the problem of task-to-nodes mapping by using different mapping algorithms, graph communication patterns and network topologies. In the following subsections we have explained the most important parts that LibTopoMap needs in order to be used. Due to time constraints and complications during the exploration of LibTopoMap we could not manage to explore MapLib.

3.1 Evaluation Metrics

In this section we will discuss about the evaluation metrics used in this project. As explained in [15], LibTopoMap considers *congestion* and *dilation* as evaluation metrics. The main goal is to reduce the maximum congestion or better know as worst-case congestion. In our experiments we are focused more on *congestion* measure. Before showing the notations for these metrics, first we need to get familiar with some terms and formulas that are used in order to derive to the main metrics (detailed in [15]), as following:

- **Diameter**: maximum distance between two processes.
- **Bisection bandwidth**: the minimum total bandwidth of links that need to be cut in order to divide the processors into two equal sets.
- **Topology mapping**: finding the allocation of processes to nodes such that the sparse application communication topology efficiently utilizes the physical links in the network.
- **Logical communication pattern:**

$$G = (V_G; w_G)$$

where:

V_G	set of processes
w_G	the weight of the edge connecting u to v. Represents the volume of communication from u to v (0 if there is no connection)

- **Physical communication pattern:**

$$H = (V_H; C_H; c_H; R_H)$$

where:

V_H	set of physical nodes (processes and switches)
$C_H(u)$	number of processes that can be hosted to $u \ 2 \ V_H$ (0 if it is switch)
$c_H(uv)$	bandwidth of the link connecting u to v
R_H	routing algorithm

As we can see, this approach takes in consideration two types of communication patterns. The logical communication pattern is a weighted, directed graph that defines the volume communication between processes (by the weight of the edge when the communication occurs) or 0 when the communication does not occur [15]. The physical communication pattern represents the network interconnection of physical nodes (processors and switches) [15]. Based on these communications, the calculations for the two main metrics, *dilation* and *congestion*, are done.

- **Dilation(uv)**: the average length of path taken by a message sent from u to v [15].

$$Dilation(uv) = \sum_{u;v2V_G} R_H(\ _{(u)}\ \ _{(v)})(p) \ \ j\,p\,j$$

where:

$R_H(\ _{(u)}\ \ _{(v)})(p)$	fraction of the traffic from u to v that is routed through p, when a mapping exists
j p j	length of the path p
	function that maps $V_G \ ! \ V_H$

There are different calculations for *dilation* in [17][11], but we will be focused to use the ones from LibTopoMap since our work is focused on this library. In short, *dilation* means the

number of edges traversed by packets, which is a measure of the total "communication work" performed by interconnection network [15]. *Congestion* of a link *uv* of the interconnection network is the ratio between the amount of traffic on that link and the capacity of the link.[15]

- **Congestion(uv)** is computed by the traffic on the link and it capacity as follows:
  - **Traffic(uv)**:

$$Traffic(uv) = \sum_{u;v2V_G} w_G(uv) \left(\sum_{p2P(\ (u) \ (v));e2p} R_H(\ (u) \ (v))(p) \right)$$

    where:

$w_G(uv)$	the weight of the edge connecting u to v. Represents the volume of communication from u to v (0 if there is no connection)
$R_H(\ (u) \ (v))(p)$	fraction of the traffic from u to v that is routed through p, when a mapping exists function that maps $V_G \ ! \ V_H$

  - **Congestion(uv)**:

$$Congestion(uv) = \frac{Traffic(uv)}{c_H(uv)}$$

    where:

$c_H(uv)$	capacity of the link connecting edge (uv)

In this project, *the worst case congestion* is considered. This means that the maximum over all congestions for each vertice of the graph is measured as a result. The same measure is also done when re-mapping (if the worst congestion of the original mapping is higher then the worst congestion of the re-mapping). So:

$$Congestion(\ ) = max_{uv} Congestion(uv)$$

3.2 Mapping algorithms

LibTopoMap considers different mapping algorithms to do the re-mapping of tasks-to-nodes. These different techniques consider heuristic approach (*greedy heuristic algorithm*), bisection mapping (*recursive bisection mapping*) and mapping based on graph similarity (*Reverse Cuthill McKee (RCM)*[12]).

The heuristic approach, *greedy heuristic algorithm*, is based on the following execution steps [15]:

1. Algorithms starts from a vertex in H (explained in Subsection 3.1).
2. It chooses the heaviest vertex in G (explained in Subsection 3.1).
3. Maps greedily, the heaviest vertices from G to the heaviest neighbor vertices to H with the heaviest communication.
4. Performs recursively steps 1, 2, 3, until the algorithm stops.

Bisection mapping divides the graph into two equal halves recursively by determining the minimum over edge weights. It uses SCOTCH [18] and METIS [20] libraries for this achieve this purpose.

Graph similarity approach to map tasks to nodes is a well know technique. The basic idea is to make adjacency matrices between the physical and logical topology mapping into similar shape. *Reverse Cuthill McKee* (RCM) algorithm is used to solve the reduction of bandwidth problem in a heuristic way.

3.3 Applications and Communication Matrices

In this subsection we discuss about the applications and communication matrices used for experimental purposes in our work. To explore the application, we have used the sparse matrix collection, SuiteSparse [6] (formerly known as University of Florida sparse matrix collection). This collection is distributed along with LibTopoMap to test the installation of the library. It is collected from a wide range of applications, from different domains, and commonly used for benchmarking purposes. Figure 3, shows a part of the structure and the information that this sparse matrix collection has. We have also considered other applications during our experiments such as: the applications CG and BT-MZ from NAS parallel benchmarks [10][5], AMG and LAGHOS from CORAL2 [8] and LULESH from CORAL [7]. From these applications we have considered the communication matrices taken from Korndörfer et al. [17]. These application were executed using 64 processes, therefore their communication matrix is a 64 x 64 matrix and they show the amount of data transferred from one process to another during the whole execution.

