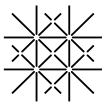


Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
High Performance Computing Group
hpc.dmi.unibas.ch

Viacheslav Sharunov
viacheslav.sharunov@stud.unibas.ch
15-059-322

Optimized parallel tasks to nodes mapping in 3-D high performance interconnection topologies

Master Thesis

Examiners:
Prof. Florina M. Ciorba
Prof. em. Helmar Burkhart

Supervisors:
Prof. Florina M. Ciorba
MSc. Ahmed Hamdy Mohamed Eleliemy
MSc. Ali Omar Abdelazim Mohammed

Basel, Switzerland
2017

Contents

Abstract

The constant increasing number of processors in high performance computing (HPC) systems and processes (parallel tasks that each application consists of) in parallel scientific applications leads to a challenge of efficient mapping between processes and processors. The way the parallel tasks are placed onto the computation units (processors) interconnected by a given network topology is of a paramount importance for the overall performance of the application. Approaching the Exascale computing era in HPC, the communication overheads and latencies play a significant role and require a thorough and accurate assessment, considering their impact on the application's execution time. An efficient mapping strategy should be able to reduce not only computation time for the given application, but even its energy consumption, and allow the application's tasks to make an efficient usage of the assigned resources of the computing system [8]. The present work is focused on an implementation of the different mapping strategies proposed in the literature and an experimental evaluation of the results on the highly adaptive energy-efficient computing platform (HAEC) - a research project of the Technical University of Dresden, Germany [14].

List of Algorithms

List of Tables

Introduction

Nowadays, steadily increasing interest in high performance applications can be observed in different fields of science: from study of molecular dynamics, calculation of physical equations, to simulations of behavior of black holes and providing experiments to obtain new chemical elements [1]. The constant demand in such applications and exponential growth in data they need lead to the development of massively parallel machines with hundreds of thousands and even millions cores on board — the top supercomputer in the list of TOP500 [2] China's "Sunway TaihuLight" has more than 10 million cores. The development and science are fast approaching the Exascale computing era, and one can claim that new Exascale supercomputers should have on board hundreds of millions of processing units to achieve the goal of Exascale — a peak performance above 1 Exaflop (10^{18} floating operations per second). The next-generation Exascale computer should not only lead to an improving in calculation speed, but also in data transmission efficiency.

The challenging issue for high performance computing (HPC) is still the energy consumption for computation and cooling [8]. It is becoming a significant concern, because a politico-economic pain threshold is suggested to be 25 megawatts (MW) as a working boundary, according to [9]. However, the HPC community anticipates that Exascale systems could consume over 100 MW, and just few of the existing computer centers can provide an adequate infrastructure. As a consequence, the challenge is turning into the question: How to exploit efficiently the available computing resources of the

given network topology and make the energy consumption economically and environmentally acceptable [8]? One practical method to solve the problem is to improve *data locality*, in other words, the way the data are placed onto available processors units (the CPUs). In fact, by doing so, the communication cost has a potential to be reduced. Therefore, the overall execution time of the application and its energy consumption would be decreased: *"Data movement across the system, through the memory hierarchy, and even for register-to-register operations will likely be the single principal contributor to power consumption. ... Since much of the power in an exascale system will be expended moving data, both locally between processors and memory as well as globality, the X-stack must provide mechanisms and APIs for expressing and managing data locality. These will also help minimize the latency of data accesses"* [9].

Placing the highly communicative pairs of application's tasks as close as possible reduces the communication time significantly (the number of hops each message must traverse between two ends is decreasing), and as a result, the total performance of the application is increasing. As most of the HPC applications and network topologies can be represented as (un)directed graphs, the problem of placing the parallel tasks becomes a mapping of application's communication graph onto the underlying network topology graph. Mapping of two graphs is an NP-hard [10]. The researchers propose various methods for topology mapping: greedy methods of placing most-heavily communicative vertices as close as possible [10], recursive graph bipartition [11], heuristically minimizing the total number of hop-bytes communicated and exploiting application's geometric information [12], [13].

To the best of our knowledge, there is not a sufficient number of works considering numerous algorithms for static mapping and assessing their pre-simulation, during simulation, and post-simulation performance. The creation of a list with the most efficient mapping algorithms is necessary in

order to arrive at an insight into the problem of using certain approaches in each special case — whether one is interested in the performance of the parallel application or in the energy efficiency of the application. The aims of this thesis are to provide an overview of different mapping techniques existing in the literature, to implement these algorithms, to experimentally evaluate their results (pre-simulation) and to validate them using a simulation framework for highly adaptive energy-efficient computing platforms — HAEC-SIM [14].

This work is divided into the following parts: in Chapter 1, the insights into the problem of mapping tasks or processes to the nodes (CPUs or processing element (PE)) of the target computing system are given. The definition and the background of tasks-to-nodes mapping found in the literature are also discussed in Chapter 1. The target architectures are detailed in this section as well. An overview of the selected mapping algorithms and their characteristics are represented in Chapter 2. The review of the commonly used performance metrics, as well as other proposed metrics in the literature, the test instances, and the experimental setup are provided in Chapter 3. The algorithmic performance with respect to the mapping quality assessed by the metrics and the execution time for the algorithms are provided in Chapter 4. The validation of the results with via the simulation using the HAEC-SIM framework of the best mapping strategies, obtained from Chapter 4, and their analysis are described in Chapter 5. The thesis is completed in Chapter 6 with conclusions and an outline of the future work.

Chapter 1

Background

The chapter gives an introduction into the topic of a tasks-to-nodes mapping problem, current limitations in the field, tasks and network topologies representation, as well as some details about HAEC Box.

1.1 Mapping Problem

Parallel applications consist of a number of tasks that exchange information and synchronize among them by using different communication patterns. Usually, such applications are designed in the way that they have to be run on parallel architectures. Hence, their interchange actions are arranged with certain communication schema, that tries to use efficiently the architecture of machine (cluster, high performance supercomputer, etc.), where the application is executed. The communication pattern of the given application can be represented as a virtual topology. Knowing that there is a variety of the network topologies (mesh, torus, fat-tree, hypercube, etc.), it is hard to find an ideal or even good match between the virtual and physical network topologies. The mapping problem can be seen as a minimization problem of some cost metric associated with the assignment processes (application's tasks) to nodes. In this thesis, the focus is on minimizing the following metrics: Inter-node physical communications (IeNPC) and Weighted Task Average

Distance (WTAD) (see Chapter 3). It turns out, that the mapping problem, i.e. the problem of deciding whether there exists such mapping M, that has a number of IeNPC or WTAD lesser than or equal to given x, is NP-hard [10]. Moreover, an exact polynomial time algorithm for the mapping problem is hardly to be created [19].

The poor mapping may lead to an inefficient usage of the resources of the underlying physical topologies, that causes large delays and latency (two communication tasks are placed far away to each other, so number of traversable hops is increasing), bandwidth bottlenecks (most communicative pairs of tasks use the same links), and as well big number of messages congestion and dilation [22]. As a result, the applications are losing their performance just due to the fact, that communication pairs are placed in a poor way. Hence, the mapping problem is a vital problem in HPC world, and the manner the tasks are mapped onto allocated processors has a remarkable impact. The careful mapping allows the tasks more efficiently to use the assigned resources.

1.2 Definitions

Both, the tasks of the application and HPC platform architecture can be represented as undirected weighted graphs.

Application Tasks Graph The parallel application can be represented as an undirected graph $G_t =< V_t, E_t >$. The tasks of the application are considered as separate processes. The vertices in V_t are tasks (processes) of the high performance application and edges in E_t represent communications between the tasks. Each vertex $v_t \in V_t$ may have a weight w_t, that is an execution cost associated with the vertex on a given processor [23]. However, in this thesis only homogeneous applications and topologies are considered, in other words, the amount of computation for each task on each node (CPU

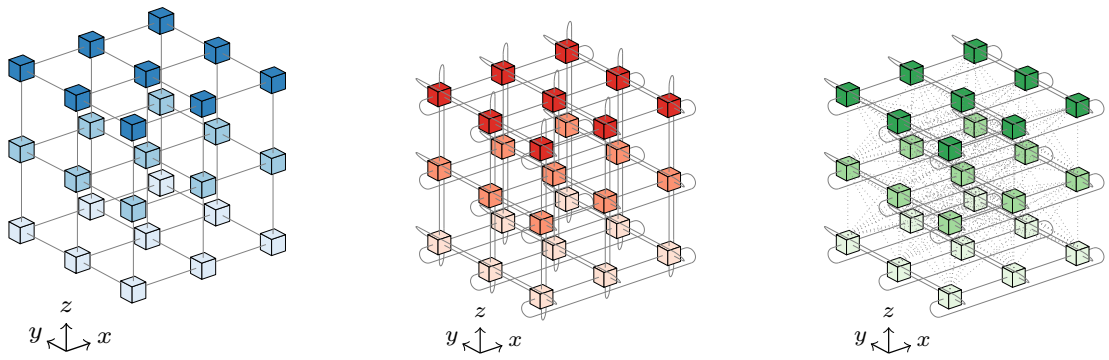
or PE) is the same. Similarly, each edge $e_{ij} = (v_i, v_j) \in E_t$ has a weight c_{ij}, that represents either the amount of communication in bytes between vertices v_i and v_j, or a number of communication messages [19], [20]. The present work considers the latter as the weights of the edges.

Processing Elements Topology Graph The network topology can be modeled as an undirected graph $G_p = < V_p, E_p >$, on $p = |V_p|$ vertices. Each of the V_p vertices represents a processor. The edges $E_p \subseteq V_p \times V_p$ represent the links in the network architecture. If the capacities of the connecting links between the processors are not equal, i.e. one considers a topology with heterogeneous links, the graph can be defined as $G_p = < V_p, \omega_p >$, where $\omega_p(i, j)$ defines the link capacity between any two processors $i, j \in V_p$ and $\omega_p(i, j) = 0$ in case of absent such link [8],[10].

Task Mapping In the present work, the emphasis lies on the static point-to-point mapping. It means that only one task can be processed by one of the allocated nodes (CPU or PE), and the mapping is calculated before any execution of the application. In this case, two graphs must have the same number of vertices. Two graphs are said to be isomorphic to each other if there is a one-to-one correspondence between their vertices and between their edges such that the incidence relationships are preserved [18]. Hence, the mapping problem according to [19] is equivalent to the graph isomorphism problem. Therefore, using the definition of the graph isomorphism and its notions, the tasks mapping can be specified as a mapping function $M : V_t \rightarrow V_p$, which maps the tasks (processes) of V_t onto the nodes (processors) of V_p. If the task $v_t \in V_t$ is mapped onto the node $v_p \in V_p$, one could say that the relation $M(v_t) = v_p$ is defined. The objective of the mapping function is to reduce the IeNPC and WTAD values between two communication tasks.

1.3 3-D Interconnection Topologies

Once the mapping problem and the mapping function are defined, one can consider the hardware topologies, that must run our high performance applications. For the different mapping algorithms (see in Chapter 2) three physical topologies were chosen: a three dimensional mesh (k-ary 3-mesh), a three dimensional torus (k-ary 3-cube) and the HAEC topology. The mesh and torus are mostly common in HPC systems (in June 2017 seven of the top ten supercomputers in the Top500 list have torus network or its variation), whereby the HAEC topology is unique to the HAEC Box [14]. Below in Figure 1.1 one can observe a graphical representation of each of the topologies. One must note, that $3 \times 3 \times 3$ version instead of regular $4 \times 4 \times 4$ version of the HAEC Box is illustrated for better visibility. In this thesis, the three topologies of size of power of two in each dimension are considered, namely, $4 \times 4 \times 4$, $8 \times 8 \times 4$, $16 \times 8 \times 8$, and $16 \times 16 \times 16$.



(a) 3-D Mesh (b) 3-D Torus (c) HAEC Box

Figure 1.1: Graphical representation of the $3 \times 3 \times 3$ topologies: (a) 3-ary 3-mesh (mesh); (b) 3-ary 3-cube (torus); (c) HAEC Box.

1.3.1 HAEC Box

The HAEC (Highly Adaptive Energy-efficient Computing) Box (Figure 1.1(c)) is a novel architectural concept of the Technical University of Dresden, Germany[1], that utilizes innovative ideas of optical and wireless chip-to-chip communication [14]. The HAEC Box belongs to high performance-low energy parallel computing platforms. The topology is designed to be 3-D with 64 processing elements organized as $4 \times 4 \times 4$. A group of 16 nodes (4×4) is placed onto a single 2-D board and connected using optical links into a 2-D torus (solid lines in Figure 1.1(c)) with a transmision bandwidth of 250 Gbit/s. Processing elements of neighboring boards can all communicate with each others via wireless links (dotted lines in Figure 1.1(c)) with transmission bandwidth of 100 Gbit/s. The wireless links represent a fully connected topology, i.e. each of the processing elements communicate with all of his 16 neighbors placed on the adjacent board [14], [17]. Due to the fact that the actual hardware development is still being ongoing, the authors present the event trace-based simulator HAEC-SIM [14]. Running energy-aware applications, the simulator allows the deep exploration of the conjectural performance and energy costs of the HAEC Box.

[1] https://tu-dresden.de/sfb912

Chapter 2

Tasks-to-Nodes Mapping Strategies

The task mapping problem is known as an NP-hard [10], but still considered as an interesting and important problem. Several approaches to solve the problem have been proposed in the literature [8]. They can be classified according to [20] into three categories: graph-theoretic, mathematical programming, and heuristic methods. *Graph-theoretic* category represents application's tasks as an undirected weighted graph and uses minimal-cut algorithm to retrieve a task assignment with minimum interprocessor communication. In the second category, *mathematical programming*, the main problem is how to assign a certain number of facilities to a certain number of locations with the minimum cost [22]. This optimization problem is solved using mathematical programming techniques. And the last, not the least group is heuristic methods that frequently provide fast but suboptimal solutions because of the fact, that usually it is hard to find an optimal solution in rational time due to a space complexity of the problem [23]. The general classification of the strategies by Kafil et al. [23] is given in Fig. 2.1.

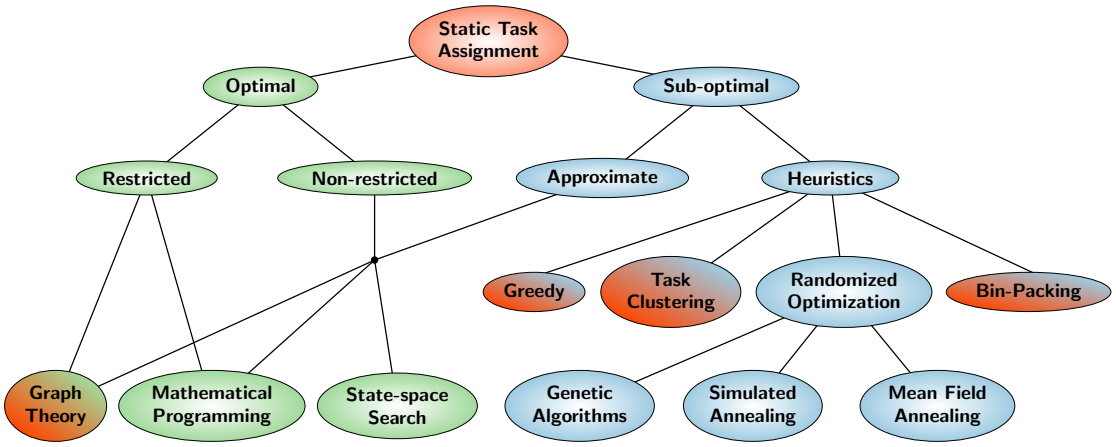


Figure 2.1: Classification of mapping strategies. Algorithms from the classes highlighted using color gradient are considered in the chapter.

All static tasks-to-nodes algorithms can be classified into two large groups: *optimal* and *sub-optimal* strategies according [23]. As one can observe, optimal mapping strategies are further divided to *restricted* and *non-restricted*, where the first ones yield optimal solutions in polynomial time, and the second ones return optimal solutions, but very often, not in polynomial time. The sub-optimal strategies in turn consist of two classes: *approximate* and *heuristics*. The approximate algorithms may guarantee a solution that is within a certain range from the optimal solution [23]. Heuristics are designed for finding any solution (it may not be necessary optimal) more quickly, when classic methods are too slow or fail in solving of the problem. Most of the research efforts are put on the development of heuristic algorithms, as one is visually demonstrated on Fig. 2.1 (three groups out of four, considered in this thesis, are heuristic algorithms). This fact can be explained due to the intractable and NP-hard nature of the problem, where everyone tries to find any solution for the particular mapping problem.

In this thesis, the additional mapping algorithms are studied as a complement to already described in the work [24]. The full list of the algorithms can be found on page 6. They are all considered in terms of the

four mapping strategies, represented in Fig. 2.1: Graph theory (Recursive Bipartition Mapping algorithm [11]), Task clustering (PaCMap: Topology Mapping [26]), Bin-Packing (Topology-aware Task Mapping [12]), Greedy algorithms (Generic Topology Mapping [10], GreedyAllC Mapping [27], Fast and High Quality Greedy Mapping [13], Utilization-based depth-first Mapping [28]).

Task clustering algorithms try to put the groups of most communicative tasks together. In turn of Bin-Packing, the algorithms use a policy of tasks ordering and a policy of a placement for the tasks. All greedy algorithms assign one task to one node (CPU or PE) at each step, trying to greedily map task's heaviest neighboring to the neighboring of allocated node, until a complete assignment is reached.

In the following sections all algorithms will be considered in details. At first, two already described algorithms in [24] will be explained: Bokhari's Pairwise Interchange and Minimum Manhattan Distance. Then, additional to them algorithms come to the turn.

2.1 A Pairwise Interchange Algorithm: Mapper

Shahid Bokhari in [19] proposed a heuristic algorithm called *Mapper*. The algorithm receives as input a communication (adjacency) matrix of the application graph and returns a permutation matrix that matches as close as possible an adjacency matrix of the topology (the considered topology in [19] is a finite element machine). The algorithm uses a pairwise interchanges of the elements of the input matrices with probabilistic jumps to improve the initial provided mapping and a cardinality of the given problem matrix. The good explanation of the algorithm in short can be found in [24], and the algorithm in [19]. The following listing (Algorithm 1) represents the algorithm.

Algorithm 1 A Pairwise Interchange: Mapper

Data: $G_p, G_t, M : V_t \to V_p$ (initial mapping)
Result: $B : V_t \to V_p$ (improved mapping)
$B \leftarrow M$;
$done \leftarrow false$;
while *not done* **do**
  repeat
    $improved \leftarrow false$;
    foreach $u \in V_t$ **do**
      1: examine pairwise exchange of u with every $v \in V_t \setminus \{u\}$;
      2: select pair $\langle u, v \rangle$ with largest *gain* in $|M|$;
      3: if largest $gain \geqslant 0$ then exchange pair $\langle u, v \rangle$;
      4: if largest $gain > 0$ then $improved \leftarrow true$;
    end
  until $improved = false$;
  if $|M| < |B|$ **then**
    $done \leftarrow true$;
  else
    $B \leftarrow M$;
    randomly interchange n pairs in M;
  end
end

2.2 Minimum Manhattan Distance Algorithm

This algorithm, proposed in [24], exploits the idea of minimizing a Manhattan distance [3] to map applications tasks. The goal of the strategy is to place the most communicative pairs as close as possible with respect to the hops between allocated nodes. As a first step a list of all communicative pairs is created, that follows a priority policy — the most communicative ones (total amount in bytes of exchanged messages or total amount of sent messages) are in the top of the list. Then, the first communicative pair is placed onto the center of the network topology with a node distance of 1 hop. The next pairs are either mapped to a free node, that is nearest to an already assigned processors (with respect to the minimizing of Manhattan distance), if one of the processes in the pair has been already mapped, or as close as possible to the center of the topology. The listing of the algorithm is repre-

sented in Algorithm 2. The function NEARESTNEIGHBOR is implemented as a breadth-first search algorithm. It finds a next free node to the given node-parameter.

Algorithm 2 Minimum Manhattan Distance

Data: E_t (decreasing ordered by weight), G_p
Result: $M : V_t \to V_p$
$M \leftarrow \{\,\}$;
start $\leftarrow (\lfloor d_x/2 \rfloor, \lfloor d_y/2 \rfloor, \lfloor d_z/2 \rfloor)$;
foreach $(u, v) \in E_t$ **do**
  switch (u, v) **do**
    case $u \in M, v \notin M$ **do**
      M(v) $\leftarrow$ NEARESTNEIGHBOR(u);
    case $u \notin M, v \in M$ **do**
      M(u) $\leftarrow$ NEARESTNEIGHBOR(v);
    case $u \notin M, v \notin M$ **do**
      M(u) $\leftarrow$ NEARESTNEIGHBOR(start);
      M(v) $\leftarrow$ NEARESTNEIGHBOR(u);
  end
end

2.3 Recursive Bipartitioning Algorithm

In computer science graph partitioning is a common technique. Graph partitioning is mostly used to partition the underlying graph model of computation and communication. In case of tasks-to-nodes mapping an application and topology can be represented as graphs. In the application graph the nodes denote objects to be computed, and the edges — a communication between these nodes. This graph needs to be partitioned such that there are few edges between the blocks (pieces). In particular, if one wants to use k processors the graph will be partitioned into k blocks of about equal size [29]. Then, it is possible to map the blocks of nodes (or groups of nodes) onto allocated processors of the network topology. The algorithm proposed in [11] by Jingjin Wu bipartitions both the communication graph and the physical topology graph recursively until the mapping $M : V_t \to V_p$ is found. For the

graph partition a set of serial programs for partitioning graphs "METIS" [5] is used in the experiments section of the present work. The cardinality of the application graph G_t is equal to the cardinality of the topology graph G_p.