3.4 Network Topology

One of the most important parts of the communication between processes is *network topology*. Since LibTopoMap considers the on-node mapping, which is a mapping of tasks to computing nodes, we can say that network topology can be a bottleneck to the performance. Therefore, in this project we consider two types of network topology: *3x3x3 torus* (provided by LibTopoMap distribution) and a *two-level fat tree* (miniHPC network topology[4], detailed in Section 4). A graphical illustration of both network topologies used in this project is shown in Figure 3.

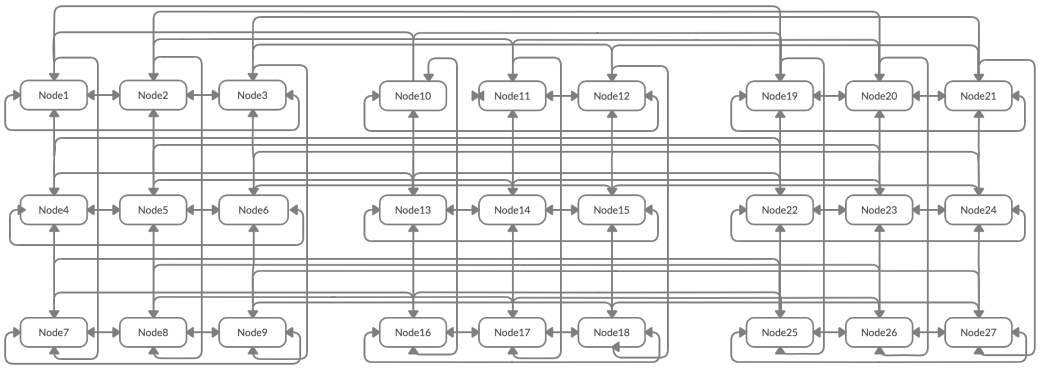


Figure 1: 3x3x3 direct torus network topology (illustrated in 2D mode).

As shown in Figure 1, 3D torus is a direct network topology since the computing nodes are connected directly with each-other without any switches.

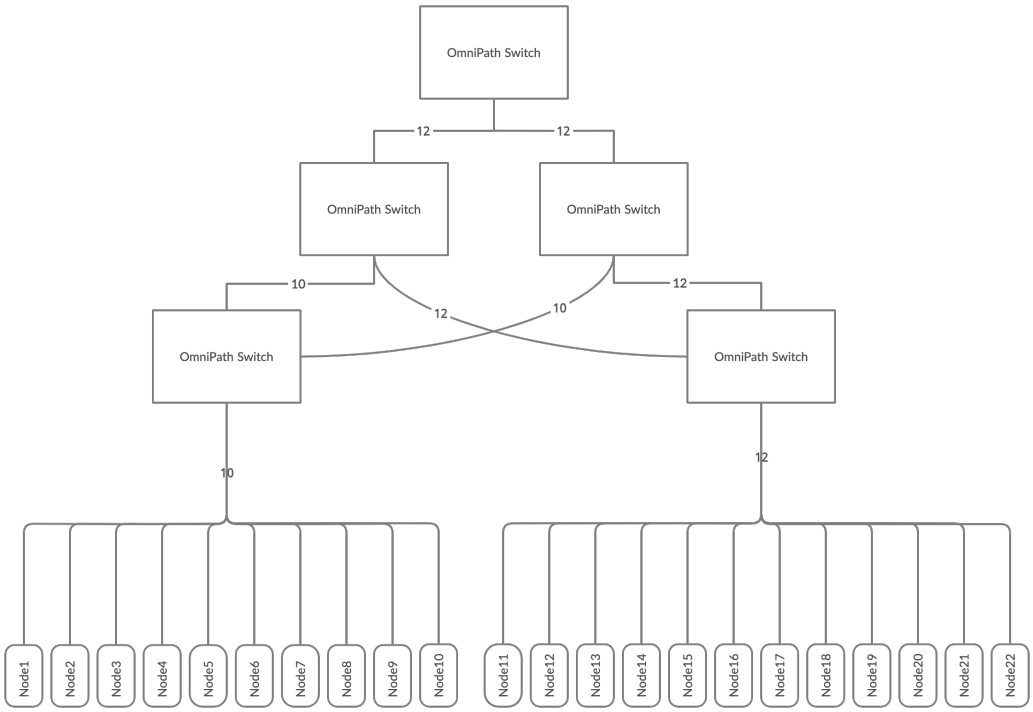


Figure 2: Two-level fat tree network topology.

In Figure 9, a two-level fat tree (switch based) network topology is illustrated. This topology is non-blocking (due to the equality of input and output channels for each switch) and is the same used in miniHPC [4], for which you will find detailed information in Section 4, with some small changes. The *storage*, *login* and *KNL* nodes are not considered in the experimental setup. In Appendix A, you will also find the configuration files of the network topology used in this project.

3.5 LibTopoMap Assumptions

This library is focused to achieve on-node mapping while taking into consideration the application communication pattern and also the network topology in which the application is executed on. In this library, there are also some assumptions and constraints, according to the authors [15], that have to be considered when benchmarking different applications as following:

1. each process can store the whole graph,
2. on-node mapping (after the global communication graph has been mapped to the network), an infinite in-node bandwidth is assumed.

4 Proposed Workflow

The initial step to use LibTopoMap is to get to understand the inputs that it considers. In the paper presented from Hoefler et al., [15] and in the website `https://htor.inf.ethz.ch/research/mpitopo/libtopomap/`, there is not a clear information of how and what LibTopoMap needs as an input. Therefore, we started our exploration with the command that is provided to test LibTopoMap:

```
$ mpirun -n 12 ./reader 0 ./aug2dc/aug2dc.mtx ../3x3x3.map ../3x3x3.fake
```

From the execution command we could observe that LibTopoMap is taking in consideration some inputs. According to our observation, *reader*, is a gateway application that is using the matrix (*aug2dc.mtx*), the mapping file (*3x3x3.map*) and the nodes naming (host names) file (*3x3x3.fake*) in this case. The sparse matrix collection SuiteSparse (introduced in Subsection 3.3) has the following structure as shown in Figure 3.

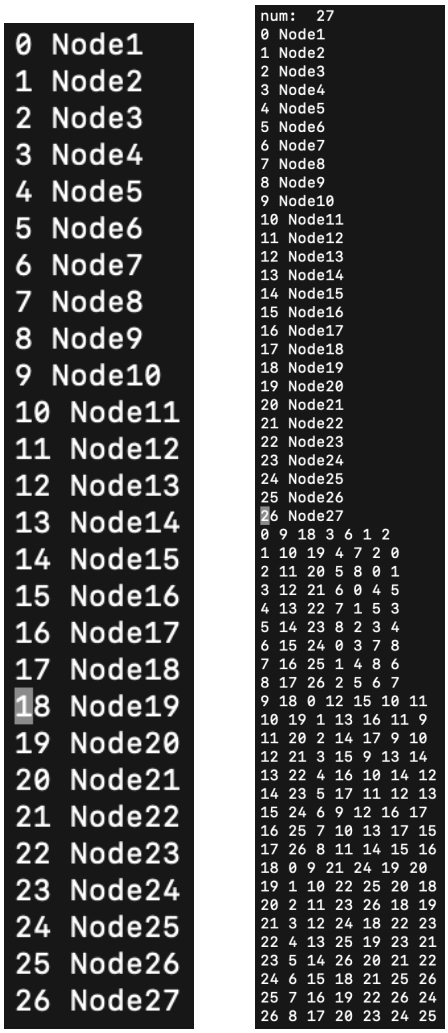
```
%%MatrixMarket matrix coordinate integer symmetric
%-------------------------------------------------------------------------------
% UF Sparse Matrix Collection, Tim Davis
% http://www.cise.ufl.edu/research/sparse/matrices/GHS_indef/aug2dc
% name: GHS_indef/aug2dc
% [Gould, Hu, Scott:   expanded system-2D PDE (CUTEr)]
% id: 1216
% date: 1994
% author: N. Gould
% ed: N. Gould
% fields: name title A id date author ed kind
% kind: 2D/3D problem
%-------------------------------------------------------------------------------
30200 30200 40000
20201 1 1
20202 1 -1
20201 2 1
20301 2 -1
20202 3 1
20203 3 -1
20202 4 1
20302 4 -1
20203 5 1
20204 5 -1
20203 6 1
20303 6 -1
20204 7 1
20205 7 -1
20204 8 1
20304 8 -1
20205 9 1
20206 9 -1
20205 10 1
20305 10 -1
20206 11 1
20207 11 -1
20206 12 1
20306 12 -1
```

Figure 3: SuiteSparse(formerly know as University of Florida) sparse matrix collection. [6]

By looking at the matrix, we have derived the information that the first line (after the description part) provides the information about the rows, columns and the number of

non-zero elements. We are not sure how this matrix or graph communication pattern is derived because we do not have the information for the application(s). For the same reason, we also do not know what exactly the values in this graph communication pattern show. In the workflow that we have followed, we assume that graph communication pattern is derived from the communication matrix of the application. We also show how to achieve this conversion.

The other inputs that are being used are, the node mapping file and node naming file, as shown in Figure 4.



(a) Node naming (host names) file.

(b) Node mapping file.

Figure 4: Configuration file for 3x3x3 (3D) Torus network topology.

The configuration file shown in Figure 4 refers to the network topology introduced in

Subsection 3.4 and visualized in Figure 1. As explained in LibTopoMap website [2], the node naming file defines the host names of the nodes used in the network topology. The node mapping file defines three important informations:

1. Number of vertices (switches or nodes) in line 1.
2. Host names for each vertex from line 2 to <number of vertex> + 1 line.
3. Adjacency list of each vertex from line <number of vertex> + 1 to the end of the file.

Regarding this valuable information that we could get from our observation we propose the workflow that we have been using during this project. The workflow that we have followed during this project is a as shown in Figure 5:

Figure 5: Proposed workflow.

As shown in Figure 5, the first step of the workflow is to execute the applications. First of all, the applications are executed in order to extract the communication matrices between processes. The information in the communication matrices can represent different quantities such as number of message exchanges, the volume of exchanges, the average message transfer time, and others [17]. All our communication matrices have been taken from Korndörfer et al. [17], and we are focused on volume (in Byte) of point-to-point message exchanges. This means that we have to take in consideration the following structure:

$$cell[p_i; p_j] = \begin{cases} 0 & \text{if } i \neq j \text{ and } p_i \text{ does not communicate with } p_j \\ n & \text{if } i \neq j \text{ and } p_i \text{ communicates with } p_j \end{cases}$$

where:

p	process.
cell$[p_i; p_j]$	the cell for process (p_i) and process (p_j).
n	the value for cell$[p_i; p_j]$ based on the communication matrix.