Algorithm 3 Recursive Bipartitioning Mapping

```
bipartitioning_mapping(G_t, G_p)
if |G_p| == 1 then
 {
    M(G_t) = G_p
    return;
 }
/* bipartition both graphs G_p and G_t into subgraphs                    */
/* G_{i=1,2}, such that |G_1| = |G_2| for tasks and processors graphs     */
(G_1p, G_2p) ← graph_bipartition(G_p);
(G_1t, G_2t) ← graph_bipartition(G_t);
bipartitioning_mapping(G_1t,G_1p);
bipartitioning_mapping(G_2t,G_2p);
```

2.4 Topology Mapping of Unstructured Communication Patterns: PaCMap

The idea behind the PaCMap algorithm (partitioning and center mapping) is to simultaneously make a job allocation and task mapping in order to reduce overall communication overhead and time needed to execute HPC application [26]. The algorithm could be divided into two main parts: Center Machine Node selection and Center Task Group selection. As a preparation step, if it is needed, the communication (task) graph can be partitioned into k task groups, where k is equal to the number of available allocated nodes in the underlying network topology. After the preparation step each task group fits into a single node in the cluster. The authors of the algorithm propose to use the "METIS" library [5] for graph partitioning. Next, the PaCMap algorithm finds a center of each task group and maps it to the selected center node of the network topology. After, the expansion of the allocation takes place until all tasks are mapped. An interested reader can find the details of

the algorithm, namely how the center node and the center of each task group are selected, in [25], or deeper analysis and evaluation of the algorithm with mapping background in [26].

2.5 Topology-aware Task Mapping

The algorithm, proposed in [12], is an iterative algorithm, the main goal of which is a selection of a next process from the set of the application tasks to be placed on a next node from the set of available topology's nodes (CPU or PE). The way the next process and next node are chosen is guided by an *Estimation function*. The function makes an assessment of the placing of certain process on certain processor in the current cycle of the algorithmic loop. Hence, one can find the best processor, where the placing cost of the current chosen task is the least. However, for some tasks it is not necessary to be placed on the best processor. What matters is a criticality of the placing of the task in the current cycle of the loop. With help of the *Estimation function* one can find the measure of the criticality, and, therefore, one can select the most critical task and place it onto the best processor in the current cycle. The listing of the algorithm is given below in Algorithm 4. Here, V_t and V_p are the sets of the processes (tasks) in the application and allocated for the application physical processors (CPU or PE) of the network topology, respectively.

There are three estimation functions proposed by authors in [12]. For all of them it is needed to define the following parameters: T_k — the set of the tasks that remain to be placed, P_k — the set of available processors on k^{th} iteration. Respectively, $\overline{T_k}$ — the set of already placed tasks and $\overline{P_k}$ — the set of processors that have been already used for mapping tasks from $\overline{T_k}$.

The distance $d_p(\cdot\,,\cdot)$ between any two processors p_1 and p_2, needed to compute the value of the *Estimation function*, could be considered, for ex-

ample, as a Manhattan distance [3]. The number of messages sent between two processes is represented by a value of c_{ij} [19], [20].

Algorithm 4 Topology-aware Task Mapping

Data: V_t,V_p, $|V_t| = |V_p| = n$
Result: $M : V_t \mapsto V_p$
$T_1 \leftarrow V_t$; $P_1 \leftarrow V_p$
for $k \leftarrow 1$ **to** n **do**
  $max\_criticality \leftarrow -\infty$;
  for $task \in T_k$ **do**
    $criticality(t) = \frac{\sum_{p \in P_k} f_{est}(t,p)}{n-k} - \min_{p \in P_k} f_{est}(t,p)$;
    if $criticality(t) > max\_criticality$ **then**
      $t_k \leftarrow t$;
      $max\_criticality \leftarrow criticality(t)$;
    end
  end
  $min\_cost \leftarrow \infty$;
  for *processor* $p \in P_k$ **do**
    if $f_{est}(t_k, p) < min\_cost$ **then**
      $p_k \leftarrow p$;
      $min\_cost \leftarrow f_{est}(t_k, p)$
    end
  end
  $M(t_k) = p_k$;
  $T_{k+1} \leftarrow T_k - \{t_k\}$;
  $P_{k+1} \leftarrow P_k - \{p_k\}$;
end

2.5.1 First order approximation

The estimation function considers the situation, when the placement of some of the tasks is not yet known. Therefore, only the contribution of the communication with already assigned tasks could be taken into account:

$$f_{est}(t_i, p, M) = \sum_{t_j \in \overline{T_k}} c_{ij} \times d_p(p, M(t_j)) \tag{2.1}$$

This estimation function was used for conducting the experiments, described in Chapter 4.

2.5.2 Second order approximation

This type of estimation function calculates the contribution to communication with not yet assigned tasks. The function assumes that certain task $t_j \in T_k$ would be placed on a random processor. Hence, it approximates the distance between p and $M(t_j)$ by the expected distance of p to other processors:

$$d_p(p, M(t_j)) \approx \frac{\sum_{p_j \in V_p} d_p(p, p_j)}{|V_p|} \tag{2.2}$$

Thus, the estimation function takes the form:

$$f_{est}(t_i, p, M) = \sum_{t_j \in \overline{T_k}} c_{ij} \times d_p(p, M(t_j)) + \sum_{t_j \in T_k} c_{ij} \times \frac{\sum_{p_j \in V_p} d_p(p, p_j)}{|V_p|} \tag{2.3}$$

2.5.3 Third order approximation

The last approximation of the estimation function considers as well as in Section 2.5.2 the contribution to communication of not yet assigned tasks, but takes into account the fact, that they could be mapped only onto the processors that are available, i.e. that are still in the set P_k. The second order approximation does not take into consideration this constraint. In other words, the distance between any processor p and $M(t_j)$ can be represented as:

$$d_p(p, M(t_j)) \approx \frac{\sum_{p_j \in P_k} d_p(p, p_j)}{|P_k|} \tag{2.4}$$

2.6 Generic Topology Mapping Strategy

The algorithm proposed in [10] is similar to all other greedy algorithms. However, this version considers edge weights, and, as authors claim, could be applied to heterogeneous network topology. The greedy mapping starts

at some vertex $v_p \in G_p$, chooses the heaviest (the most communicative) vertex $v_t \in G_t$ and greedily maps its heaviest neighboring vertices in V_t to the neighboring vertices in G_p with heaviest connections. The process does it recursively until the moment where all tasks are mapped onto allocated processors. The listing of the presented algorithm is given in Algorithm 5.

Algorithm 5 Greedy Graph Embedding

```
Data: G_p, G_t, C(v) = 1  v ∈ V_t
Result:  M : V_t ↦ V_p
S ← V_t; Q ← priority queue; s ∈ V_p - start;
while  S ≠ ∅ do
    find vertex m ∈ S with heaviest out − edges;
    if C(s) = 0 then
        pick new s  ∈ V_p : C(s) ≥ 1;
    end
    M(m) = s; S = S \ m; C(s) = 0;
    foreach u|(m, u) ∈ E_t and u ∈ S do /* add neighbors of m ∈ S to Q      */
        Q ← (m, u)|u ∈ S;
    end
    while Q ≠ ∅ do
        (u, m) ← Q;
        if C(s) = 0 then
            find closest to s vertex t ∈ V_p :  C(t) = 1, using Dijkstra's algorithm
            s = t
        end
        M(m) = s;  S = S \ m;  C(s) = 0;
        foreach u|(m, u) ∈ E_t and u ∈ S do /* add neighbors of m ∈ S to Q   */
            Q ← (m, u)|u ∈ S;
        end
    end
end
```

2.7 GreedyALLC Mapping Strategy

The algorithms of the group *GreedyALL* proposed in the [27], are the descendants of the usual greedy algorithms, in particular, from Section 2.6. The authors require, that the network topology to be homogeneous, i.e. one processor of it can process only one task of the application and $|G_p| = |G_t|$. As a prerequisite for the algorithm one needs to calculate $t(\cdot\,,\cdot)$ — a value

of time that a sending of one unit of information takes through the edges $e_1, \ldots, e_l$ of the topology along some routing path P in G_p:

$$t(P) = \sum_{i=1}^{l} \frac{1}{\omega_p(e_i)} \tag{2.5}$$

$$t(u_p, v_p) = min(\ t(P) \mid P\ connects\ processes\ u_p\ and\ v_p) \tag{2.6}$$

Due to the case that one should consider a homogeneous topology, as the authors require, all connections have the same weights, for instance $\omega_p(e_i) = 1$. The listing of the algorithm is given in Algorithm 6.

Algorithm 6 The GreedyALLC Mapping

Data: G_t, G_p, with $|V_t| = |V_p|$
Result: Pairs (v_t^i, v_p^i), such that $M : V_t \mapsto V_p$ defined by $M(v_t^i) = v_p^i$
Find $v_t^0 \in V_t$ with maximal $\sum_{e=\{v_t^0, v_t\} \in E_t} \omega(e)$; $v_p^0 \in V_p$ with minimal $\sum_{u_p \in V_p} t(u_p, v_p)$
Create vectors $sum_t = \vec{0}$ and $sum_p = \vec{1}$ of length $|V_t|$
for $i \leftarrow 0$ **to** $|V_c| - 1$ **do**
  $sum_t(v_t^i) \leftarrow -1$ `// Mark as assigned`
  $sum_p(v_p^i) \leftarrow INT\_MAX$ `// Mark as assigned`
  forall $e_t = \{v_t^i, \omega\} \in E_t$ **do**
    if $sum_t[\omega] \geq 0$ **then** // *ω is not yet assigned*
      $sum_t[\omega] \leftarrow sum_t[\omega] + \omega_t(e_t)$
    end
  end
  Pick v_t^{i+1} such that $sum_t(v_t^{i+1})$ is maximal
  for $j \leftarrow 0$ **to** $|V_p| - 1$ **do**
    if $sum_p[j] < INT\_MAX$ **then** // *j is not yet assigned*
      $sum_p[j] \leftarrow 0$
      forall $e_t = \{v_t^{i+1}, \omega\} \in E_t$ **do**
        if $sum_t[\omega] < 0$ **then** // *ω has already been assigned*
          $sum_p[j] \leftarrow sum_p[j] + \omega_t(e_t) * t(j, \Gamma(\omega))$
        end
      end
    end
  end
  Pick v_p^{i+1} such that $sum_p(v_p^{i+1})$ is maximal
end

"The Greedy algorithm does not link the choices of vertex v_t^i and vertex v_p^i. The algorithm aims at a high communication volume of v_t^i with all or one of the already mapped vertices of G_t and a high centrality of v_p^i w.r.t. all or one of the already mapped vertices of G_p. The actual increase of communication times caused by the new pair (v_t^i, v_p^i) is not considered" [27]. Especially, the choice of v_p^i strongly depends on the choice of v_t^i. GreedyALLC takes this increase of communication into account.

2.8 Fast and High Quality Greedy Mapping

The algorithm proposed by Mehmet Deveci et al. in [13] tries to minimize the number of hops taken by each packet (the *weighted hop*). An interested reader can find an explanation of the metric in the paper mentioned before. As a prerequisite to the algorithm one must stress that it considers a task graph G_t as symmetric, in other words, a distance (number of hops) between some task v_i and v_j is the same regardless of direction. The algorithm starts with mapping the most-communicative (with the maximum send-receive communication volume) task to a randomly chosen node of the topology. The algorithm stores the total connectivity of each task, that is a neighbor to the already mapped ones, in the heap and updates this heap every time when a new task is mapped. Until all tasks are mapped, the algorithm pops a head of the heap (the most communicative task) — t_{best}, and looks for a best node from the list of the available ones for the task, using `GetBestNode` function. It returns either one of the farthest allocated nodes, in case of an absent connections between t_{best} and the mapped tasks, or the nearest one from the nodes to whom one of the neighbors of t_{best} is mapped, that can be found with a breadth-first search (BFS) on the graph G_p. In the algorithm a value N_{BFS} represents the number of initially mapped vertices. As authors claim, the large N_{BFS} considers the loosely connected components of the application

graph G_t to the nodes that are located far away from each other. But, these proposal could not work for the task graphs with a very low diameter. The listing of the algorithm is presented in Algorithm 7.

Algorithm 7 Fast and High Quality Greedy Mapping

Data: $G_t = (V_t, E_t)$, $G_p = (V_p, E_p)$, with $|V_t| = |V_p|$,
N_{BFS}: # vertices to be initially mapped
Result: $M : G_t \rightarrow G_p$
Initialize the max-heap: $conn_t \leftarrow 0 \ \forall \ t \in V_t$
▶ Find the most communicative task (MCT):
$t_0 \leftarrow t_{MCT}$
▶ Map t_0 to an arbitrary node:
$M(t_0) \leftarrow m_0$
▶ Update connectivity for the tasks in $nghbor(t_0)$:
forall $t_n \in nghbor(t_0)$ **do**
  $conn.update(t_n, c(t_0, t_n))$
end
while $\exists$ *an unmapped* t **do**
  if *number of mapped tasks* $< N_{BFS}$ **then**
    $t_{best} \leftarrow$ the farthest unmapped task found by BFS
  else
    $t_{best} \leftarrow conn.pop()$ ▶ the one with maximum connectivity
  end
  $m_{best} \leftarrow \texttt{GetBestNode}(t_{best}, G_p, G_t, M, conn)$
  $M(t_{best}) \leftarrow m_{best}$
  forall $t_n \in nghbor(t_best)$ **do**
    $conn.update(t_n, c(t_{best}, t_n))$
  end
end

2.9 Utilization-based Depth-first Algorithm

The efficient mapping should place adjacent tasks from the application graph G_t as close as possible to each other on the processors of the topology. In this way, when tasks communicate, the messages can traverse a lower number of hops to reach their destination. The key aspect of the algorithm proposed in [28] is the order in which the task graph G_t is traversed. Most utilized edges are traversed firstly to reduce interconnection between tasks.

The algorithm is recursive and greedy. First, it maps the most communicative task to the central node in the topology. Then, the `map_next` function searches among all outgoing edges of the task, mapped in the previous step, one with the biggest weight (most utilizable edge) and places its another task-end onto the next free processor. This process is repeated until there are no outgoing edges between already mapped and remaining tasks. The variable p shows which processor is currently chosen to being allocated for the next task.

The algorithm has been slightly modified to satisfy the prerequisite of $1:1$ mapping this thesis considers. Initial algorithm in [28] was set up to *many to one* mapping, i.e. one physical processor could host more than one application task. The new version not only takes into account $1:1$ mapping, but as well chooses as a start node the most central one, and as a next free — the closest node to whom where one of the neighbors of the chosen task has been mapped. The modified algorithm is presented in the listing below (Algorithm 8).

Algorithm 8 Utilization-based Depth-first Mapping Modification

Data: V_t, V_p, $|V_t| = |V_p| = n$
Result: $M : V_t \mapsto V_p$
`function` map_next(i, p)
while $\exists\ e_{i,j}$ *with* t_j *unmapped* **do**
    $k \leftarrow argmax_j(\omega(e_{i,j}))$ `// find the task connected by the \heaviest" edge`
    $node \leftarrow$ the closest node to $M(i)$
    $M(k) = node$
    $V_t = V_t \setminus k$
    $V_p = V_p \setminus node$
    $p \leftarrow$ map_next$(k, node)$
end
`return` p
`function` map()
    $M(start\_task) = start\_node$
    map_next$(start_task, start_node)$
`return` M

2.10 A* Algorithm

The well-known A^* algorithm from the area of artificial intelligence could be considered in additional to already described algorithms. The informed-search algorithm guarantees an optimal solution, but could not work with the problems of big sizes due to its high time and space complexity. The algorithm is used as a tree search algorithm. It means, that the root of tree (start node) is an initial (null) solution of the problem, intermediate nodes are the partial solutions, and the leafs represent the goal. With each node the cost function f is associated. The nodes are ordered for search according to the costs in the way the node with the lowest cost is considered first. The value $f(n)$ of the node n is computed as a sum of two other functions $g(n)$ and $h(n)$, where first one is the cost of the path from the start node to n, and second one estimates the cost of the cheapest path from n to the goal. In the case of tasks-to-nodes mapping, $g(n)$ would be the number of the inter-node physical communications and should be represented as an accumulative sum; $h(n)$ has a value, which can be obtained if all the remaining unmapped communications would be mapped to the nodes using single hop distance between them. Thus, $h(n)$ is the number of the remaining unmapped communications multiplied by 1 hop.

Choosing the next node to consider, the algorithm minimizes the cost of communications and maximizes the number of mapped communications. It means that in each step the algorithm tries to map the maximum number of communications for the lowest cost.

Chapter 3

Proposed Evaluation Approach

In this section the test instances, the experimental setup and the way the mapping algorithms are evaluated will be specified.

3.1 Performance Metrics

In literature, to assess a quality of a mapping algorithm, two different cost metrics are used: *dilation* and *congestion*. Dilation represents the average length of the path taken by a message sent from some process u to process v:

$$dilation(uv) = \sum_{p \in P(M(u)M(v))} \Upsilon(M(u)M(v))(p) \cdot |p|, \tag{3.1}$$

where the function $\Upsilon(\cdot)$ represents the routing algorithm, and $P(uv)$ is the set of simple paths (each edge occurs only once). For each pair processes u and v, $\Upsilon(uv)$ is a probability distribution on $P(uv)$. In other words, if $p \in P(uv)$, then $\Upsilon(uv)(p)$ is the fraction of traffic from u to v that is routed through path p [10].

The congestion of a link e connecting u and v is the ration between the amount of traffic on that link and the capacity of the link:

$$Traffic(e) = \sum_{u,v \in V_t} \omega(uv) \left(\sum_{p \in P(M(u)M(v)), e \in p} \Upsilon(M(u)M(v))(p) \right); \quad (3.2)$$

$$congestion(e) = \frac{Traffic(e)}{c(e)}, \quad (3.3)$$

where $\omega(uv)$ is the amount of communication between u and v, and $c(e)$ is the capacity (bandwidth) of the link e [10].

In the thesis the mapping algorithms are evaluated by using two groups of the metrics:

1. Logical metrics:
   (a) Inter-process logical communication (IePLC) defines the **total** number of exchanged messages between any two processes, is independent of the mapping strategy and a property of the task graph.
   (b) Intra-node logical communication (IaNLC) shows the **total** number of processes that are mapped onto the same physical node.
   (c) Inter-node logical communication (IeNLC) is the sum of **all exchanged** messages between nodes resulting from the mapping of the communication pairs but neglecting the network topology.
2. Physical metrics:
   (a) Inter-node physical communication (IeNPC) is the **total** number of messages two nodes exchange regarding the network topology and the applied routing protocol [17].
   (b) Weighted Task Average distance (WTAD) defines the weighted average distance between communicating application tasks [21].

These metrics are considered in the thesis, because they represent not only physical nature of the performance, when the application tasks are already

mapped onto the allocated processors, but as well the logical behavior of the tasks in the given parallel application.

Because of the fact, that one considers 1 : 1 mapping of processes and nodes in the present work, the values of the inter-node logical communications (IeNLC) are equal to the values of the inter-process logical communications (IePLC). The intra-node logical communications (IaNLC) are equal to zero for the same reason — there is no messages exchange **inside** the node. Thus, both values of IeNLC and IaNLC are not listed in this thesis.

The value of IeNPC (pre-simulation) can be calculated as:

$$IeNPC_{pre-sim} = \sum_{i}^{N} \#hops_i \times \#messages_i, \tag{3.4}$$

where N is the number of communication pairs in the application, $hops_i$ is the number of hops between the processors where the tasks of the i^{th} communication pair are mapped, and $messages_i$ — the overall number of exchanged messages between the tasks of the i^{th} communication pair.

It should be noted that the authors of [21] propose two methods to calculate the task average distance: normal one (TAD) and weighted one (WTAD). They claim that frequent communication can increase an utilization of the allocated resources, and the most talkative pairs of application tasks must be mapped as close as possible. However, the normal average distance (TAD) considers all pairs of communicating tasks. It could be excessive due to heterogeneity of the traffic between task pairs. For this reason, they suggest to use **WTAD** version of the metric, in which the communication matrix is used as an indicator of weights [21].

$$WTAD = \frac{1}{|V_t| \cdot |V_t| - 1} \sum_{a \in V_p} \sum_{b \in V_p} d(a, b) \cdot W_{\pi^{-1}(a), \pi^{-1}(b)}, \tag{3.5}$$

where $\pi^{-1}(a)$ returns the identifier of the task t running on processor a, and

matrix $W = [w_{i,j}]_{i,j \in V_t}$ represents the number of exchanged messages between tasks i and j, respectively;

$$TAD = \frac{1}{|V_t| \cdot |V_t| - 1} \sum_{a \in V_p} \sum_{b \in V_p} d(a,b) \cdot B_{\pi^{-1}(a),\pi^{-1}(b)}, \tag{3.6}$$

where matrix B is a boolean (0-1) version of communication matrix W and captures whether there is a communication between tasks (1) or not (0).