All of the communication matrices are saved in CSV (comma-separated value) files. To create graph communication patterns we have used a python-based script that considers the CSV files and does the conversion, from communication matrices to graph communication patterns. A visualization of this transformation you can find it in Figure 6 below:

$$\begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & p_3 & p_4 \end{array} \\ \begin{pmatrix} 296400000 & 296432832 & 296432832 & 0 \\ 296432832 & 0 & 0 & 296432832 \\ 296432832 & 0 & 0 & 296432832 \\ 0 & 296432832 & 296432832 & 0 \end{pmatrix} \end{array}$$

(a) Communication matrix.

$$\begin{pmatrix} p_1 & p_1 \\ p_1 & p_2 \\ p_1 & p_3 \\ p_2 & p_1 \\ p_2 & p_4 \\ p_3 & p_1 \\ p_3 & p_4 \\ p_4 & p_2 \\ p_4 & p_3 \end{pmatrix} \begin{array}{c} 296400000 \\ 296432832 \\ 296432832 \\ 296432832 \\ 296432832 \\ 296432832 \\ 296432832 \\ 296432832 \\ 296432832 \end{array}$$

(b) Graph communication pattern of (a).

Figure 6: Example of the conversion from CG benchmarking application.

```
%%MatrixMarket matrix coordinate integer symmetric
%
64 64 132
1 1 296400000
2 1 296432832
3 1 296432832
4 2 296432832
4 3 296432832
5 1 296432832
6 2 296432832
6 5 296432832
7 3 296432832
7 5 296432832
8 4 296432832
8 6 296432832
8 7 296432832
9 2 296400000
10 9 296432832
10 10 296400000
11 9 296432832
12 10 296432832
12 11 296432832
13 9 296432832
14 10 296432832
14 13 296432832
```

Figure 7: CG application graph communication pattern.

Figure 7 presents the graph communication pattern of CG application that we used in the our work. In the first line (after the title of the file), it is defined the number of rows, columns and non-zero elements that this communication pattern has. In the next lines, there is the information about the row process (row process rank of the communication matrix), column process (column process rank of the communication matrix) and the volume (in

Byte) of point-to-point message exchanges.

After we converted all communication matrices to graph communication patterns and defined the network topologies (refer to Subsection 3.4), we used them as input for LibTopoMap. Detailed information about the script used for the conversion from communication matrix to graph communication pattern, network topologies configuration files and challenges of this work can be found in the Appendix A. In the following section we will discuss about the results of the experiments.

5 Performance Results and Discussion

5.1 Experimental Setup

5.1.1 HPC System

All the experiments where executed using miniHPC cluster `https://hpc.dmi.unibas.ch/en/research/minihpc/`, provider from High Performace Computing research group at University of Basel. The miniHPC has a peak performance of 28.9 double precision TFLOP/s. The miniHPC has two types of nodes, Intel Xeon nodes and Intel Xeon Phi Knights Landing (KNL) nodes. The Intel Xeon nodes amount to 22 computing nodes, 1 login node, and 1 node for storage. The Intel Xeon Phi nodes amount to 4 computing nodes. All nodes are interconnected through two different types of interconnection networks. The first network is an Ethernet network with 10 Gbit/s speed, reserved for users and administrators access. The second network is the fastest network, an Intel Omni-Path network with 100 Gbit/s speed, reserved for the high-speed communication between the computing nodes. The topology of this second network interconnects the 28 nodes (24 Xeons and 4 KNLs) of the miniHPC cluster via a two-level fat-tree topology. [4]

5.1.2 Design of Experiments

Before going into the results part, we introduce in Table 1 a summarized information about all the applications, systems, mapping strategies, network topologies and number of processes used for experimental purposes.

Applications and/or communication matrices	**System**	**Number of Processes**	**Mapping Strategies**	**Network Topologies**
SuiteSparse matrix collection	miniHPC	4	RCM	3x3x3 (3D) Torus
AMG		8	GREEDY	
BT-MZ		16	RECURSIVE	
CG		32		Two Level Fat Tree
LAGHOS		64	SCOTCH	
LULESH				

Table 1: Experiments design table.

5.2 Results and Discussion

In this section we will discuss the results taken from the experiments performed in miniHPC [4]. In order to do some benchmarking we had to use some communication matrices taken from different application, as shown in Table 1. These applications are: CG and BT-MZ from NAS parallel benchmarks, AMG and LULESH from CORAL2 benchmark suite [17]. In [17], you can find more details about these applications and their characteristics. These application matrices were taken, while running the applications with 64 processes or MPI ranks. In our work, although we know the number of the processes used to produce these communication matrices, we have run LibTopoMap with different number of processes for testing purposes. As you can also see the result tables, we have run the application with 4, 8, 16, 32 and 64 processes. Since miniHPC [4] has only 22 computing nodes, to make 32 and 64 processes we have use the following configuration:

Number of processes	Number of nodes	Number of MPI ranks per node
32	16	2
64	16	4

Table 2: Nodes configuration for the experiments.

The tables of the results below, are created based on the results taken from the executions performed in miniHPC [4] with differend configurations. These tables have the following information:

1. The name of the application.
2. The original and mapping strategy congestion (refer Subsection 3.1).
3. The number of processes.
4. The name of the mapping strategy used for each congestion.
5. The name of the network topology used (in caption).

The original mapping considers the application to be mapped in a sequential numbering order from the processes with the lowest id value to the ones with the highest id value (eg., 0 1 2 3 4 5 ...). If the congestion value of the mapping strategy is higher or equal to the original congestion value, it means that the re-mapping can not improve the communication time needed for the given specifications. The re-mapping has to be considered when the congestion value of the mapping strategy is lower than the original congestion value.

One of the limitations that we experienced is that due to the installation of LibTopoMap version without SCOTCH[18] , we observed from the results that there is no re-mapping happening because the library is missing, when SCOTCH mapping strategy is used. Another important limitation to mention is that, we experienced for most of the applications (as shown in the result tables), segmentation faults while trying to use LibTopoMap with 32 and 64 processes. We assume that this error is triggered when the number of processes to run LibTopoMap is higher then the number of processes or nodes defined in the network topology, but it is still a assumption. When there is not information about the congestion (defined as - in the result tables), we run into segmentation error and therefore we could not get that information.

	SuiteSparse matrix collection							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	73.00	73.00	73.00	73.00	73.00	73.00	73.00	73.00
8	120.00	120.00	120.00	120.00	123.00	68.00	98.00	120.00
16	81.00	81.00	81.00	81.00	94.00	80.00	106.00	81.00
32	-	83.00	-	-	-	59.00	-	-
64	-	144.00	-	-	-	114.00	-	-

Table 3: SuiteSparse matrix collection results with torus network topology.

	SuiteSparse matrix collection							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	143.00	143.00	143.00	143.00	143.00	143.00	143.00	143.00
8	198.00	198.00	198.00	198.00	198.00	198.00	198.00	198.00
16	321.00	321.00	321.00	321.00	152.00	320.00	152.00	321.00
32	-	361.00	-	-	-	243.00	-	-
64	-	394.00	-	-	-	273.00	-	-

Table 4: SuiteSparse matrix collection results with two level fat-tree network topology.

From Table 3 and 4 we can observe that LibTopoMap can achieve e lower(better) congestion with different mapping strategies when the number of processes increases. If we would look at the results when the number of processes is 4, then we would see that the congestion does not change, therefore no re-mapping can be possible to improve the communication time. As the number of processes increases, then we can also see that some of the mapping

strategies are able to find a lower congestion than the original congestion, and perform a re-mapping. In this specific example the library faced segmentation errors while running with 32 and 64 processes and using RCM, Recursive and SCOTCH as mapping strategies. The only mapping strategy that we could get results for was Greedy, while using 32 and 64 processes.

	AMG (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	256.00	256.00	256.00	256.00	256.00	255.00	256.00	256.00
8	317.00	317.00	317.00	317.00	255.00	255.00	313.00	317.00
16	159.00	159.00	159.00	159.00	172.00	128.00	160.00	159.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 5: AMG application benchmarking results with torus network topology.

	AMG (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	766.00	766.00	766.00	766.00	766.00	766.00	766.00	766.00
8	448.00	448.00	448.00	448.00	448.00	448.00	448.00	448.00
16	949.00	949.00	949.00	949.00	939.00	938.00	948.00	949.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 6: AMG application benchmarking results with two level fat-tree network topology.

As we can see from Table 5 and 6, we have run AMG[1] application with different specifications with LibTopoMap. Based on the results, we can see that the algorithms used in LibTopoMap (RCM, Recursive and Scotch) were not able to find a better mapping than the original mapping, in most of the cases. Therefore their congestion is equal or higher compared to the original congestion. While the mapping strategy named Greedy is able to find a better mapping of tasks-to-nodes than the original, in most of the cases, and therefore the congestion of this strategy is lower than the original one. Also, in the results of this application we observe that the congestion can not be improved when the library is running on a small number of processes. We, can also say that the re-mapping is performed more often when the direct (torus) network topology is used compared to the two level fat-tree (indirect) network topology.

	BT-MZ (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	51.00	51.00	51.00	51.00	51.00	51.00	51.00	51.00
8	56.00	56.00	56.00	56.00	63.00	45.00	51.00	56.00
16	32.00	32.00	32.00	32.00	32.00	27.00	35.00	32.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 7: BT-MZ application benchmarking results with torus network topology.

	BT-MZ (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	131.00	131.00	131.00	131.00	131.00	131.00	131.00	131.00
8	89.00	89.00	89.00	89.00	89.00	89.00	89.00	89.00
16	155.00	155.00	155.00	155.00	170.00	162.00	156.00	155.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 8: BT-MZ application benchmarking results with two level fat-tree network topology.

Table 7 and 8, present the results for the application BT-MZ. Also in this application, as in AMG we observed that there is a slightly improvement on some mapping strategies for both network topologies that we consider. This might come as a result of the application specifications. In most of the cases, the original mapping can not be improved further by using different mapping strategies. The congestion values, either the original or mapping strategy congestion, are relatively small compared to AMG application.

	CG (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
8	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
16	8.00	8.00	8.00	8.00	9.00	6.00	6.00	8.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 9: CG application benchmarking results with torus network topology.

	CG (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
8	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00
16	23.00	23.00	23.00	23.00	27.00	15.00	15.00	23.00
32	-	-	-	-	-	-	-	-
64	-	-	-	-	-	-	-	-

Table 10: CG application benchmarking results with two level fat-tree network topology.

In Table 9 and 10 we can see the results for CG application. The congestion of this

application is very small for both mappings, the original and mapping strategies one. This comes as a result of the characteristics of the application communication. The improvement in this application is very small, and the re-mapping is only possible in a small number of cases. From here we can also see that, when the number of processes is small the probability of the re-mapping to happen is 0. This means that, the original mapping is considered to have a lower congestion and the re-mapping will not decrease it more.

	LAGHOS (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	86.00	86.00	86.00	86.00	86.00	86.00	86.00	86.00
8	94.00	94.00	94.00	94.00	79.00	64.00	61.00	94.00
16	43.00	43.00	43.00	43.00	37.00	41.00	57.00	43.00
32	-	48.00	-	-	-	32.00	-	-
64	-	-	-	-	-	-	-	-

Table 11: LAGHOS application benchmarking results with torus network topology.