3.2 Experimental Setup

To be able to compare different mapping strategies, one need to run the pre-simulation experiments on real-world applications. The OTF2 [16] traces of the NAS Parallel Benchmark (NPB 3.3) [4] applications were chosen for this purpose, namely the traces of the pseudo applications BT — Block Tri-diagonal solver and LU — Lower-Upper Gauss-Seidel solver of classes C and D, and compiled with different number of MPI processes (64, 256, 1024, 4096). *"Discrete event traces capture the run time behavior of parallel applications on existing systems and form the basis of the simulation"*[14]. The traces of LU.C.64 and BT.C.64 were collected as performance relevant events using Score-P [15] measurement infrastructure during the execution of the desired applications on 4 KNL (Knights Landing) nodes of the cluster "miniHPC" of High Performance Computing department of Mathematics and Computer Science faculty at University of Basel. The details of the KNL CPU architecture are presented in Table 3.1. The commands used to create the traces can be found in Appendix A.

Initially, the traces of all problem sizes were recorded on the HPC system "Taurus" at Technical University of Dresden, Germany. However, the reason of the re-collection the applications traces with 64 processes on "miniHPC" cluster is the necessity to validate the experimental results in terms of *message-*

related statistics (see Section 5.3).

Model name	7210
Number of CPUs	64
CPU rate	1300 MHz
Threads per core	1
L1d cache	32K
L1i cache	32K
L2 cache	1024K
Main memory	96 GB
MCDRAM (mode)	16 GB (flat)
The cluster mode of the chip	all to all
Linux version	Centos7.XX
Hyper threading	not enabled

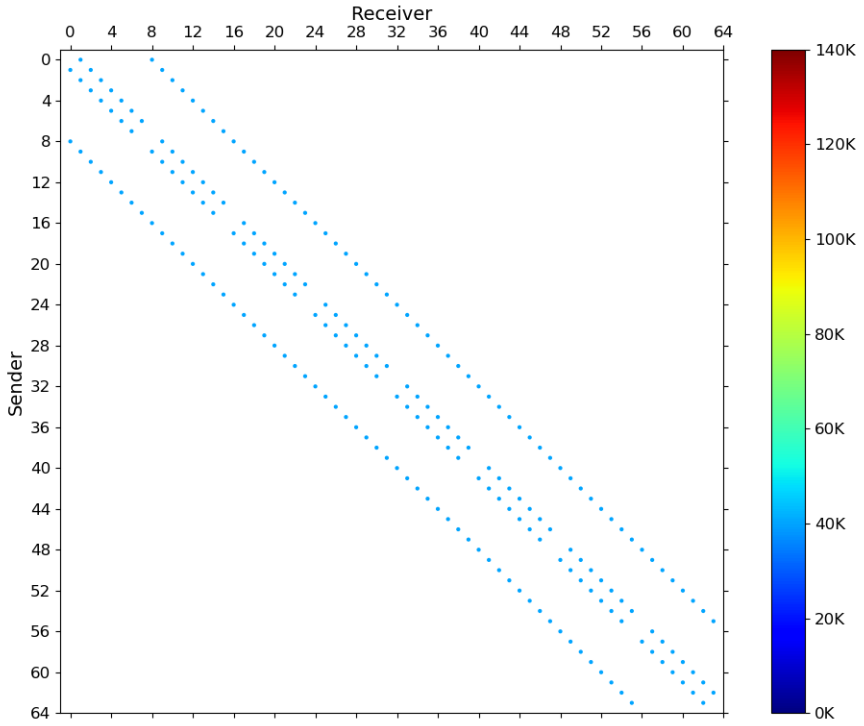
Table 3.1: The detailed information of a KNL node of the miniHPC cluster

3.3 Proposed Experiments

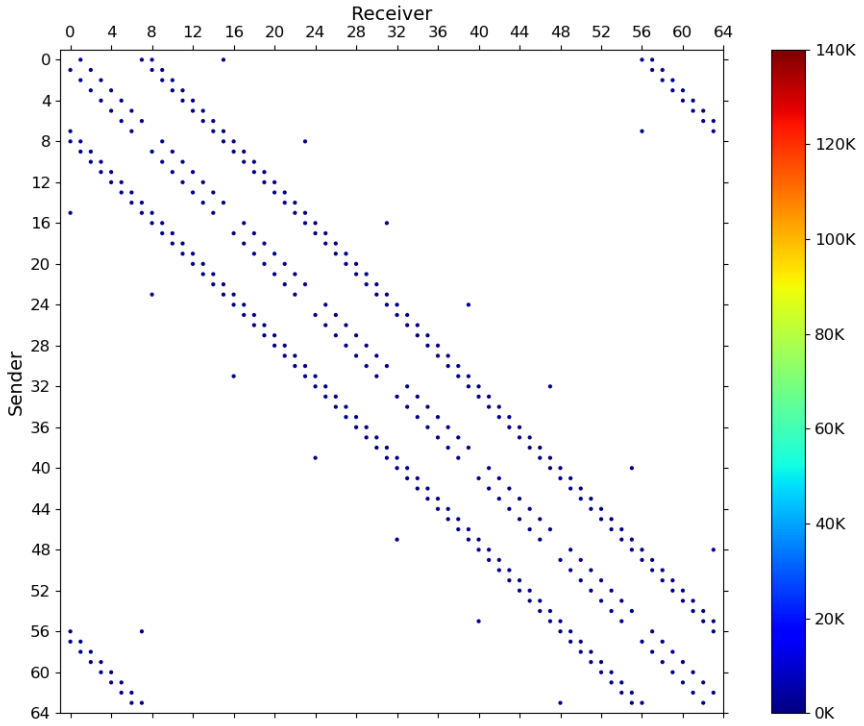
The experiments could be considered as a sequence of three main steps: pre-simulation, simulation and post-simulation. During the first step, all described algorithms are implemented in `Python`, the resulting mappings are evaluated with the physical performance metrics, considered earlier in Section 3.1, and the list of the "best" mappings is created. Then the simulation step comes in turn. The benchmark traces are mapped to mesh, torus, HAEC Box topologies with the "best" mappings, obtained on the previous step, with HAEC-SIM framework. Finally, the post-simulation step compares the impact of these "best" mappings for benchmark traces of the LU and BT applications (the values of IeNPC before the simulation and after are compared).

3.3.1 Implementation of the mapping algorithms and their setup for pre-simulation evaluation

All algorithms were implemented with `Python 3`. The graph representation of both application and network topologies is possible with help of



(a) Communication pattern of the point-to-point messages exchanged in LU.C.64



(b) Communication pattern of the point-to-point messages exchanged in BT.C.64

Figure 3.1: Graphical representation of the NAS benchmark applications LU and BT of class C compiled with 64 processes on 64 cores of 4 KNL nodes (16 cores on each). The heat map shows the amount of sent/received messages between processes. The sender/receiver of the y and x axes denote ranks of the sender processes and of the receiver processes, respectively

`NetworkX`. It is a `Python` language software package for the creation, manipulation, and study of the structure, and function of complex networks [6].

The extracted from the traces point-to-point communication matrices (see on Fig. 3.1) are taken as input for each of the mapping algorithms. Then the algorithms create graphs with data from the matrices (the intersection of a row and a column in the matrix represents the communicative pair of tasks (vertices in a graph) and an amount of sent messages, respectively (the weight of an edge between vertices)), and map them onto the selected and defined previously network topologies — 3-D Mesh, 3-D Torus, HAEC Box. The topologies as well could be easily represented as graphs: the vertices have an edge between them only if the respective nodes in the network topology have a connection link between them. The number of nodes in the topologies is equal to the number of MPI processes, used for the compilation of the benchmark applications.

Each of the implemented mapping function returns a dictionary (`Python`'s data format of the following form: "$element : value$"), that consists of the list of the IDs of the application tasks and IDs of the processors, where the tasks were mapped. An example of such output can be proposed: {1:0, 2:1, 3:4, 2:3}. It means that a task with $ID = 1$ has been placed on a processor with $ID = 0$, a task with $ID = 2$ — on a processor with $ID = 1$, and so on. Then the output, created by `Python`'s script, are the mapping file, needed for the HAEC-SIM simulations, the statistic of the mapping — the calculated values of the performance metrics, described in Section 3.1, and the execution time of the mapping algorithm.

The mapping file, as it has been mentioned earlier, contains a map of the tasks IDs (they are could be considered as MPI ranks or OpenMP threads) to the processors of the network topology. It should be noticed, that the dictionary, returned by each of the mapping functions, must be slightly modified to be written in the mapping file. The dictionary is first reversed, namely

the elements and values are becoming now values and elements, respectively (they are just swapped). Then, the IDs of the processors are changing in the form of xyz representation, i.e. for the $4 \times 4 \times 4$ topology the processor with $ID = 0$ becomes the processor with $0, 0, 0$ coordinates, the processor with $ID = 1$ gets $1, 0, 0$ coordinates, the processor with $ID = 63$ becomes the processor with $3, 3, 3$ xyz coordinates. The format of such mapping file for a $4 \times 4 \times 4$ topology is represented in the listing below.

```
<mapping name>
x_coord y_coord z_coord number_of_processes process_id(s)
0 0 0 1 3
1 0 0 1 11
[...]
3 3 3 1 45
```

The first row shows the name of the used mapping strategy to generate the mapping file. Starting from the third row, each line defines the fact, that the computing nodes with coordinates (first three number in xyz notion) are allocated for one application task (its ID is in the last column). In other words, the task with $ID = 3$ is mapped to the node with $0, 0, 0$ coordinates. The results of the pre-simulation step for all mapping algorithms are discussed in more details in Chapter 4.

3.3.2 Simulation setup using the HAEC-SIM framework

During this step the properties of the application's events, captured by the OTF2 [16] traces, are modified. Thus, the output of the simulations are the event traces that describe the predicted application behavior. For providing the simulations on the HAEC-SIM platform, one needs to have three input files: an input trace file, a configuration file and a mapping (positions) file.

The input trace files have been created (see Section 3.2) and can be used for the simulation. The mapping files are results of the running `Python`'s script during the pre-simulation step, described in the previous subsection.

One thing left for the successful simulation is the configuration file. The file contains all parameters related to the simulated system, such as type of links of the topology, their bandwidth and latency, rate of the errors, the shape of the topology (mesh, torus, ...), type of routing and parameters of sent messages, etc. For the experiments *Dimension Order Routing* (DOR) communication model is chosen. Using DOR in a 3-D topologies, the packets are first routed in the x dimension, then in the y dimension, and lastly in the z dimension. All the results, presented in Chapter 5 are valid just for this communication model. It must be noted as well, that the $shortest - path\ routing\ protocol$ is used in the present work for routing the messages in 3-D Mesh and Torus. For HAEC Box — its own *haec_box* protocol. For more details, see [7], [14], and [17]. The examples of such configuration files for 3 topologies considered in this thesis and the commands for running the simulations are given in Appendix A.

3.3.3 Assessment of the impact of mapping via post-simulation analysis

As a result of each simulation, one will get the simulation trace, the statistic of which is calculated, using HAEC-SIM module **trace_stats** to make an insight in the impact of the certain mapping, whether the results, that were obtained in the pre-simulation step, are still valid on the real network topologies. In other words, whether the assumptions about the good performance of several mappings before simulation hold after the simulation as well.

The module **trace_stats** gets as input the simulation trace and returns the statistics of this trace, such as total duration of the trace in picoseconds,

average number of hops traversed by each message size group, total number of exchanged messages in each of the message size group, etc. The illustration of the returned statistics file will be given in Section 5.2. The command used for the retrieving the statistics with this module are given in Appendix A.

The overall representation of the experiments discussed above is given in Figure 3.2.

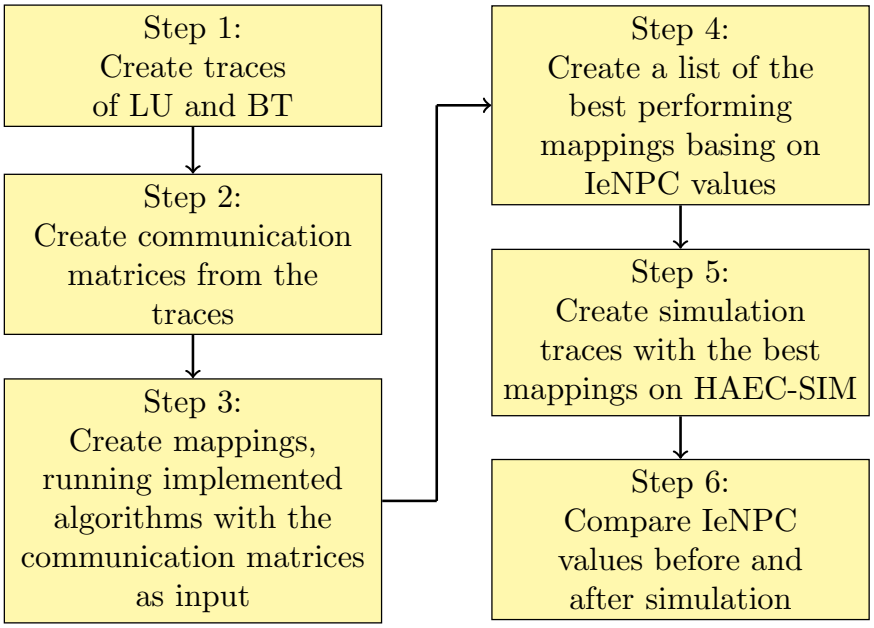


Figure 3.2: The workflow of the experiments

The following Tables 3.2—3.4 represent the design of experiments. The mark **x** in the cell means that the experiment (the action in the table) was conducted, the empty cell — the experiment was not conducted for the present thesis and is left for future work. The details of the experiments are given in Chapters 4—5.

The total amount of CPUs in the "miniHPC" cluster is 696 (22 Xeon nodes with 20 CPUs and 4 Xeon Phi nodes with 64 CPUs), and it is not sufficient to run the parallel applications with 1024 (and more) processes, it seemed wiser to not create traces for them on "miniHPC" and use for the experiments recorded on Taurus traces (to create communication matrices).

Action \ **Application**	LU.C.64	BT.C.64	LU.C.256	BT.C.256	LU.C.1024	BT.C.1024	LU.D.4096	BT.D.4096
Traces for the application on **miniHPC**	**x**	**x**	**x**	**x**				
Traces for the application on **Taurus**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
Communication matrices for the application	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**
Comparison on **mapping-related** statistics	**x**	**x**						
XYZ-trace for the application (with xyz mapping for an initial trace)	**x**	**x**						
Simulation traces with the best mappings using xyz-trace as an input	**x**	**x**						
Comparison on **message-related** statistics	**x**	**x**						

Table 3.2: Design of experiments: applications

First two rows of Table 3.2 are connected with Step 1 of the workflow of the experiments (see Figure 3.2), and represent **what** traces of **what** parallel applications **where** were recorded. Third row demonstrates from **what** traces of the applications the communication matrices were created (Step 2 of the workflow). The last four rows — the traces of **what** applications were used to perform HAEC-SIM simulations (Steps 5 and 6 of the workflow).

Table 3.3 shows **what** algorithms were implemented for the considered parallel applications (Step 3 of the workflow), and **which** of the created mappings were chosen for the experiments (Steps 4—6). Table 3.4 informs **what** models of the network topologies were used for the experiments.

Action \ Algorithm	Mapper	MinMD	Bipartition	PaCMap	TopoAware	Greedy	GreedyALLC	FHGreedy	UDFS	A^*
Mappings for **LU.C.64** and **BT.C.64**	x	x	x	x	x	x	x	x	x	
Mappings for **LU.C.256** and **BT.C.256**	x	x	x	x	x	x	x	x	x	
Mappings for **LU.C.1024** and **BT.C.1024**	x	x	x	x	x	x	x	x	x	
Mappings for **LU.D.4096** and **BT.D.4096**		x	x	x		x	x	x	x	
Comparison on **mapping-related** statistics for LU.C.64 and BT.C.64 (HAEC-SIM simulations)	x			x	x		x	x		
Comparison on **message-related** statistics for LU.C.64 and BT.C.64 (HAEC-SIM simulations)	x			x	x		x	x		

Table 3.3: Design of experiments: algorithms

Action \ Topology	3-D MESH				3-D TORUS				HAEC BOX			
Number of nodes	64	256	1024	4096	64	256	1024	4096	64	256	1024	4096
Topology model for **execution** algorithms in pre-simulation	x	x	x	x	x	x	x	x	x	x	x	x
Topology model for **validation** algorithms in HAEC-SIM simulator	x				x				x			

Table 3.4: Design of experiments: 3-D interconnection topologies

Chapter 4

Pre-simulation Experiments

4.1 Results

The results of the execution time of all above mentioned mapping algorithms for three main network topologies are given below. The execution time was measured by Python's library `cProfile`. Table 4.1 lists the execution time of generating the mapping for the LU (class C, 64 MPI processes) application of the NAS parallel benchmarks.

Algorithm	**Execution Time** [s]	**Execution Time** [s]	**Execution Time** [s]
LU.C.64	3-D Mesh	3-D Torus	HAEC Box
Mapper	1.964	4.418	6.214
MinMD	0.002	0.002	0.002
Bipartition	0.054	0.067	0.069
PaCMap	0.100	0.105	0.252
TopoAware	0.876	0.905	1.027
Greedy	0.038	0.059	0.187
GreedyALLC	0.064	0.067	0.098
FHGreedy	0.017	0.013	0.046
UDFS	0.013	0.022	0.022

Table 4.1: Time to generate the mapping (in sec) for LU.C.64 on $4 \times 4 \times 4$ topologies

Table 4.2 shows the execution time of generating the mapping for the BT (class C, 64 MPI processes) application of the NAS parallel benchmarks.

The statistics of the inter-node physical communications and weighted task average distances (IeNPC and WTAD, see Section 3.1) for the consid-

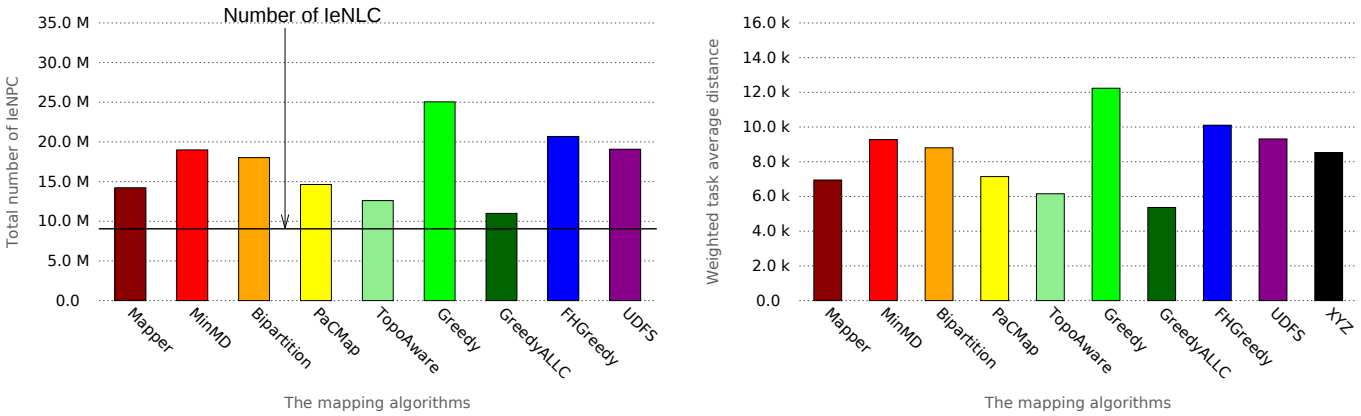
Algorithm	**Execution Time [s]**	**Execution Time [s]**	**Execution Time [s]**
BT.C.64	3-D Mesh	3-D Torus	HAEC Box
Mapper	18.098	6.347	20.500
MinMD	0.001	0.002	0.002
Bipartition	0.058	0.067	0.078
PaCMap	0.127	0.105	0.251
TopoAware	1.234	0.905	1.721
Greedy	0.049	0.059	0.211
GreedyALLC	0.070	0.067	0.130
FHGreedy	0.021	0.013	0.078
UDFS	0.012	0.022	0.023

Table 4.2: Time to generate the mapping (in sec) for BT.C.64 on $4 \times 4 \times 4$ topologies

ered algorithms with communication matrices of the applications LU.C.64 and BT.C.64 are shown in Figures 4.1 — 4.6. Each pair of graphical plots demonstrates the values of two metrics, used in this thesis to compare the quality of the proposed mapping algorithms. The horizontal black line on the left graphics represents the value of the inter-node logical communication (IeNLC), that is the sum of exchanged messages between nodes, neglecting the network topology. The distance between bars and this line should be minimized. The best mapping algorithms have the smallest difference between the value of IeNLC and values of IeNPC their bar graphs represent.

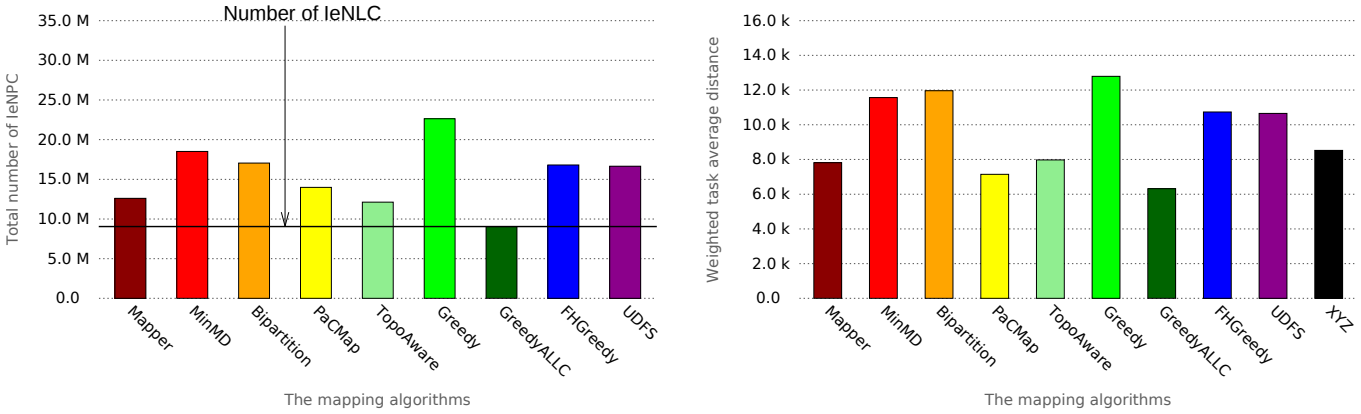
On the right plots the values of the weighted task average distance (WTAD) are visualized. Additionally, the values of the default consecutive mapping *"xyz"* are added, like the authors of [21] propose to compare the algorithmic performance. It means, that the tasks are mapped onto the nodes in the order of identifiers, starting an assignment of the task with $Id = 1$ to the node with $Id = 1$, and ending the assignment of the last task to the node with highest identifier. The lesser the value of WTAD is, the better performance the algorithm, showed this value, demonstrates.

The corresponding numerical values can be found in Appendix B. All measurements were conducted on Intel Core $i5-2410$M CPU at 2.30GHz$\times$4, with 12 GB of RAM, running Ubuntu 14.04.



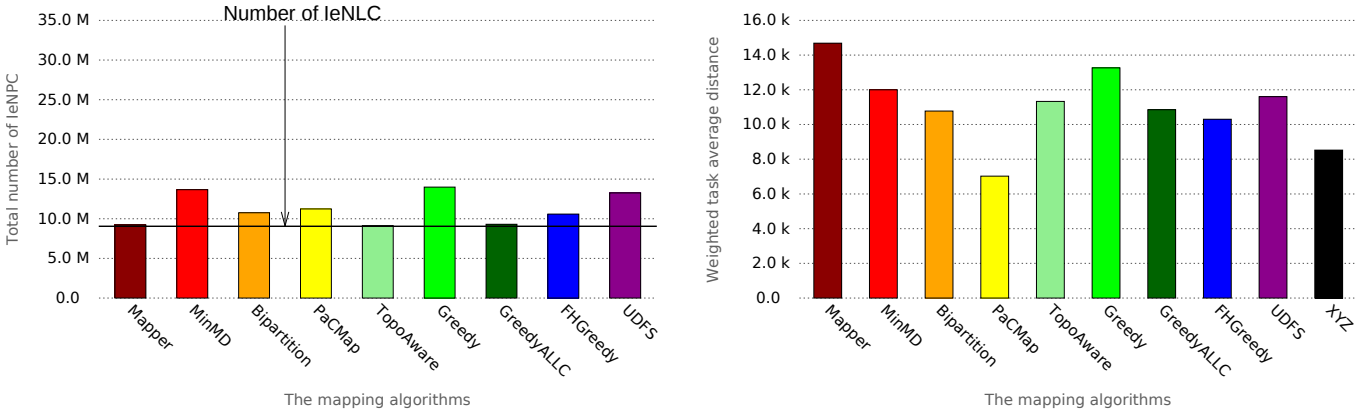
(a) Number of IeNPC (b) WTAD

Figure 4.1: Statistics for LU.C.64 on $4 \times 4 \times 4$ 3-D Mesh



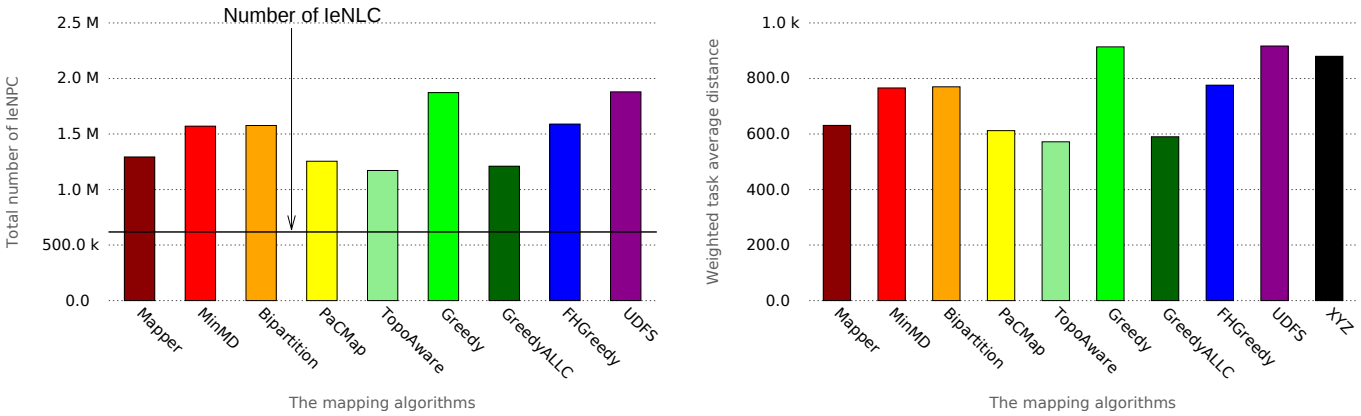
(a) Number of IeNPC (b) WTAD

Figure 4.2: Statistics for LU.C.64 on $4 \times 4 \times 4$ 3-D Torus



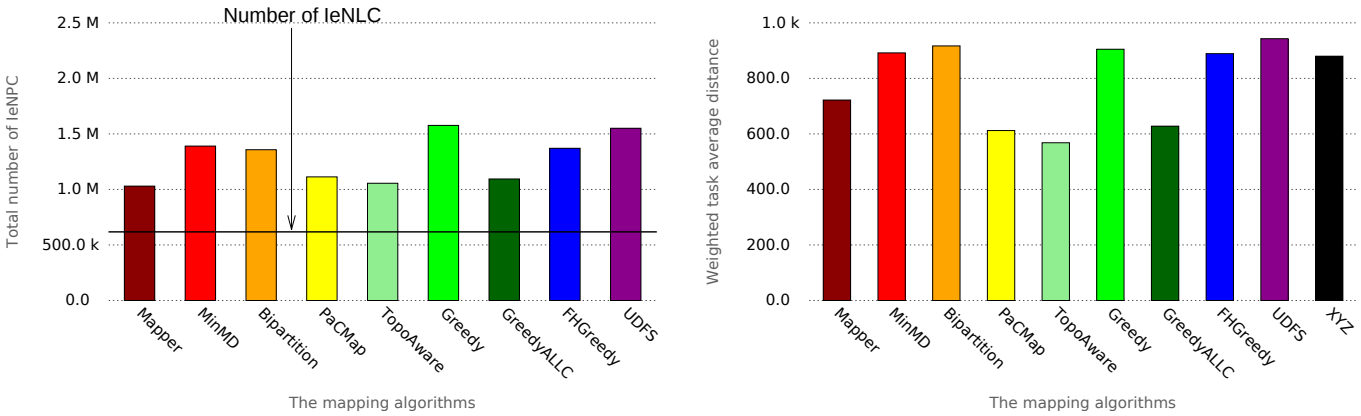
(a) Number of IeNPC (b) WTAD

Figure 4.3: Statistics for LU.C.64 on $4 \times 4 \times 4$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC



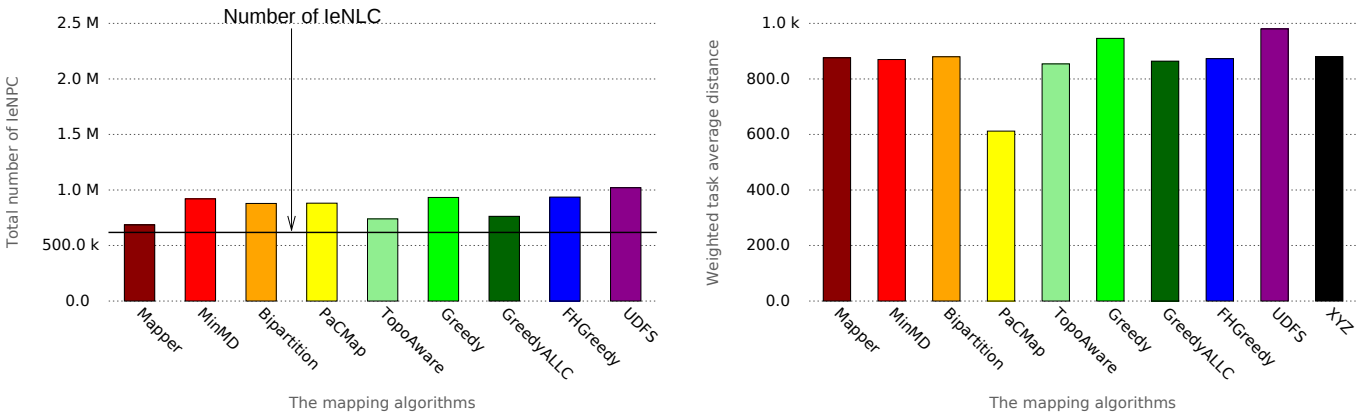
(a) Number of IeNPC (b) WTAD

Figure 4.4: BT.C.64 on $4 \times 4 \times 4$ 3-D Mesh



(a) Number of IeNPC (b) WTAD

Figure 4.5: BT.C.64 on $4 \times 4 \times 4$ 3-D Torus



(a) Number of IeNPC (b) WTAD

Figure 4.6: Statistics for BT.C.64 on $4 \times 4 \times 4$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC

Like the graphical plots show, a list of "leaders" from the algorithms could be created. There are 5 of the strategies, that demonstrate their quality, in average in the best way w.r.t. the both metrics, for all tests made for the parallel applications LU.C.64 and BT.C.64, namely:

- Bokhari's *Mapper*,
- *Partitioning and Center Mapping (PaCMap)*,
- *Topology-aware Task Mapping*,
- *GreedyALLC Strategy*,
- *Fast and High Quality Greedy Mapping.*

One need to compare the pre-simulation statistics, what were obtained during the experiments on the local machine, with the post-simulation statistics, in order to see whether the assumptions **"before"** hold **"after"** the simulation as well. In other words, whether the tendency of a reducing the number of IeNPC is keeping, or not, on the real physical computing machines. Therefore, the results of these five "best-quality" algorithms are chosen to check their performance after the simulations on the HAEC-SIM framework (see Chapter 5), that will return the statistics for each message's group (the groups are differed by message size): average number of hops (meaning physical links) traveled, average duration of transfer time in seconds, and average delivered transferred rate for the links in bytes per seconds. It is the most important aspect of using the HAEC-SIM simulations, testing whether the hypothesis before simulation are still valid after the simulation. This is a form of verification of the presumption that the mapping A is better/not than the mapping B for the application C on the architecture D.

One could now assume that the *GreedyALLC* mapping shows the best performance for LU parallel application of NAS benchmark on 3-D Torus network topology. The same observation for BT application can be done from the graphical plots: Bokhari's *Mapper* algorithm produces the best per-

formance values on HAEC Box topology. The assumptions will be validated in Chapter 5.

However, one thing should be stressed out. If one is interested only in time, the algorithms produce the mappings, one must distinguish time from the quality. Looking at the results, presented above, one can definitely figure out, that the best-quality algorithms are usually not among the **quickest** ones, even for the small problem size. Hence, it is necessary to keep in mind, whether one is ready to sacrifice the quality to time, needed to get some (sometimes, even the worst) results.

Tables 4.3 and 4.4 list the execution time of generating the mappings for the LU and BT (class C, 256 MPI processes) applications of the NAS parallel benchmarks.

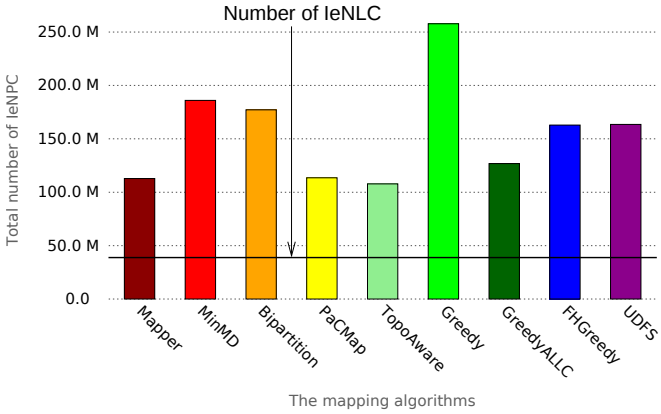
Algorithm	**Execution Time** [s]	**Execution Time** [s]	**Execution Time** [s]
LU.C.256	3-D Mesh	3-D Torus	HAEC Box
Mapper	18.069	129.993	570.569
MinMD	0.014	0.006	0.012
Bipartition	0.258	0.274	0.640
PaCMap	2.056	2.482	11.408
TopoAware	172.542	170.339	189.863
Greedy	0.632	0.718	10.341
GreedyALLC	0.973	0.964	2.537
FHGreedy	0.072	0.076	1.890
UDFS	0.130	0.142	0.402

Table 4.3: Time to generate the mapping (in sec) for LU.C.256 on $8 \times 8 \times 4$ topologies

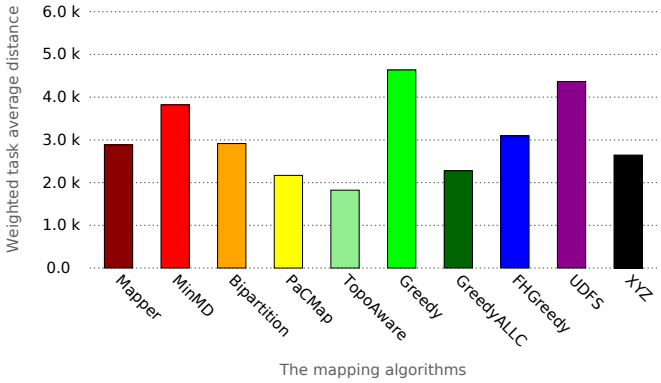
Algorithm	**Execution Time** [s]	**Execution Time** [s]	**Execution Time** [s]
BT.C.256	3-D Mesh	3-D Torus	HAEC Box
Mapper	100.995	169.521	1449.485
MinMD	0.015	0.009	0.014
Bipartition	0.142	0.153	0.344
PaCMap	2.321	2.872	12.935
TopoAware	171.838	176.841	217.270
Greedy	0.647	0.762	10.813
GreedyALLC	1.033	1.112	2.938
FHGreedy	0.155	0.121	3.066
UDFS	0.128	0.197	0.607

Table 4.4: Time to generate the mapping (in sec) for BT.C.256 on $8 \times 8 \times 4$ topologies

The statistics of the inter-node physical communications and weighted task average distances for the applications are shown in Figures 4.7 — 4.12.

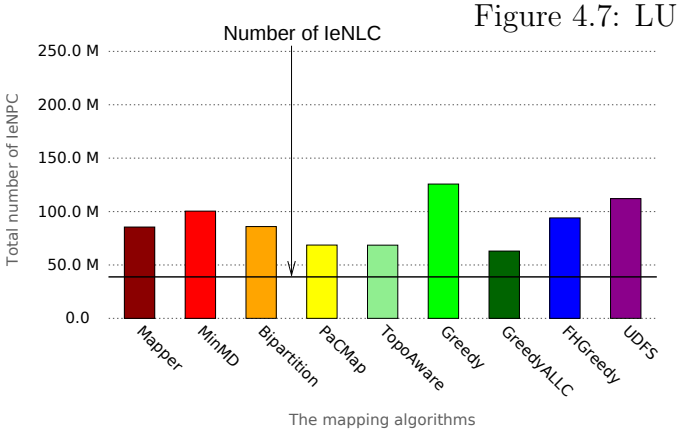


(a) Number of IeNPC

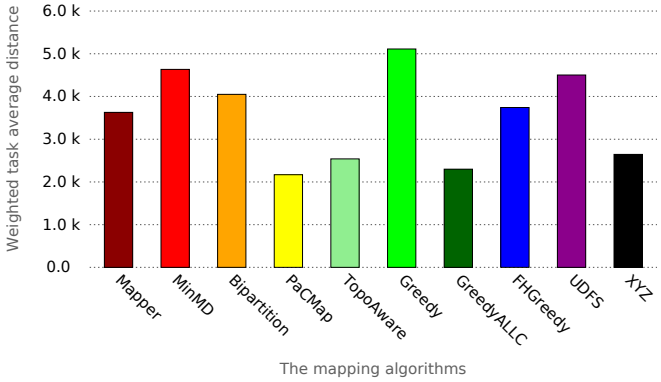


(b) WTAD

Figure 4.7: LU.C.256 on $8 \times 8 \times 4$ 3-D Mesh

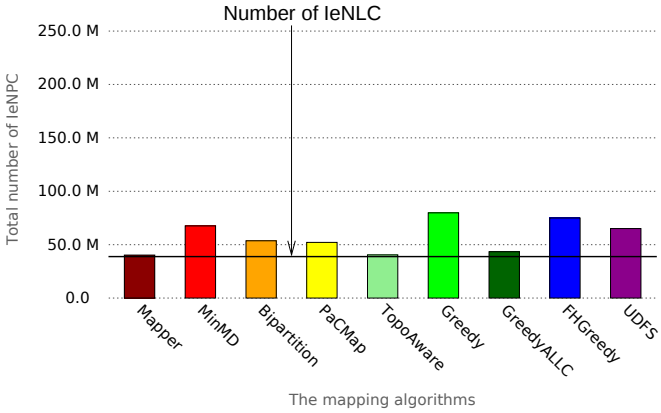


(a) Number of IeNPC

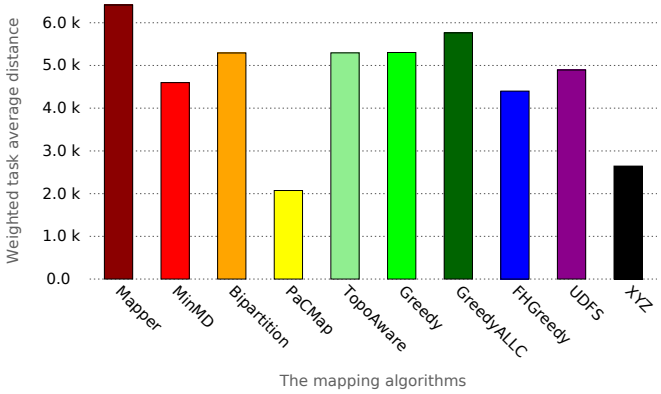


(b) WTAD

Figure 4.8: LU.C.256 on $8 \times 8 \times 4$ 3-D Torus

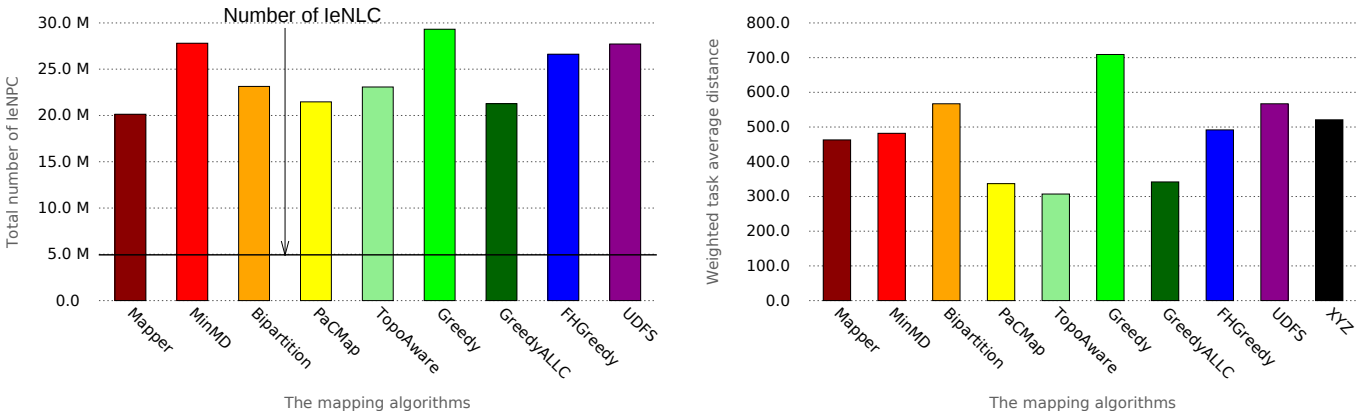


(a) Number of IeNPC



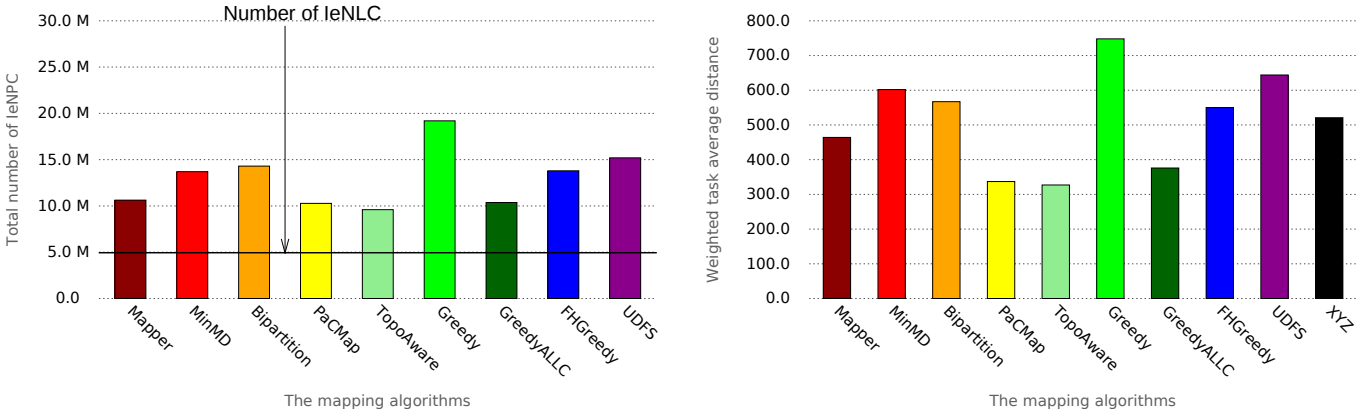
(b) WTAD

Figure 4.9: Statistics for LU.C.256 on $8 \times 8 \times 4$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC



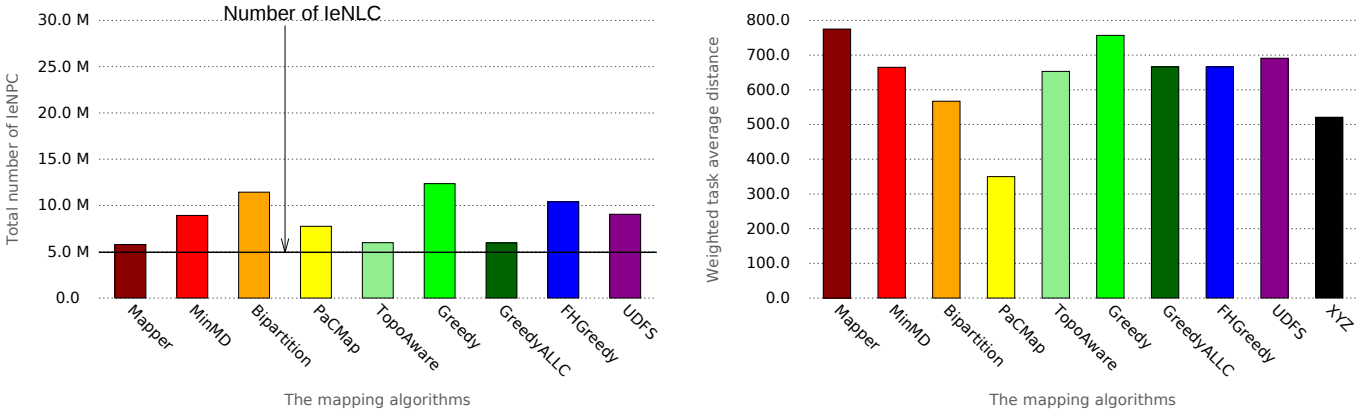
(a) Number of IeNPC (b) WTAD

Figure 4.10: BT.C.256 on $8 \times 8 \times 4$ 3-D Mesh



(a) Number of IeNPC (b) WTAD

Figure 4.11: BT.C.256 on $8 \times 8 \times 4$ 3-D Torus



(a) Number of IeNPC (b) WTAD

Figure 4.12: Statistics for BT.C.256 on $8 \times 8 \times 4$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC

Table 4.5 lists the execution time of generating the mapping for the LU (class C, 1024 MPI processes) application of the NAS parallel benchmarks.