	LAGHOS (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	162.00	162.00	162.00	162.00	162.00	162.00	162.00	162.00
8	119.00	119.00	119.00	119.00	119.00	119.00	119.00	119.00
16	184.00	184.00	184.00	184.00	195.00	210.00	157.00	184.00
32	-	254.00	-	-	-	241.00	-	-
64	-	-	-	-	-	-	-	-

Table 12: LAGHOS application benchmarking results with two level fat-tree network topology.

In LAGHOS application, Table 11 and 12, we can observe that the re-mapping of the application is possible nearly in 60% of the cases in both topologies. This might come as a result of the application communication even though we are not quite sure about it. In this application, we could get the congestion results while running with 32 processes and only for Greedy mapping strategy, while for the other mapping strategies we run into segmentation errors.

	LULESH (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	40.00	40.00	40.00	40.00	40.00	40.00	40.00	40.00
8	65.00	65.00	65.00	65.00	36.00	32.00	37.00	65.00
16	35.00	35.00	35.00	35.00	34.00	29.00	39.00	35.00
32	-	28.00	-	-	-	20.00	-	-
64	-	-	-	-	-	-	-	-

Table 13: LULESH application benchmarking results with torus network topology.

	LULESH (commMatrix size)							
	Original Congestion				Mapping Strategy Congestion			
Number of processes	RCM	Greedy	Recursive	SCOTCH	RCM	Greedy	Recursive	SCOTCH
4	90.00	90.00	90.00	90.00	90.00	90.00	90.00	90.00
8	61.00	61.00	61.00	61.00	61.00	61.00	61.00	61.00
16	109.00	109.00	109.00	109.00	111.00	111.00	111.00	109.00
32	-	153.00	-	-	-	107.00	-	-
64	-	-	-	-	-	-	-	-

Table 14: LULESH application benchmarking results with two level fat-tree network topology.

In Table 13 and 14, the results for the LULESH application are presented. This application follows nearly the same mapping characteristics as LAGHOS application. This might be the case as a result of similarity of the intensity of communication between processes in both applications.

From the results above we could learn some of the LibTopoMap characteristics. Firstly, we can mention the segmentation fault error while trying to use more processes than defined in the network topology configuration file. This might be a limitation that LibTopoMap has which should be considered while using the library for different benchmarking experiments. Secondly, as we observed for most of the applications that we used, LibTopoMap was not able to find a better mapping when the number of processes used in the system was relatively small. This reason might be considered as a future work improvement for LibTopoMap algorithms.

6 Conclusions and Future Work

From this project, we could get a lot of knowledge about the task-to-nodes mapping problem, which indeed has been (is) an interesting topic for the community. Also, a better understanding about the parallel applications and machines could be retrieved from this project. In the end, also a hands-on experience with the libraries developed from the researchers was part of the lessons learned from this work.

Based on our exploration, we could achieve to perform experiments with different applications and different network topologies. To enhance LibTopoMap with more algorithms used as mapping techniques, we will have to explore MapLib [3] and make both libraries work together. Another possible future work will be to consider, based on the re-mapping we could get from LibTopoMap, the re-mapping of the application while being executed to measure the execution time. Therefore we will have both execution times of the application, with and without re-mapping, which would help us measure the performance gain for every

application when the re-mapping is possible. Lastly but not less important point to be considered as a future work would be to also perform some intra node (in-node) mapping, from which Jeannot et al. [16] have shown some promising results.

References

[1] Application AMG from CORAL2 Benchmarks. https://asc.llnl.gov/coral-2-benchmarks. (Accessed on December 20, 2020).

[2] LibTopoMap website. https://htor.inf.ethz.ch/research/mpitopo/libtopomap/. (Accessed on December 20, 2020).

[3] MapLib GitHub repository. https://github.com/unibas-dmi-hpc/MapLib. (Accessed on December 20, 2020).

[4] miniHPC: small but modern HPC. https://hpc.dmi.unibas.ch/en/research/minihpc/. (Accessed on December 20, 2020).

[5] The NAS Parallel Benchmarking Suite. https://www.nas.nasa.gov/publications/npb.html. (Accessed on December 20, 2020).

[6] SuiteSparce Matrix Collection. https://www.cise.ufl.edu/research/sparse/matrices/GHS_indef/aug2dc. (Accessed on December 20, 2020).

[7] The CORAL Benchmarks Suite. https://asc.llnl.gov/coral-2-benchmarks. (Accessed on December 20, 2020).

[8] The CORAL2 Benchmarks Suite. https://asc.llnl.gov/coral-benchmarks. (Accessed on December 20, 2020).

[9] TOP500 supercomputers site. https://top500.org/. (Accessed on December 20, 2020).

[10] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

[11] Abhinav Bhatelé, Gagan Raj Gupta, Laxmikant V Kalé, and I-Hsin Chung. Automated mapping of regular communication graphs on mesh interconnects. In *2010 International Conference on High Performance Computing*, pages 1–10. IEEE, 2010.

[12] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

[13] Mehmet Deveci, Sivasankaran Rajamanickam, Vitus J Leung, Kevin Pedretti, Stephen L Olivier, David P Bunde, Umit V Çatalyürek, and Karen Devine. Exploiting geometric partitioning in task mapping for parallel computers. In *2014 IEEE 28th international parallel and distributed processing symposium*, pages 27–36. IEEE, 2014.

[14] Torsten Hoefler, Rolf Rabenseifner, H. Ritzdorf, Bronis R. de Supinski, Rajeev Thakur, and Jesper Larsson Träff. The Scalable Process Topology Interface of MPI 2.2. *Concurrency and Computation: Practice and Experience*, 23(4):293–310, Aug. 2010.