Algorithm	**Execution Time [s]**	**Execution Time [s]**	**Execution Time [s]**
LU.C.1024	3-D Mesh	3-D Torus	HAEC Box
Mapper	1245.601	1857.210	27915.100
MinMD	0.124	0.042	0.104
Bipartition	0.880	0.869	2.170
PaCMap	86.940	85.216	440.974
TopoAware	38996.723	38346.131	39997.982
Greedy	10.479	12.647	356.345
GreedyALLC	14.887	15.597	74.454
FHGreedy	0.477	0.449	40.165
UDFS	1.950	2.128	17.225

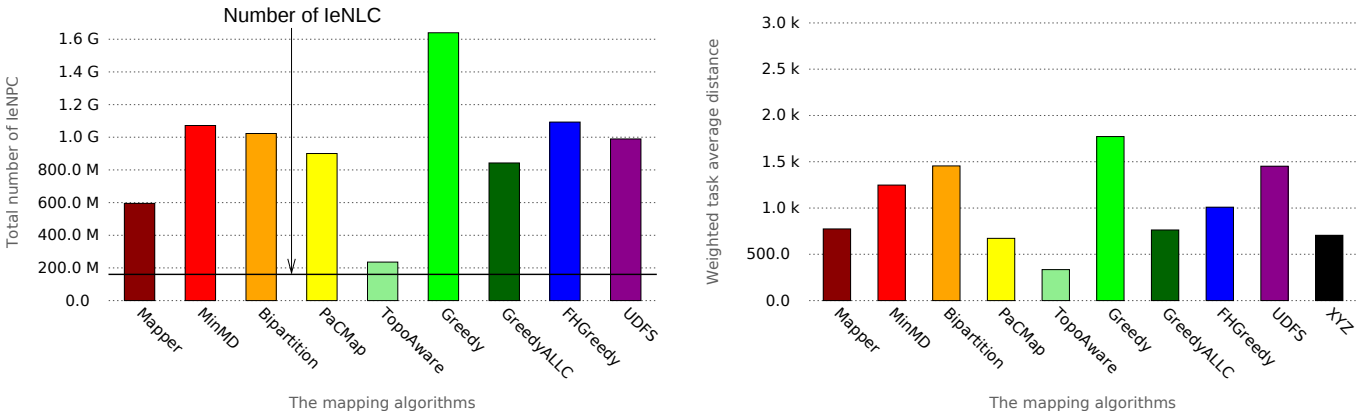
Table 4.5: Time to generate the mapping (in sec) for LU.C.1024 on $16 \times 8 \times 8$ topologies

Table 4.6 shows the execution time of generating the mapping for the BT (class C, 1024 MPI processes) application of the NAS parallel benchmarks.

Algorithm	**Execution Time [s]**	**Execution Time [s]**	**Execution Time [s]**
BT.C.1024	3-D Mesh	3-D Torus	HAEC Box
Mapper	3734.294	2448.568	32271.433
MinMD	0.303	0.059	0.138
Bipartition	0.917	0.933	2.236
PaCMap	85.160	91.245	422.109
TopoAware	37997.220	36066.030	40945.284
Greedy	10.629	12.504	343.360
GreedyALLC	17.502	17.298	76.144
FHGreedy	0.938	0.725	66.717
UDFS	1.982	2.062	17.273

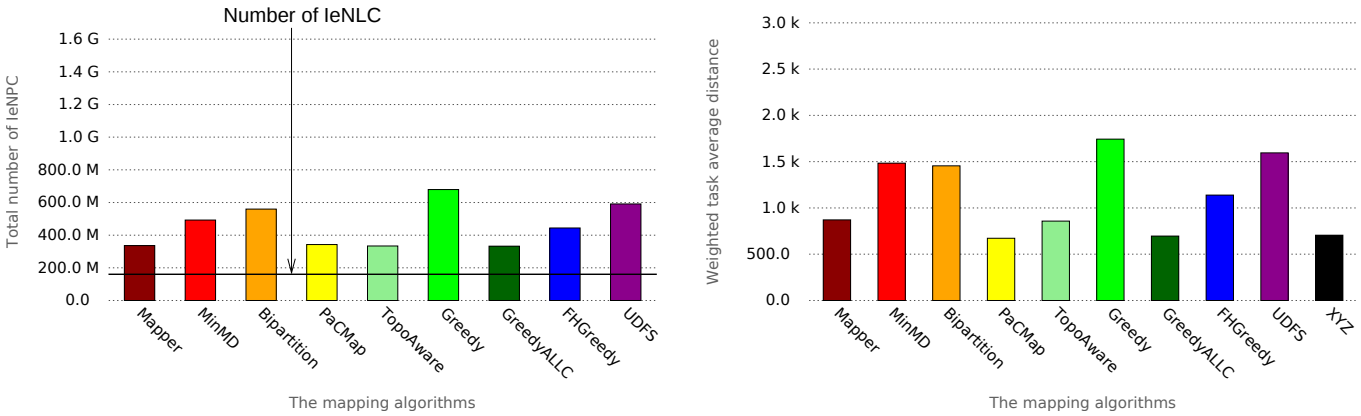
Table 4.6: Time to generate the mapping (in sec) for BT.C.1024 on $16 \times 8 \times 8$ topologies

The statistics of the inter-node physical communications and task average distances are illustrated in Figures 4.13 — 4.18 below.



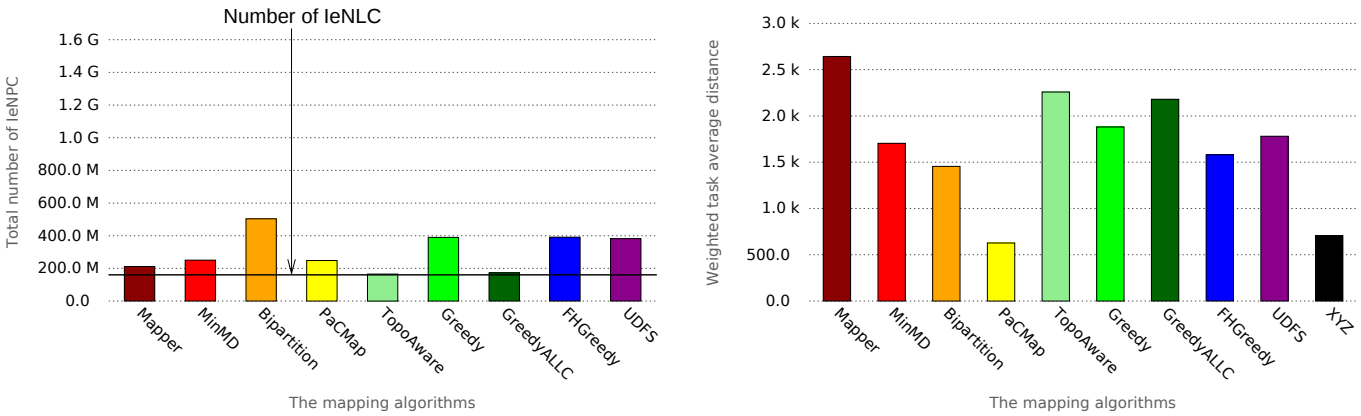
(a) Number of IeNPC (b) WTAD

Figure 4.13: LU.C.1024 on $16 \times 8 \times 8$ 3-D Mesh



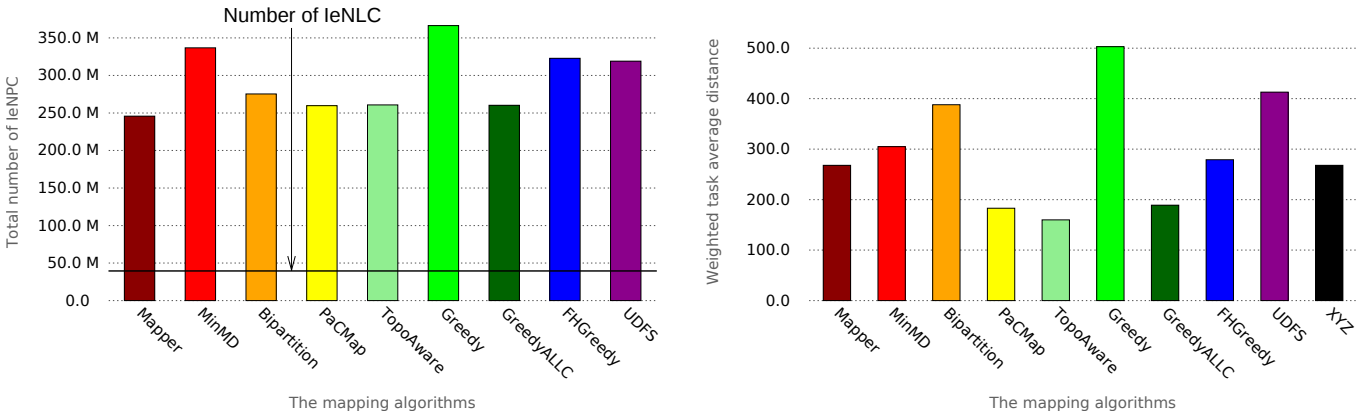
(a) Number of IeNPC (b) WTAD

Figure 4.14: LU.C.1024 on $16 \times 8 \times 8$ 3-D Torus



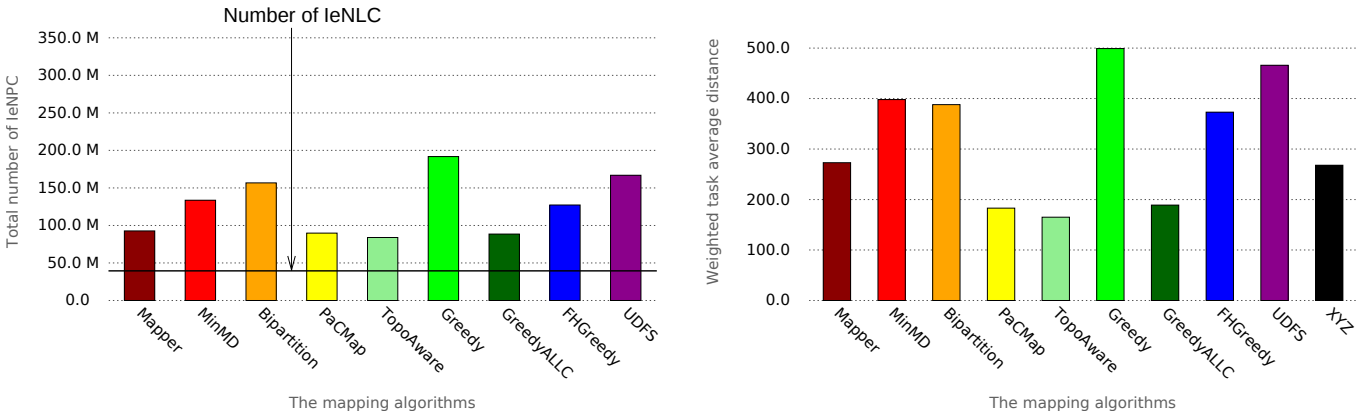
(a) Number of IeNPC (b) WTAD

Figure 4.15: Statistics for LU.C.1024 on $16 \times 8 \times 8$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC



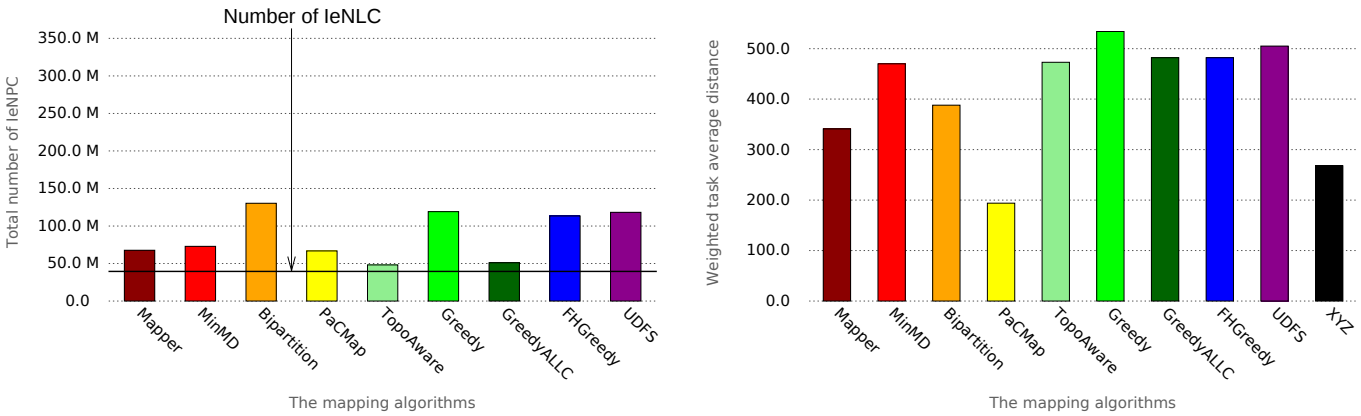
(a) Number of IeNPC (b) WTAD

Figure 4.16: BT.C.1024 on $16 \times 8 \times 8$ 3-D Mesh



(a) Number of IeNPC (b) WTAD

Figure 4.17: BT.C.1024 on $16 \times 8 \times 8$ 3-D Torus



(a) Number of IeNPC (b) WTAD

Figure 4.18: Statistics for BT.C.1024 on $16 \times 8 \times 8$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC

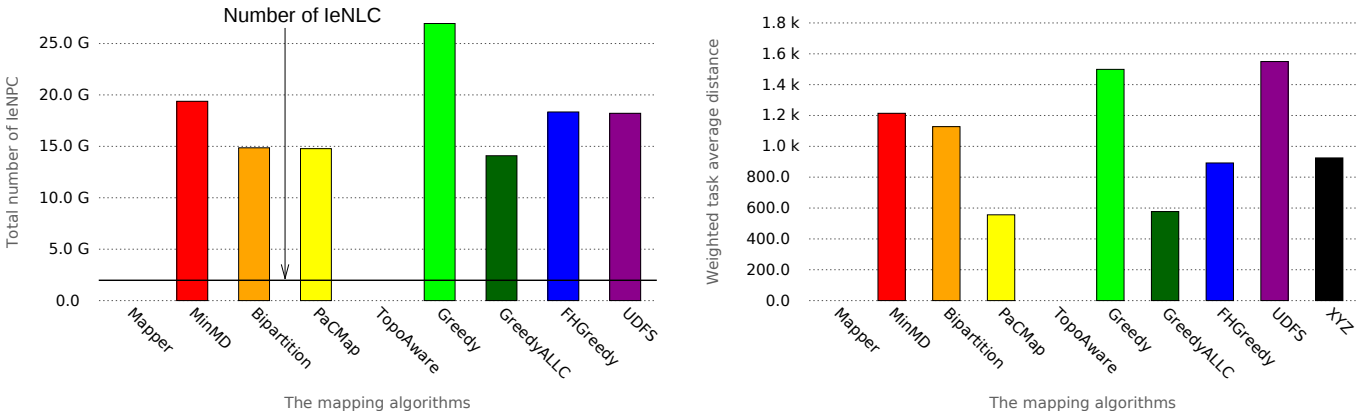
Tables 4.7 and 4.8 demonstrate the execution time of generating the mappings for the LU and BT (class D, 4096 MPI processes) applications of the NAS parallel benchmarks. One thing to be noted: seeing the tendency of the execution time increasing for Bokhari's *Mapper* and *Topology-aware* algorithms (even for 1024 processes execution time is about 10 hours), it seemed wiser to not test these mapping strategy for 4096 processes. Therefore, their algorithmic run time results and statistics are not included in the present work. The statistics of the inter-node physical communications and task average distances for other algorithms are given below on pages 56— 57.

Algorithm	**Execution Time** [s]	**Execution Time** [s]	**Execution Time** [s]
LU.D.4096	3-D Mesh	3-D Torus	HAEC Box
Mapper	*	*	*
MinMD	1.062	0.238	0.959
Bipartition	8.026	8.230	19.162
PaCMap	4302.019	4402.985	15824.939
TopoAware	*	*	*
Greedy	181.425	188.204	11503.082
GreedyALLC	248.142	238.176	2229.684
FHGreedy	4.377	3.919	738.127
UDFS	38.592	40.302	520.989

Table 4.7: Time to generate the mapping (in sec) for LU.D.4096 on $16 \times 16 \times 16$ topologies

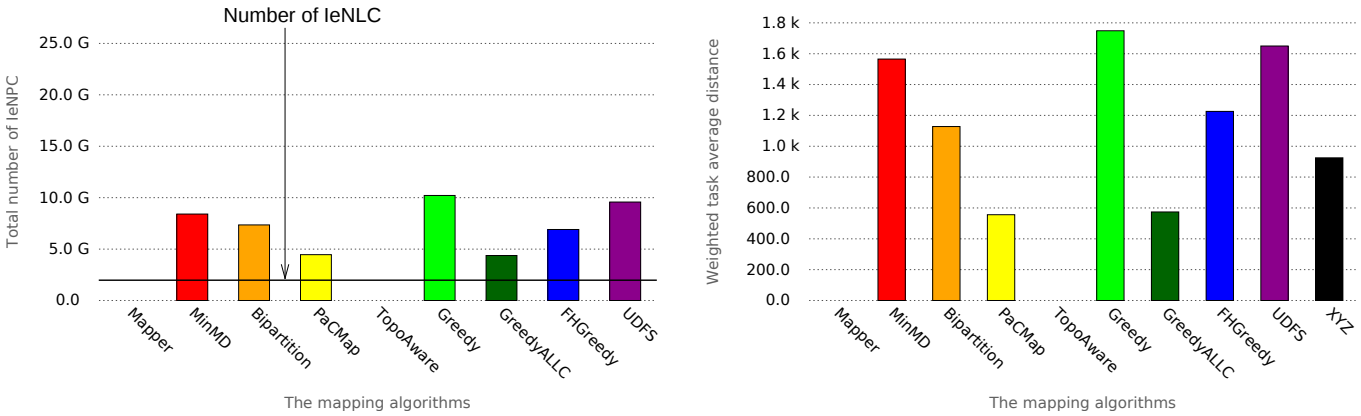
Algorithm	**Execution Time** [s]	**Execution Time** [s]	**Execution Time** [s]
BT.D.4096	3-D Mesh	3-D Torus	HAEC Box
Mapper	*	*	*
MinMD	2.075	0.382	1.180
Bipartition	8.235	9.460	22.389
PaCMap	4586.804	4902.733	17198.404
TopoAware	*	*	*
Greedy	182.596	219.868	12222.683
GreedyALLC	274.469	282.343	2231.909
FHGreedy	6.639	5.554	1424.939
UDFS	36.930	38.425	508.314

Table 4.8: Time to generate the mapping (in sec) for BT.D.4096 on $16 \times 16 \times 16$ topologies



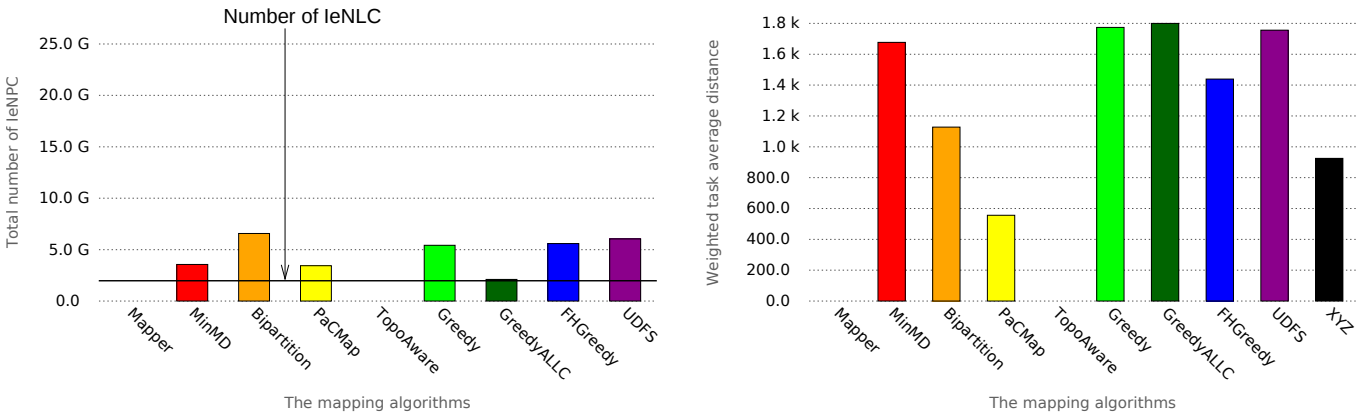
(a) Number of IeNPC (b) WTAD

Figure 4.19: LU.D.4096 on $16 \times 16 \times 16$ 3-D Mesh



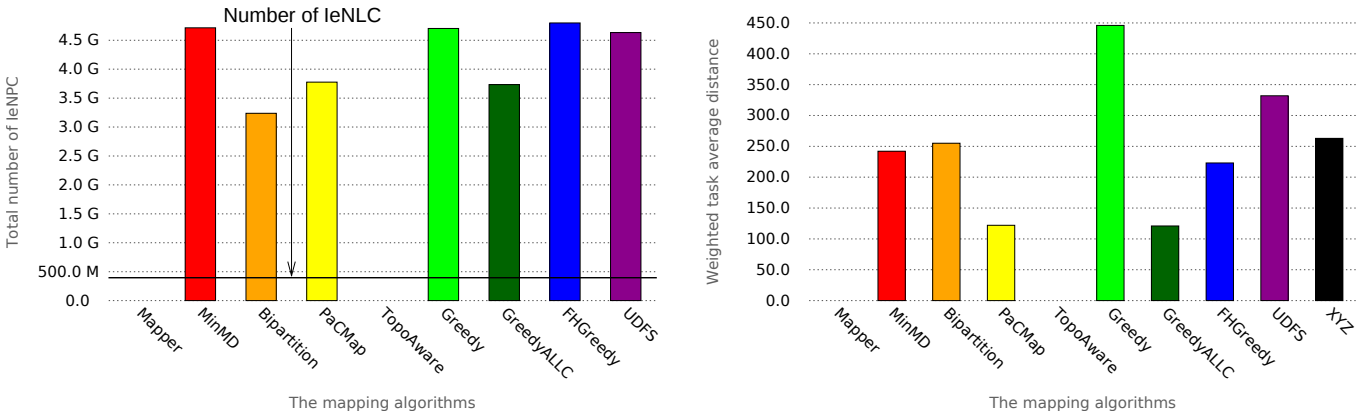
(a) Number of IeNPC (b) WTAD

Figure 4.20: LU.D.4096 on $16 \times 16 \times 16$ 3-D Torus



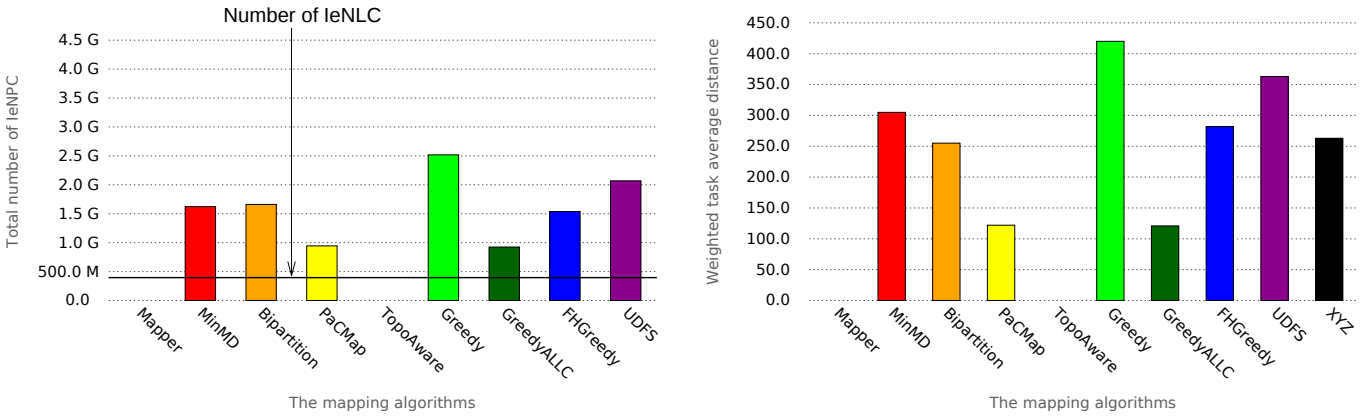
(a) Number of IeNPC (b) WTAD

Figure 4.21: Statistics for LU.D.4096 on $16 \times 16 \times 16$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC



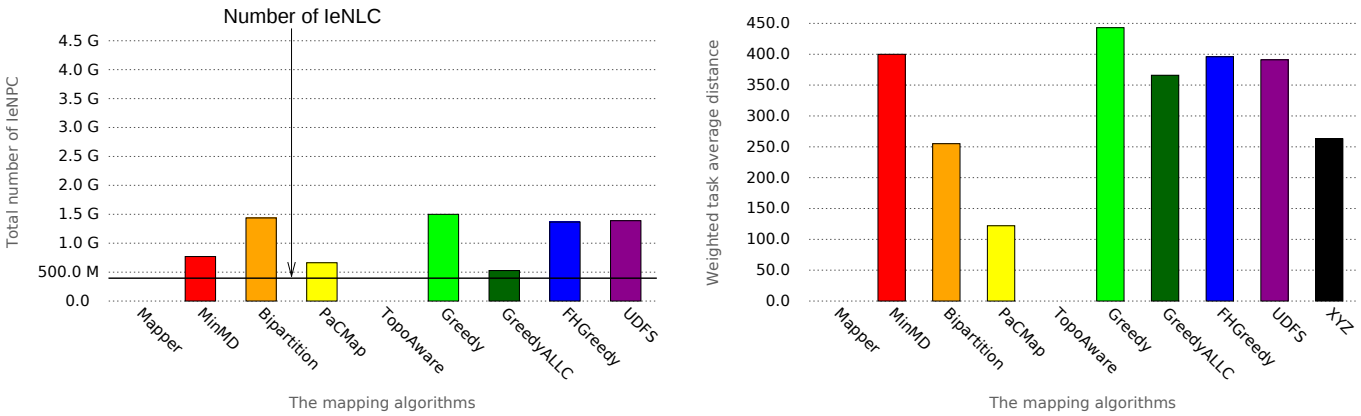
(a) Number of IeNPC (b) WTAD

Figure 4.22: BT.D.4096 on $16 \times 16 \times 16$ 3-D Mesh



(a) Number of IeNPC (b) WTAD

Figure 4.23: BT.D.4096 on $16 \times 16 \times 16$ 3-D Torus



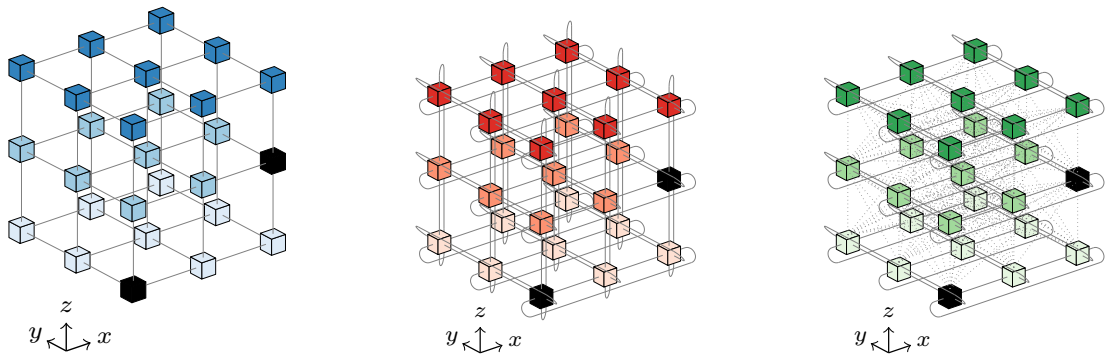
(a) Number of IeNPC (b) WTAD

Figure 4.24: Statistics for BT.D.4096 on $16 \times 16 \times 16$ HAEC Box. The horizontal black line in the left plots represents the total value of IeNLC

4.2 Discussion of the Results

As it can be seen from the graphical plots, Bokhari's *Mapper, Partitioning and Center Mapping, Topology-aware Task Mapping, GreedyALLC Mapping, and Fast and High Quality Mapping* algorithms are keep demonstrating their good performance for other applications with different sizes. However, the pre-simulation step has shown, that the winner of the mappings competition could be already chosen. The algorithm *GreedyALLC*, proposed in [27], produces almost in all cases superb results — the algorithm creates the mappings with good values of the both metrics in **reasonable** amount of time (in comparison with *Mapper* and *Topology-aware* algorithms) for **all** topologies sizes.

The general tendency for all applications of different sizes and mapping algorithms is that the number of inter-node physical communications (IeNPC) has a decreasing nature from the 3-D Mesh, over the 3-D Torus to the HAEC Box topology. This fact is associated with the increasing number of links in the topologies and with the decreasing diameter of the topologies (longest path length between two processors). If $4 \times 4 \times 4$ 3-D Mesh has the diameter equal to 9, 3-D torus — 6, HAEC Box has already the diameter equal to 4.



(a) 3-D Mesh (b) 3-D Torus (c) HAEC Box

Figure 4.25: Graphical representation of the $3 \times 3 \times 3$ topologies

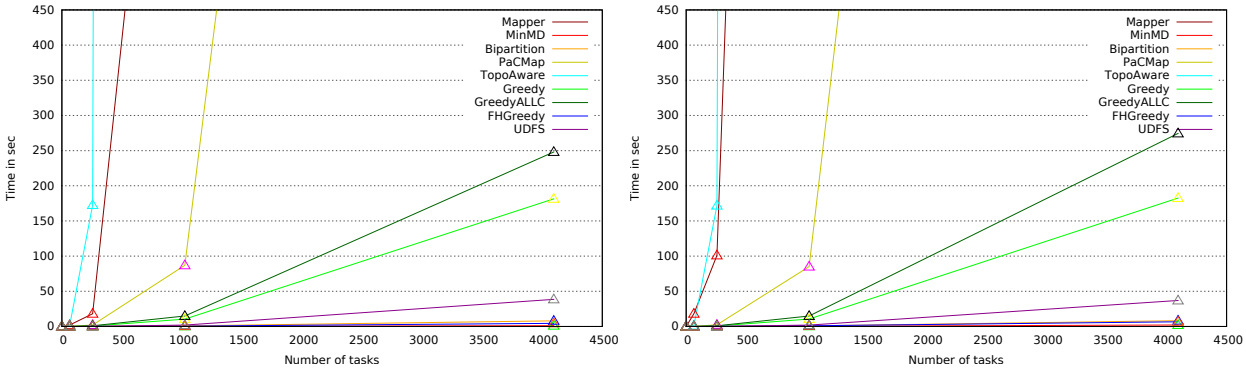
As a consequence, the average distance (number of hops) between any two nodes is decreasing, as the new path, the sent messages follow to reach their destination, appear.

Considering the example, presented in Figure 4.25, one can see that the number of hops between nodes (0,0,0) and (2,0,1) (the nodes are highlighted in black) is decreasing over the topologies. If in case of 3-D Mesh the minimum distance between them is 3 hops, in case of 3-D Torus the distance is already 2 hops (due to the topological property of such networks), and in case of HAEC Box just 1 hop thanks to the construction, where two adjacent plates are connected through wireless links.

The execution time could be the most important constraint in choosing specific mapping algorithm. In case of *Topology-aware* and Bokhari's *Mapper* strategies it plays a significant role. Despite the fact that these two approaches demonstrate the results, comparable with the performance of *GreedyALLC* algorithm and sometimes even better, their execution time for the topologies with small sizes (64 and 256 MPI processes) exceed the execution time of *GreedyALLC* by several times; in case of 1024 MPI processes the difference in time is already hundreds and thousands times, respectively.

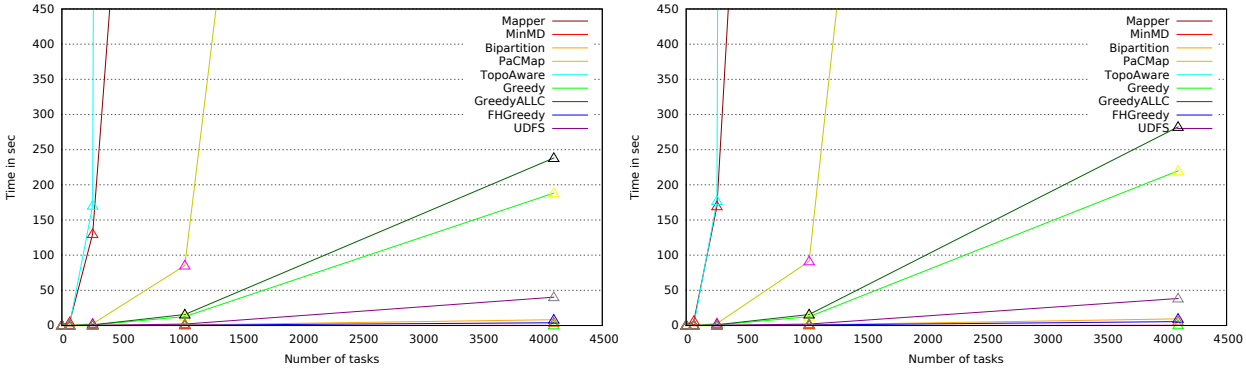
Presented in Figures 4.26—4.28 graphical plots of the algorithmic run time illustrate the overall tendency of the increasing in time with increasing the problem sizes. One can definitely see, that almost all graphics have an exponential nature of growth, except *Minimum Manhattan Distance* and *Recursive Bipartitioning* algorithms, that have a linear growth.

The execution time of *Mapper* and *Topology-aware Task Mapping* algorithms show the exponential explosion for the experiments on all network topologies — once the problem size becomes more than 64 processes, the algorithms rapidly slow down their performance. The same observation is valid as well for *PaCMap* mapping algorithm — the difference in time for 1024 and 4096 processes is about 50 times for all topologies.



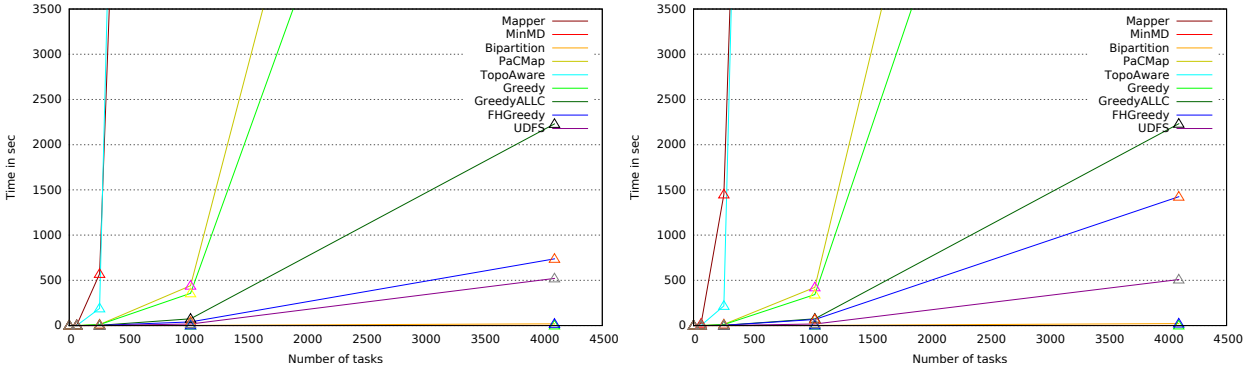
(a) LU parallel application (b) BT parallel application

Figure 4.26: Time to generate the mappings (in sec) for parallel applications for 3-D Mesh on the local machine



(a) LU parallel application (b) BT parallel application

Figure 4.27: Time to generate the mappings (in sec) for parallel applications for 3-D Torus on the local machine



(a) LU parallel application (b) BT parallel application

Figure 4.28: Time to generate the mappings (in sec) for parallel applications for HAEC Box on the local machine

Chapter 5

Simulations with HAEC-SIM

In this chapter the comparison of the mapping algorithms is given in terms of the mapping-related statistics (the number of IeNPC) obtained in pre-simulation step and after simulation on the HAEC-SIM framework, as well as in terms of the message-related statistics, using the HAEC-SIM module **trace_stats**, discussed in Section 3.3.3.

5.1 Proposed Approach

As mentioned before in Section 4.1, the validation of the mappings results can be carried out as the valuation of the mappings impact for benchmark traces mapped to mesh, torus, or HAEC Box topologies. This was the initial plan for the comparison of different mapping strategies. However, one more possibility to check the results arose from the work process. It turned out, that one could as well compare the impact of the mappings for benchmark traces obtained from a simulating the execution with the default mapping *"xyz"* of the benchmarks on mesh, torus, HAEC Box topologies, which were subsequently mapped to mesh, torus, and HAEC Box with the best mappings, chosen in Chapter 4. It means, that before any simulation one need to perform one additional step to already considered. First, the traces of the NAS benchmark applications are mapped to the topologies with default

mapping *"xyz"*, i.e. for $4 \times 4 \times 4$ topology the task with $ID = 0$ is mapped to the node with $(0, 0, 0)$ *xyz* coordinates, the task with $ID = 1$ is mapped on the node $(1, 0, 0)$, the task with $ID = 63$ — on the node $(3, 3, 3)$, respectively. Then, the resulting simulation traces are taken as input for the further simulations, using the mapping files obtained after pre-simulation step. Finally, a generating the statistics with module **trace_stats** for these new simulated traces allows to compare them with the input traces (generated with *"xyz"* mapping strategy) in terms of a message-level comparison of transfer rates, hops travelled per message, etc., and could show the effectiveness of the mapping strategy.

The main purpose of doing the simulation is to answer the question: *Which mapping and which topology result in the best performance for the given application?* The simulation results must provide an answer on that question.

The applications LU and BT of class C, compiled with 64 MPI processes, were chosen as test instances for the following reasons. All simulations were conducted on KNL Intel Xeon Phi processors, that have 64 CPU with 1 thread per core on board. Therefore, 64 processes in total suit perfectly for running the simulation, where for each process one thread is dedicated. Second, the total simulation time and the input trace size are still reasonable.

5.2 Comparison of Mapping-related Statistics

Only the most successful (based on the showed results) mappings from Chapter 4 for each of specified topology (3-D Mesh, 3-D Torus and HAEC Box) and for two applications LU.C.64 and BT.C.64 should be tested with HAEC-SIM: Bokhari's *Pairwise Interchange Algorithm: Mapper, Partitioning and Center Mapping (PaCMap), Topology-aware Task Mapping, GreedyALLC Strategy*, and *Fast and High Quality Mapping*. The comparison of the impact of the mappings for benchmark traces obtained on "miniHPC" machine, but

mapped during the HAEC-SIM simulation to the mesh, torus, and HAEC Box topologies is considered here.