[15] 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, 2011.

[16] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. Process placement in multicore clusters: Algorithmic issues and practical techniques. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):993–1002, 2013.

[17] Jonas H Müller Korndörfer, Mario Bielert, Laércio L Pilla, and Florina M Ciorba. Mapping matters: Application process mapping on 3-D processor topologies. *arXiv preprint arXiv:2005.10413*, 2020.

[18] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[19] Laércio L Pilla, Christiane P Ribeiro, Pierre Coucheney, François Broquedis, Bruno Gaujal, Philippe OA Navaux, and Jean-François Méhaut. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. *Future Generation Computer Systems*, 30:191–201, 2014.

[20] Kirk Schloegel, George Karypis, and Vipin Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[21] Viacheslav Sharunov. Optimized parallel tasks to nodes mapping in 3-D high performance interconnection topologies. Master's thesis, University of Basel, 2017.

[22] Jingjin Wu, Xuanxing Xiong, and Zhiling Lan. Hierarchical task mapping for parallel applications on supercomputers. *The Journal of supercomputing*, 71(5):1776–1802, 2015.

A Appendix

A.1 Steps followed to install LibTopoMap

First we tried with ParMETIS v-4.0.3-foss-2016a and METIS v-5.1.0-foss-2016a that are installed in miniHPC. Following the suggestion to try different compilers accordingly: **mpicxx** and **mpic++** for OpenMPI and **mpiicpc** for intel.

- For OpenMPI and **mpicxx** compiler we did try the following versions:
  - OpenMPI/2.0.2-GCC-6.3.0-2.27-opa
  - Result: error idxtype
  - OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27-opa
  - Result: error idxtype
  - OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27
  - Result: error idxtype
  - OpenMPI/3.0.0-GCC-7.2.0-2.29
  - Result: error idxtype
  - Some of the other versions of OpenMPI I could not load.
- For OpenMPI and **mpic++** compiler we did try the following versions:
  - OpenMPI/2.0.2-GCC-6.3.0-2.27-opa
  - Result: error idxtype
  - OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27-opa
  - Result: error idxtype
  - OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27
  - Result: error idxtype
  - OpenMPI/3.0.0-GCC-7.2.0-2.29
  - Result: error idxtype
  - Some of the other versions of OpenMPI I could not load.
- For compiler **mpiicpc** we did try different intel versions:
  - intel/2018a

- Result: errors in compiling
- intel/2018b
- Result: errors in compiling
- intel/2019a
- Result: errors in compiling
- intel/2019b
- Result: errors in compiling

Compiler	OpenMPI version	Result	Intel version
mpicxx	OpenMPI/2.0.2-GCC-6.3.0-2.27-opa	Error __GKfree	-
mpicxx	OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27-opa	Error __GKfree is undefined	-
mpicxx	OpenMPI/3.0.0-GCC-7.2.0-2.29	Error __GKfree	-
mpicxx	OpenMPI/3.1.1-iccifort-2018.3.222-GCC-7.3.0-2.30	Error __GKfree is undefined	-
mpic++	OpenMPI/3.1.1-iccifort-2018.3.222-GCC-7.3.0-2.30	Error __GKfree is undefined	-
mpic++	OpenMPI/3.0.0-GCC-7.2.0-2.29	Error __GKfree	-
mpic++	OpenMPI/2.0.2-iccifort-2017.1.132-GCC-6.3.0-2.27-opa	Error __GKfree is undefined	-
mpic++	OpenMPI/2.0.2-GCC-6.3.0-2.27-opa	Error __GKfree	-
mpiicpc	intel/2018b	-	Error __GKfree is undefined
mpiicpc	intel/2019a	-	Error __GKfree is undefined
mpiicpc	intel/2018a	-	Error __GKfree is undefined

Table 15: Summary table for the failing steps.

A.2 Steps to install LibTopoMap

1. Download LibTopoMap package:
   wget https://htor.inf.ethz.ch/research/mpitopo/libtopomap/libtopomap-0.9.tgz
2. Unpack the package:
   tar -xvf libtopomap-0.9.tgz
3. Create a directory for ParMETIS inside libtopomap-0.9:
   mkdir MPIParMETIS
4. Change to that directory:
   cd MPIParMETIS
5. Get ParMETIS v3.1.1:
   wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/OLD/ParMetis-3.1.1.tar.gz
6. Unpack ParMETIS:
   tar -xvf ParMetis-3.1.1.tar.gz
7. Get METIS v4.0.1:
   wget http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/OLD/metis-4.0.1.tar.gz

8. Unpack METIS:
   tar -xvf metis-4.0.1.tar.gz
9. Load the C++ compiler in order to compile ParMETIS
10. Change the directory to ParMETIS:
    cd ParMetis-3.1.1
11. Compile ParMETIS:
    make
12. Change directory to METIS:
    cd metis-4.0
13. Open the file proto.h:
    vi Lib/proto.h
14. Add the diff lines from `https://htor.inf.ethz.ch/research/mpitopo/libtopomap/metis_4.0-extern_c-patch.diff` to this file. Specifically when there is + sign in front, those lines should be added.
15. Change the directory to LibTopoMap directory
16. Compile LibTopoMap:
    make
17. Refer to `https://htor.inf.ethz.ch/research/mpitopo/libtopomap/` for testing the library.