The number of IeNPC (post-simulation) can be approximated as follows:

$$IeNPC_{post-sim} \approx \sum_{i}^{G} \#hops_i \times \#messages_i, \tag{5.1}$$

where G is a number of message size groups in the statistics file, obtained after the simulation with HAEC-SIM module **trace_stats**, $hops_i$ is the average number of hops in the i^{th} message size group, and $messages_i$ is the number of messages in i^{th} message size group. The Table 5.1 represents the look of such statistics file:

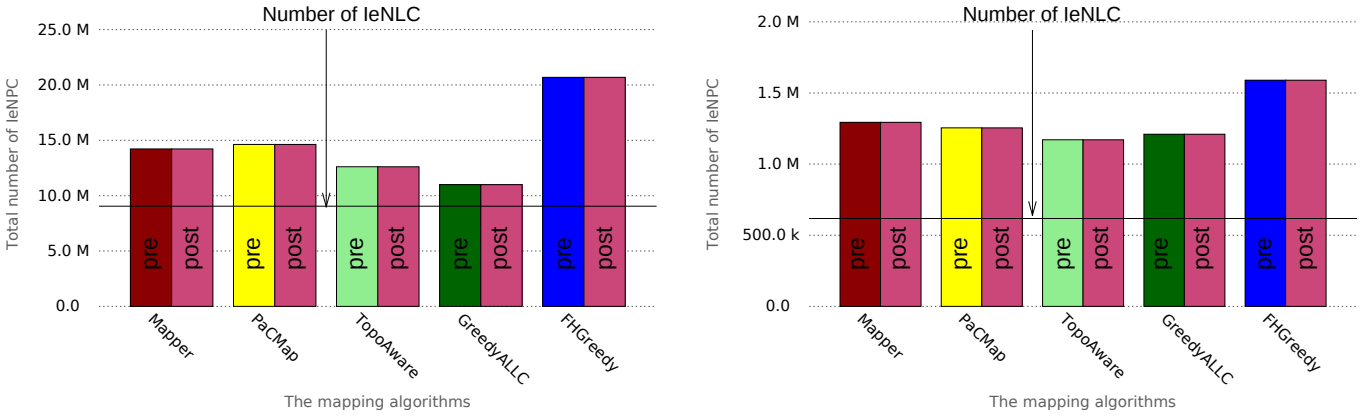
68533654909634ps				
TOT 4383153578089373ps				
APP 3527659318077742ps 80.4822%				
MPI 855494260011631ps 19.5178%				
Message size	**Number of messages**	**Avg. number of hops**	**Avg. duration**	**Avg. delivered transferred rate**
320 B	# 42	1.57143 hops	8.74411e-06 s	3.65961e+07 B/s
336 B	# 14	1.57143 hops	7.4511e-06 s	4.5094e+07 B/s
352 B	# 42	1.57143 hops	1.08433e-05 s	3.24625e+07 B/s
368 B	# 14	1.57143 hops	1.11889e-05 s	3.28897e+07 B/s
760 B	# 1124480	1.5 hops	4.21283e-05 s	1.80401e+07 B/s
800 B	# 6746880	1.57143 hops	2.94626e-05 s	2.71531e+07 B/s
840 B	# 1124480	1.64286 hops	2.77611e-05 s	3.02581e+07 B/s
1296 B	# 28	1.5 hops	9.70617e-06 s	1.33523e+08 B/s
259200 B	# 42672	1.57143 hops	0.000869493 s	2.98105e+08 B/s
272160 B	# 14224	1.57143 hops	0.000893913 s	3.04459e+08 B/s
OVERALL **B**	# **9052876**	**1.57143 hops**	**3.6142e-05 s**	

Table 5.1: Statistics file for LU.C.64 on 3-D Mesh with Mapper algorithm

The first line of the Table 5.1 shows the total duration of the trace in picoseconds (1e-12 sec). Then, total accumulative time in picoseconds spent in APP (application computation functions) and MPI (communication) functions, respectively: TOT = APP + MPI. Two next lines have as values total accumulative time and percentage from the total amount of time spent in

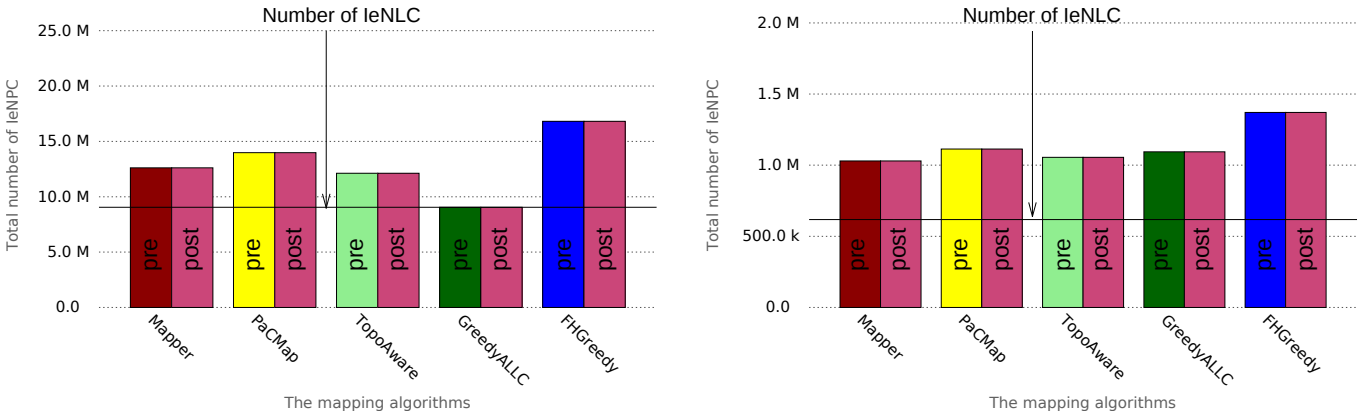
APP functions (or MPI communication functions, respectively). The columns demonstrate the followings: the first one — a message size; the second one — a number of messages of that size; third one — an average number of hops for each message to travel; the fourth one — an average duration in seconds; and fifth one — an average delivered transferred rate for the links.

The graphics below in Figure 5.1—5.3 represent the comparison of the results obtained before (pre-simulation) and after (post-simulation) the simulations in terms of the number of inter-node physical communications (IeNPC). The purple bars beside each of the colored bar, representing the number of IeNPC of each algorithm, are the values of IeNPC approximated from the statistics files after the simulations and calculated with the equation (5.1).



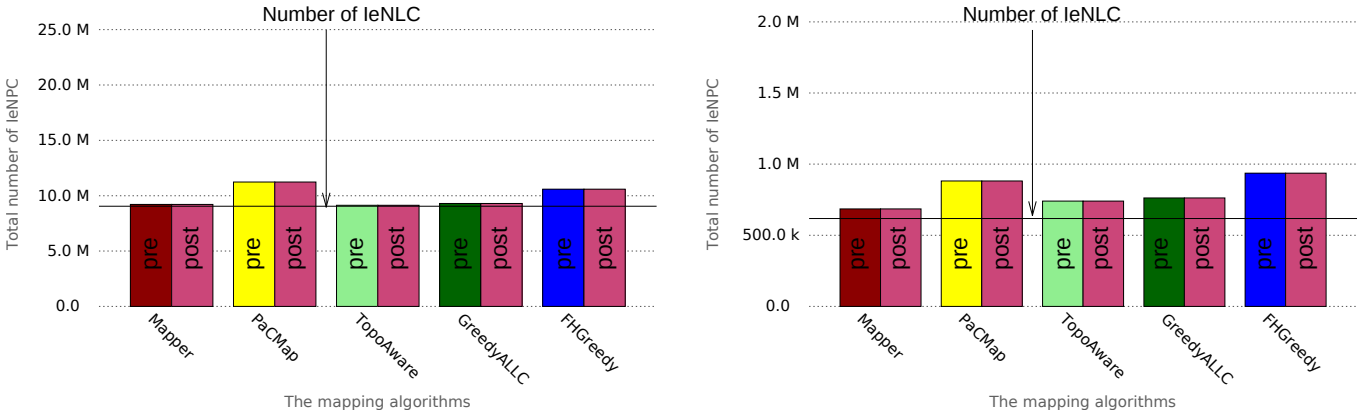
(a) LU.C.64 (b) BT.C.64

Figure 5.1: Comparison of pre-simulation and post-simulation statistics for NAS parallel applications in terms of total $IeNPC_{pre-sim}$ and total $IeNPC_{post-sim}$ on $4 \times 4 \times 4$ 3-D Mesh. The horizontal black line in the plots represents the total value of IeNLC.



(a) LU.C.64 (b) BT.C.64

Figure 5.2: Comparison of pre-simulation and post-simulation statistics for NAS parallel applications in terms of total $IeNPC_{pre-sim}$ and total $IeNPC_{post-sim}$ on $4 \times 4 \times 4$ 3-D Torus. The horizontal black line in the plots represents the total value of IeNLC.



(a) LU.C.64 (b) BT.C.64

Figure 5.3: Comparison of pre-simulation and post-simulation statistics for NAS parallel applications in terms of total $IeNPC_{pre-sim}$ and total $IeNPC_{post-sim}$ on $4 \times 4 \times 4$ HAEC Box. The horizontal black line in the plots represents the total value of IeNLC.

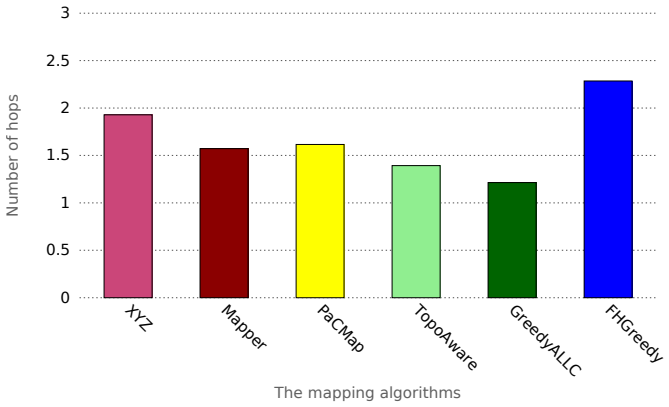
It can be observed from the presented graphics, that in all cases the results showed by the algorithms during the pre-simulation step are still valid in case of the mapping the initial benchmark traces on the mesh, torus, and HAEC Box topologies. For all of three topologies results are the same with the

predicted ones in the pre-simulation step. It means, that if only evaluate mappings based on the metrics known before any HAEC-SIM simulations, one can have a high confidence that a certain mapping will result in the expected performance, if a HAEC-SIM simulation is performed, or if the benchmark is mapped on a real system with this mapping.

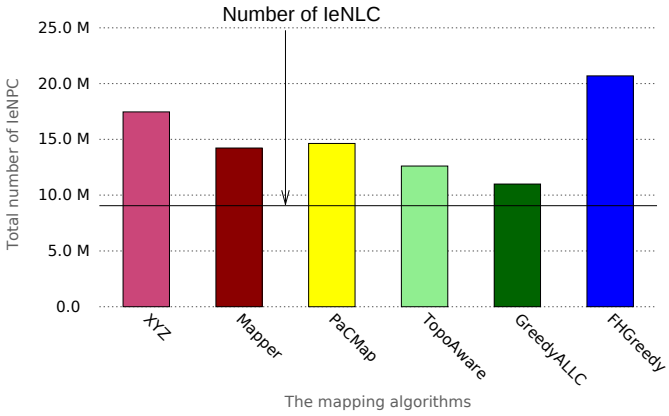
Now it is time to answer on the question, that has been asked before in Section 5.1. Analyzing the graphics, one can definitely response: for the parallel NAS benchmark LU of class C the 3-D Torus topology and the mapping, produced by *GreedyALLC* algorithm, result the best performance; in case of BT parallel application — HAEC Box topology with applied to the communication matrix of the application Bokhari's *Mapper*.

5.3 Comparison of Message-related Statistics

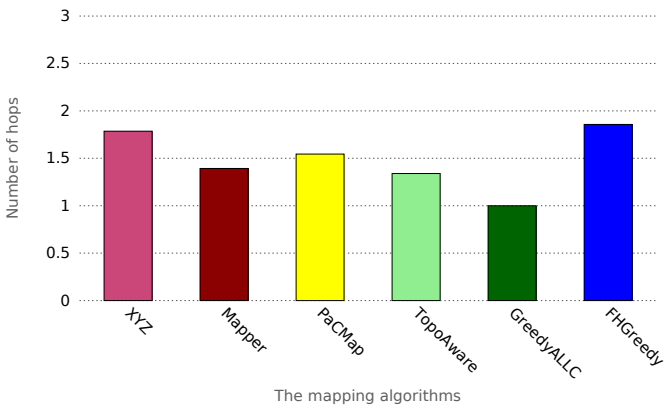
In order to compare the mappings in terms of message-related statistics, one needs to perform the following steps. First, to obtain the statistics file from the initial trace, namely from the benchmark trace mapped either on mesh, either on torus, or on HAEC Box with default *"xyz"* mapping. Second, the simulated trace from the first step has to be mapped once again on the topologies, however, with mapping files, that were chosen as the best after pre-simulation step of the experiment part. Finally, the statistics files are created from the new traces and could be used for the purposes of the comparison. Based on the information retrieved with the module **trace_stats**, one is able now to compare the delivered transferred rate for each message (in B/s) and the number of hops travelled for each message.



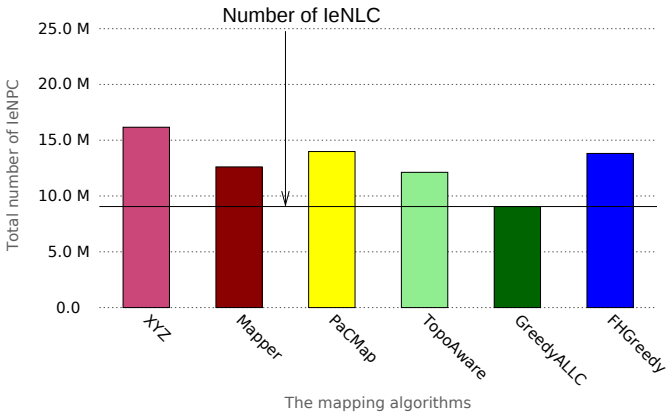
(a) 3-D Mesh. Average number of hops



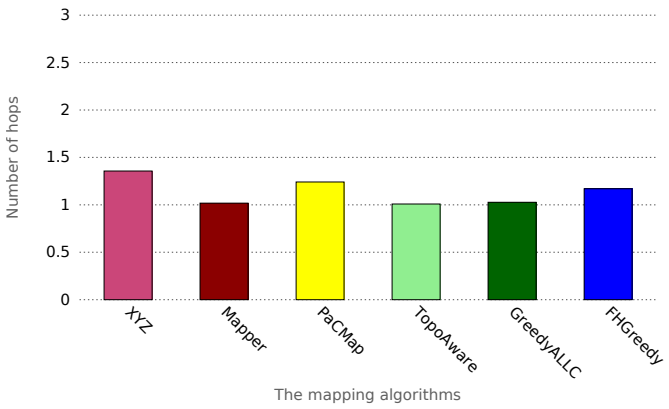
(b) 3-D Mesh. IeNPC



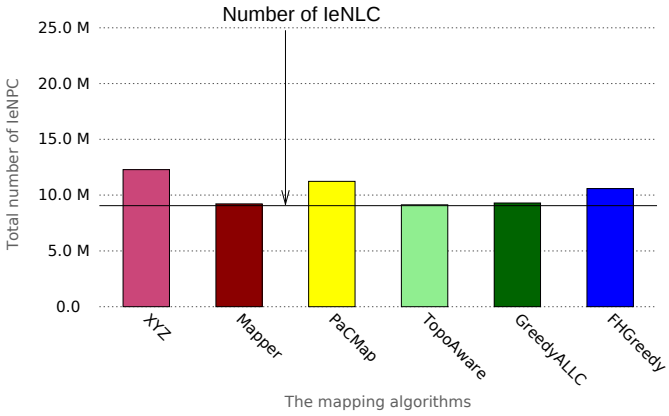
(c) 3-D Torus. Average number of hops



(d) 3-D Torus. IeNPC



(e) HAEC-Box. Average number of hops



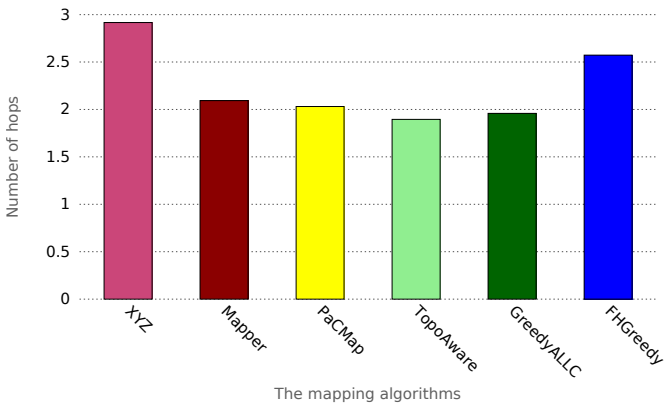
(f) HAEC-Box. IeNPC

Figure 5.4: Message statistics for LU.C.64 mapped on $4 \times 4 \times 4$ topologies.
The horizontal black line in the right plots represents the total value of IeNLC

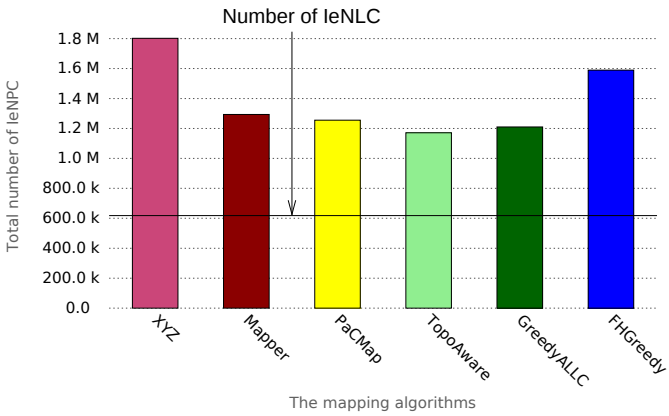
In Figure 5.4 the average number of hops, each message should traverse, and calculated (with formula (5.1) from the section 5.2) number of inter-

node physical communications (IeNPC) for parallel application LU are depicted. It can be seen that the algorithms *GreedyALLC* and *Topo-aware mapping* strategy demonstrate the best performance in case of mapping LU parallel benchmark on topologies with positions (mapping) files, produced by the algorithms. *GreedyALLC* even shows the least number of hops for all topologies, that means, that the most communicative pairs are placed as close as possible, which is reflected in number of IeNPC: in all cases it is equal or slightly more than the number of inter-node logical communications (IeNLC). Exactly this value has been chosen as a measure of the algorithms performance — all of them are striving to reach it. The chosen 5 best algorithms have better values of IeNPC for 3-D Torus and HAEC Box topologies than the default *"xyz"* mapping demonstrates, what could be considered as an indicator of the improving the overall performance of the application in case of using positions files the algorithms produce.

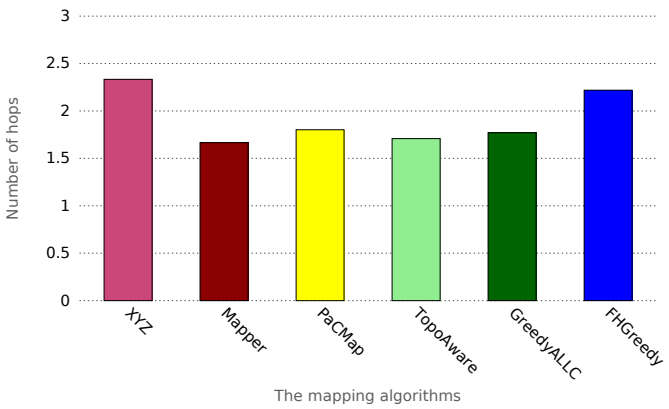
In case of BT parallel benchmark (see Figure 5.5), all 5 chosen algorithms have better values than a default *"xyz"* mapping: number of hops is less than produced by the initial mapping, that leads to the lesser number of IeNPC respectively.



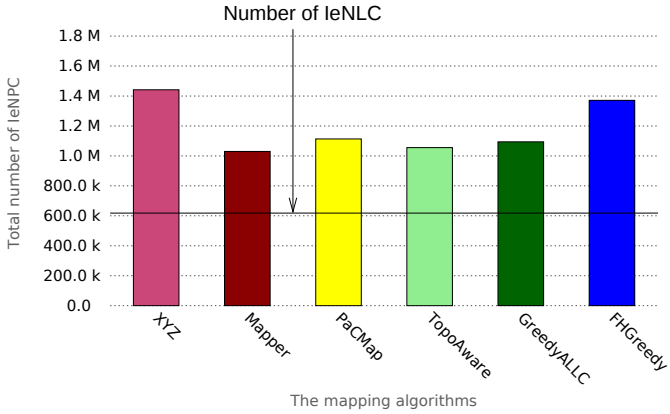
(a) 3-D Mesh. Average number of hops



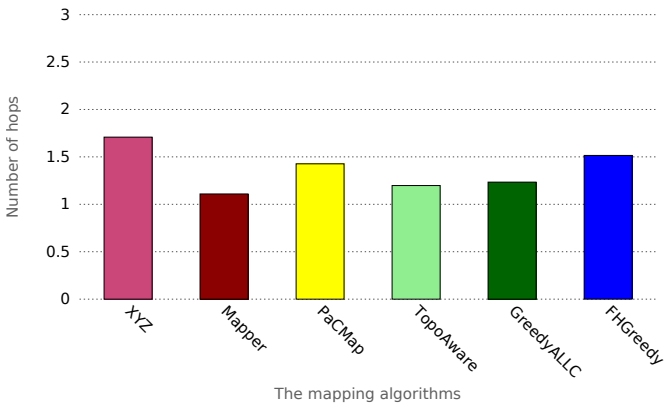
(b) 3-D Mesh. IeNPC



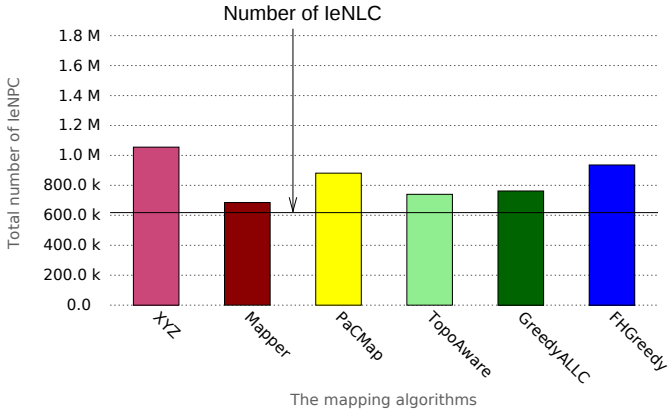
(c) 3-D Torus. Average number of hops



(d) 3-D Torus. IeNPC



(e) HAEC-Box. Average number of hops



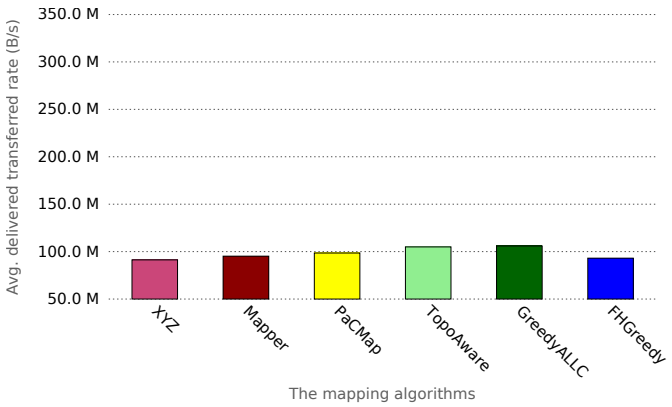
(f) HAEC-Box. IeNPC

Figure 5.5: Message statistics for BT.C.64 mapped on $4 \times 4 \times 4$ topologies.
The horizontal black line in the right plots represents the total value of IeNLC

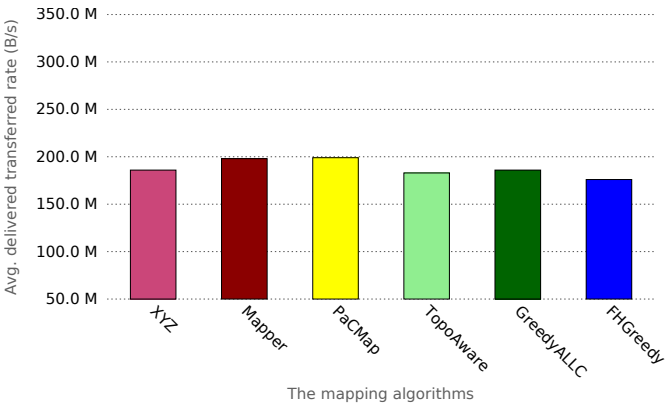
One should state, that in case of message-related statistics, the answer on the question, put in Section 5.1, is: for LU application the ideal topology is 3-D Torus with *GreedyALLC* mapping, and for BT — HAEC Box with Bokhari's algorithm *Mapper*. The winner algorithms and topologies are the same, as in case of the mapping-related statistics, considered in the section 5.2. Exactly these observation has been declared to be validated with the simulations in section 4.1.