A.3 Network topologies configuration files

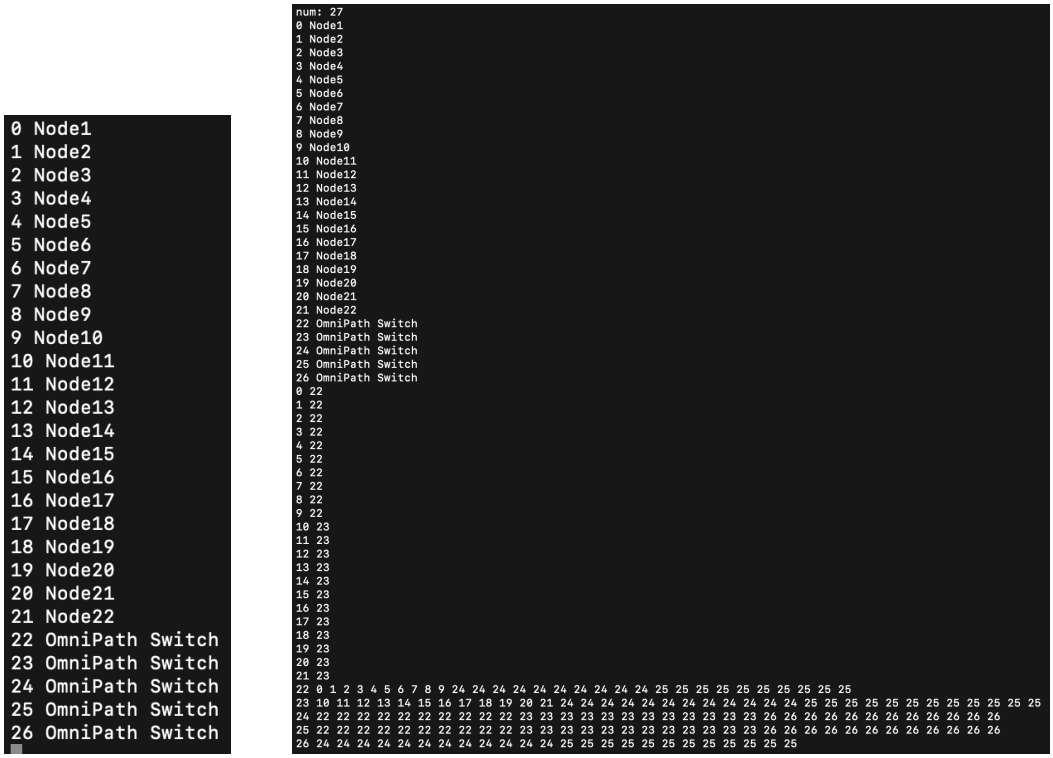
```
0 Node1
1 Node2
2 Node3
3 Node4
4 Node5
5 Node6
6 Node7
7 Node8
8 Node9
9 Node10
10 Node11
11 Node12
12 Node13
13 Node14
14 Node15
15 Node16
16 Node17
17 Node18
18 Node19
19 Node20
20 Node21
21 Node22
22 Node23
23 Node24
24 Node25
25 Node26
26 Node27
```

(a) Node naming file.

```
num:  27
0 Node1
1 Node2
2 Node3
3 Node4
4 Node5
5 Node6
6 Node7
7 Node8
8 Node9
9 Node10
10 Node11
11 Node12
12 Node13
13 Node14
14 Node15
15 Node16
16 Node17
17 Node18
18 Node19
19 Node20
20 Node21
21 Node22
22 Node23
23 Node24
24 Node25
25 Node26
26 Node27
0 9 18 3 6 1 2
1 10 19 4 7 2 0
2 11 20 5 8 0 1
3 12 21 6 0 4 5
4 13 22 7 1 5 3
5 14 23 8 2 3 4
6 15 24 0 3 7 8
7 16 25 1 4 8 6
8 17 26 2 5 6 7
9 18 0 12 15 10 11
10 19 1 13 16 11 9
11 20 2 14 17 9 10
12 21 3 15 9 13 14
13 22 4 16 10 14 12
14 23 5 17 11 12 13
15 24 6 9 12 16 17
16 25 7 10 13 17 15
17 26 8 11 14 15 16
18 0 9 21 24 19 20
19 1 10 22 25 20 18
20 2 11 23 26 18 19
21 3 12 24 18 22 23
22 4 13 25 19 23 21
23 5 14 26 20 21 22
24 6 15 18 21 25 26
25 7 16 19 22 26 24
26 8 17 20 23 24 25
```

(b) Node mapping file.

Figure 8: Configuration file for 3x3x3 (3D) Torus network topology.



(a) Node naming file.

(b) Node mapping file.

Figure 9: Configuration file for two-level fat tree network topology.

A.4 Script to convert communication matrices to graph communication pattern

Below you will find the script used to convert the communication matrices of the applications into graph communication patterns to be used for LibTopoMap.

```
import numpy as np
from scipy import sparse, io
import pandas as pd


comm_matrix = np.genfromtxt('input.csv', delimiter=',', dtype=np.int32)

sparse_matrix = sparse.csr_matrix(comm_matrix)
io.mmwrite('output', sparse_matrix)
```

UNIVERSITÄT BASEL

PHILOSOPHISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)

Bachelor's / Master's Thesis *(Please cross out what does not apply)*

Master Project

Title of Thesis *(Please print in capital letters)*:

Optimizing Parallel Processes to Nodes Mapping in Contemporary High Performance Interconnection Topologies

First Name, Surname *(Please print in capital letters):* FATJON LALA

Matriculation No.: 18-063-412

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged.

I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

In addition to this declaration, I am submitting a separate agreement regarding the publication of or public access to this work.

☐ Yes ☒ No

Place, Date: Basel, 31/12/2020

Signature: 

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .

UNI BASEL