One thing left to be considered is the average delivered transferred rate for

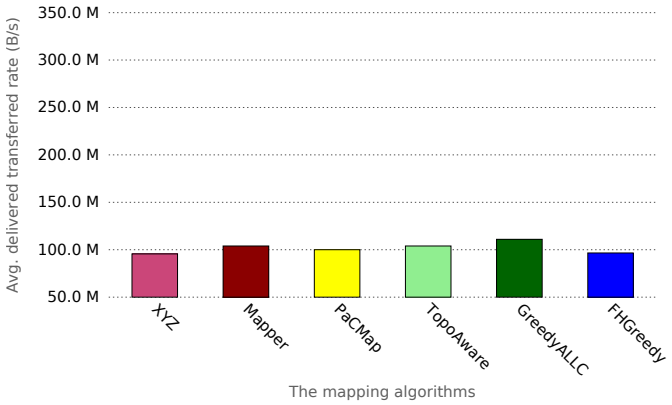
each message sent through the links of the topologies. The graphical plots in Figure 5.6 show these statistics.



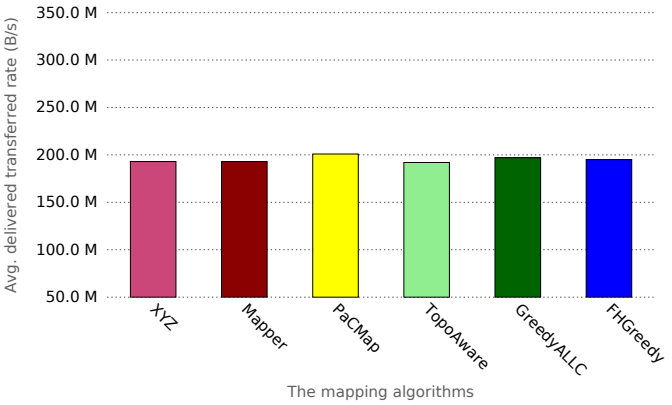
(a) 3-D Mesh. LU.C.64



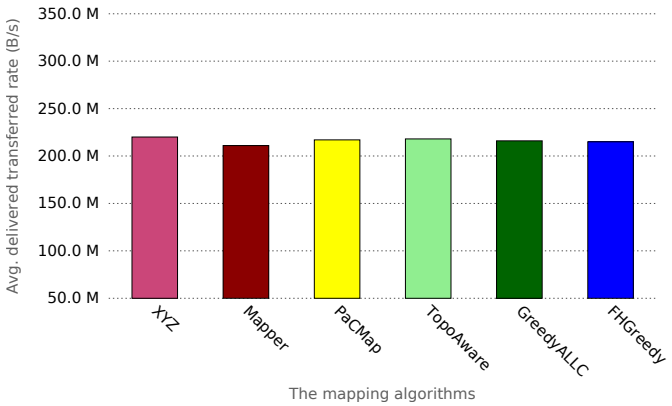
(b) 3-D Mesh. BT.C.64



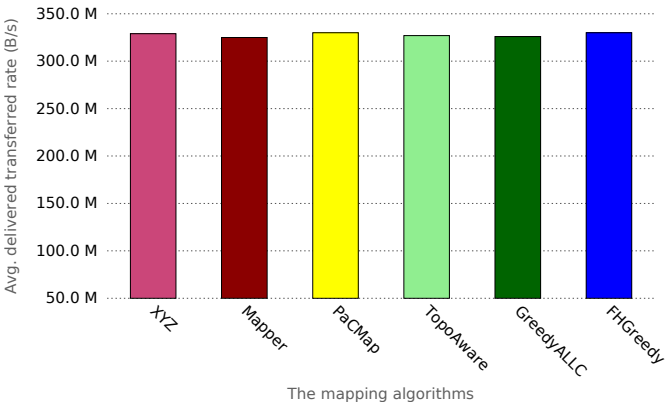
(c) 3-D Torus. LU.C.64



(d) 3-D Torus. BT.C.64



(e) HAEC-Box. LU.C.64



(f) HAEC-Box. BT.C.64

Figure 5.6: Average delivered transferred rate in B/s for applications on $4 \times 4 \times 4$ topologies

For 3-D Mesh and Torus the average gain in bandwidth is about 10% to showed one by the *"xyz"* mapping. For HAEC Box topology there is almost no difference in the links bandwidth. It could be explained by the fact that the HAEC-SIM framework yields optimistic simulations with contention-free resources. In other words, the multiple messages are simultaneously sent over the same link at the nominal bandwidth. All implemented network models by default in the simulator assume that there are no link contention, no link errors, and no link attackers [7], [17].

5.4 Discussion of the Results

Considering two types of the validation, one could make the following conclusion. The answers to the question: *Which mapping and which topology result in the best performance for the given application?* are the same in terms of as mapping-related statistics, as well as message-related statistics. For LU application the ideal topology is 3-D Torus with *GreedyALLC* mapping, and for BT — HAEC Box with *Mapper* strategy.

The results of the HAEC-SIM simulations demonstrated, that, first, one can have a high confidence, that a certain mapping will result in the expected performance, even without mapping the benchmark applications on a real HPC system. Second, using "winner" mappings, one can significantly decrease the number of traversed by each message hops (in comparison with default *"xyz"* mapping), that leads to the increasing of the overall applications performance.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The challenging issue for HPC is still being an energy consumption for computation and cooling [8]. One practical method to solve the problem is to improve *data locality*, in other words, the way the data are placed onto available processors units (CPU or cores of the network topology). In this thesis different strategies of mapping applications tasks onto the nodes of the underlying network topologies were considered and analyzed for their execution time as well as the performance of the produced mappings according to the defined performance metrics. As the results of the thesis show, the careful mapping of the parallel tasks on the nodes plays a significant role in decreasing the number of hops, each message should traverse, and the values of the inter-node physical communications, that leads to an improving the overall performance of the desired application. The simulations conducted on HAEC-SIM and their results evidence the assumptions, that if only evaluate a mapping based on the metrics known before any HAEC-SIM simulations, one can have a high confidence that the certain mapping will result in the expected performance, if a HAEC-SIM simulation is performed, or if the benchmark is mapped on a real system with this mapping.

However, the execution time of the mapping algorithms is still being the

most important constraint in selecting a specific mapping algorithm. Despite the fact that certain approaches demonstrate the comparable good results, their algorithmic run time could exceed the execution time of others in several times. The examples of Bokhari's *Pairwise Interchange Algorithm: Mapper* and *Topology-aware* mapping strategies prove this statement.

As asserted in [10], dilation is a measure of the total communication work performed by the network topology. It is an indicator of the total energy consumption. Congestion is a lower bound on time needed for communication. The choice of the underlying topology, the HPC parallel applications are being mapped onto, has a significant impact on the execution time, the overall performance and energy consumption of the applications. The results of the simulations, conducted with HAEC-SIM framework, demonstrated, that the HAEC Box topology helps to reach the best values of the performance metrics for all considered in the thesis algorithms and parallel applications. This fact is associated with the increasing number of links in the topologies (from 3-D Mesh over 3-D Torus to HAEC Box) and, as a consequence, the number of hops between any two nodes is decreasing. It means, that each of the applications tasks has to traverse lesser number of links, what decreases the dilation and congestion of the network topology, respectively.

6.2 Future Work

The mapping algorithms presented and considered in the thesis have as well some room for improvement. For example, implementing *Recursive Bi-partitioning* algorithm, the set of serial programs for partitioning graphs "METIS" [5] was used. However, one more graph partitioner could be given a try, namely "KAHIP" [30]. The authors of [27] claim, that the results of this partitioner are comparable with the results, produced by "METIS".

As mentioned in Section 2.5, there are 3 types of *Estimation functions*

the authors proposed in [12]. All of them were implemented and tested in `Python`. The running time of the algorithm with the first type of approximation is an order of $O(n|E_t|)$, where n is the size of both application and topology graphs, and $|E_t|$ is the cardinality of set of edges in the application graph. For comparison, overall execution time for second type and third type of approximation is in order of $O(n|E_t|)$ and $O(n^3)$, respectively [12]. Exactly, because of that fact, the results of the algorithm's execution with the first order approximation function were chosen and presented in this work. The first order approximation 2.5.1 demonstrated as well better performance results than others two during the implementation of the algorithms and test phase — the values of the metrics, discussed in Section 3.1, are lesser than showed by other two approximations. However, the authors of [12] claim, that their experimental results demonstrated that the second and third types of approximation outperform the first one by about 10-25%, for instance, in terms of the execution time and average number of hops travelled by each byte. Thus, the *Topology-aware Task Mapping* algorithm should be thoroughly assessed with the second and third types of approximation on other parallel applications and the lack in the performance should be detected.

The authors of *Fast and High Quality Greedy mapping* [13] have as well proposed two refinements of their algorithm. First one improves a dilation, implementing Kernighan-Lin algorithm which uses task swaps approach. However, the authors claim that this refinement could negatively affect on the maximum congestion, that leads to degradation of the overall performance. Thus, the second proposed refinement tries to improve the maximum congestion with minimal damage of the dilation.

The interesting ideas of the mappings were proposed by Bhatelé in [31]. All heuristics from the article have the same idea: they *"attempt to find the largest possible area of the application grid (in case of 2-D mapping) which overlaps with the processor mesh and maps it one-to-one"* [31], however using

different approaches to reach this goal. It will be valuable to implement these mapping algorithms and validate in HAEC-SIM their performance.

The discussed in Section 2.10 A^* algorithm has been applied in this work, but could not work with the problems of particular sizes due to its high time and space complexity. It may be caused by high tasks connectivity in the graph representation. Even for the problems of small sizes (2-D mesh $2 \times 2 \times 2$ and 8-tasks application were considered) the algorithm has returned not optimal solutions. It should be as well noted, that the selecting of the heuristic function that is not admissible would make algorithm's performance comparable to greedy algorithms, and, similarly, it would return not optimal solution, what could be observed in this case. Nevertheless, the definition of an appropriate heuristic function could solve the problem of an exhaustive search in the state-space.

The static $one - to - one$ mapping function were considered in this thesis. However, it will be interested to evaluate the performance of the mapping algorithms from the present work in case of the "oversubscription" the parallel tasks on the physical nodes. In other words, one can consider the $many - to - one$ mapping function, where several application tasks are processed by one processor.

As stated in [8], an efficient mapping strategy is able to reduce an energy consumption of the parallel HPC application. However, this assumption has not been proved in the present work and should be validated with HAEC-SIM framework and its simulation module **power_estimator**. The detailed explanation could be found in [7],[14].

As mentioned in Section 3.3.2, the *Dimension Order Routing* communication model were used for the experiments on HAEC-SIM. However, two more additional models, supported by HAEC-SIM, can be tested in future: *Practical Network Coding* model and *Macro Flow Data* model. The results of the simulations will differ from the presented ones in this thesis. It

would be valuable to compare as well the impact of the communication model on the parallel application.

Bibliography

[1] *http://insidehpc.com/* Inside HPC site (June 27, 2017).

[2] *https://www.top500.org/* Top500 supercomputers site (June 27, 2017).

[3] *http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Clustering_Parameters/Manhattan_Distance_Metric.htm* Manhattan distance definition (June 27, 2017).

[4] *https://www.nas.nasa.gov/publications/npb.html* NAS Parallel Benchmarks (June 27, 2017).

[5] *http://glaros.dtc.umn.edu/gkhome/metis/metis/overview* METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering site (June 27, 2017).

[6] *http://networkx.readthedocs.io/en/stable/overview.html* NetworkX documentation (June 27, 2017).

[7] *https://tu-dresden.de/zih/forschung/projekte/haec/simulator* Manual for HAEC Simulator Framework (June 27, 2017).

[8] *Hoefler, T., Jeannot, E., Mercier, G.* An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. Edited by Jeannot, E., Žilinskas, J., *High-Performance Computing on Complex Environments*, pages 75-94. Wiley & Sons (2013).

[9] *Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J. C., et al.* The International Exascale Software Project Roadmap. *In-*

ternational Journal of High Performance Computing Applications, 25(1), pages 3-60 (2011).

[10] *Hoefler, T., Snir, M.* Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 75-84. ACM New York, NY, USA (2011).

[11] *Wu, J., Xiong, X., Lan, Z.* Hierarchical Task Mapping for Parallel Applications on Supercomputers. *The Journal of Supercomputing archive, Volume 71, Issue 5*, pages 1776-1802 (2015).

[12] *Agarwal, T., Sharma, A., Kalé, L.,V.* Topology-aware Task Mapping for Reducing Communication Contention on Large Parallel Machines. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, page 10 (2006).

[13] *Deveci, M., Kaya, K., Uçar, B., Çatalyürek, Ü.,V.* Fast and Hign Quality Topoplogy-Aware Task Mapping. *2015 IEEE International Parallel and Distributed Processing Symposium*, Hyderabad, pages 197-206 (2015).

[14] *Bielert, M., Ciorba, F.,M., Feldhoff, K., Ilsche, T., and Nagel, W.,E.* HAEC-SIM: A Simulation Framework for Highly Adaptive Energy-Efficient Computing Platforms. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 129-138 (2015).

[15] *Knüpfer, A., Rössel, C., Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., et al.* Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. Edited by Brunst, H., Müller, M., S., Nagel, W., E., and Resch, M., M., *Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop*

on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden, pages 79-91. Springer Berlin Heidelberg (2012).

[16] *Knüpfer, A., Dietrich, R., Doleschal, J., Geimer, M., Hermanns, M.,A., Rössel, C., et al.* Generic support for remote memory access operations in Score-P and OTF2. In Cheptsov, A., Brinkmann, S., Gracia, J., Resch, M., M., and Nagel, W., E., editors, *Tools for High Performance Computing 2012*, pages 57–74. Springer Berlin Heidelberg (2013).

[17] *Ciorba, F. M., Ilsche, T., Franz, E., Pfennig, S., Scheunert, C., Markwardt, U., Schuchart, J., Hackenberg, D., Schöne, R., Knüpfer, A., Nagel, W.,E., Jorswieck, E.,A., and Müller, M.,S.* Analysis of Parallel Applications on a High Performance–Low Energy Computer. In *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*, pages 474-485. Springer International Publishing (2014).

[18] *Deo, N.* Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall, Inc. Upper Saddle River, NJ, USA (1974).

[19] *Bokhari, S.,H.* On the Mapping Problem. *IEEE Transactions on Computers, Volume C-30, Issue 3*, pages 207-214 (1981).

[20] *Shen C.,C., Tsai W.,H.* A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion. *IEEE Transactions on Computers, Volume C-34, Issue 3*, pages 197-203 (1985).

[21] *Pascual, J.,A., Miguel-Alonso, J., Lozano, J.,A.* Application-aware metrics for partition selection in cube-shaped topologies. In *Parallel Computing, Volume 40, Issues 5–6*, pages 129-139 (2014).

[22] *Pascual, J.,A., Miguel-Alonso, J., Lozano, J.,A.* Optimization-based mapping framework for parallel applications. In *Journal of Parallel and Distributed Computing, Volume 71, Issue 10*, pages 1377-1387 (2011).

[23] *Kafil, M., Ahmad, I.* Optimal Task Assignment in Heterogeneous Computing Systems. In *Proceedings of Heterogeneous Computing Workshop*, HCW '97, Geneva, pages 135-146 (1997).

[24] *Besmer, D.* Process-to-Node Mapping Strategies for the HAEC Box. Bachelor thesis. University of Basel (2016).

[25] *Tuncer, O., Leung, V.,J., Coskun, A.,K.* PaCMap: Topology Mapping of Unstructured Communication Patterns. Proposed for presentation at the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming held February 7-11, 2015 in San Francisco, CA (2014).

[26] *Tuncer, O., Leung, V.,J., Coskun, A.,K.* PaCMap: Topology Mapping of Unstructured Communication Patterns onto Non-contiguous Allocations. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, pages 37-46. ACM New York, NY, USA (2015).

[27] *Glantz, R., Meyerhenke, H., Noe, A.* Algorithms for Mapping Parallel Processes onto Grid and Torus Architectures. In *Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '15, pages 236-243. IEEE Computer Society Washington, DC, USA (2015).

[28] *Wei, C.* Task Partitioning and Mapping Algorithms for Multi-core Packet Processing Systems. Master thesis (2009).

[29] *Schulz, C.* High Quality Graph Partitioning. Dissertation. Karlsruhe Institute of Technology (2013).

[30] *Sanders, P., Schulz, C.* High Quality Graph Partitioning. In *Proc. of the 10th DIMACS Impl. Challenge Workshop: Graph Partitioning and Graph Clustering*, pages 1-17. AMS (2013).

[31] *Bhatelé, A., Gupta, G.,R., Kalé, L.,V., Chung, I.,H.* Automated Mapping of Regular Communication Graphs on Mesh Interconnects. *International Conference on High Performance Computing*, Dona Paula, pages 1-10 (2010).

Appendix A

Simulations with HAEC-SIM. Workflow

SCORE-P and HAEC-SIM commands

Here the list of the commands for SCORE-P and HAEC-SIM frameworks is given. All commands are related to the LU application of the NAS parallel benchmark class C compiled with 64 MPI processes.

1. To create a trace of the parallel application on the "miniHPC" cluster, the application must be compiled and run with `Score-P` [15] flags:

   (a) Specify all needed flags to compile the application in `make.def` file in `/config` subfolder of the NPB benchmark main folder:

      i. `MPIF77 = scorep-mpif77`

      ii. `FMPI_LIB = -L/opt/apps/easybuild/software/OpenMPI/2.0.2-GCC-6.3.0-2.27/lib -lmpi`, or the path where the subfolder `/lib` of `MPICH2` is located on your machine

      iii. `FMPI_INC = -I/opt/apps/easybuild/software/OpenMPI/2.0.2-GCC-6.3.0-2.27/include`, or the path where the subfolder `/include` of `MPICH2` is located on your machine

(b) Compile your application with command `make lu CLASS=C NPROCS=64`. The compiled application now could be found in `/bin` subfolder.

(c) Export all needed global `Score-P` variables to create a profile of the application:

i. `export SCOREP_ENABLE_TRACING=false`

ii. `export SCOREP_ENABLE_PROFILING=true`

iii. `export SCOREP_EXPERIMENT_DIRECTORY=lu_c_64_profile`

(d) Run your application with command: `mpirun -n 64 lu.c.64` from the `/bin` subfolder.

(e) The created profile file could be used to specify which functions and regions of the application will be included in the final trace, how many memory is required to create the trace, and others. To open the profile, one uses the `scorep-score -r <name_of_profile.cubex>` command.

(f) Export additional global `Score-P` variables:

i. `export SCOREP_ENABLE_TRACING=true`

ii. `export SCOREP_ENABLE_PROFILING=false`

iii. `export SCOREP_TOTAL_MEMORY=52MB`

iv. `export SCOREP_FILTERING_FILE=lu_scorep.filt`, if you have specified it

v. `export SCOREP_EXPERIMENT_DIRECTORY=lu_c_64_trace`

(g) Run the application once again with the command `mpirun -n 64 lu.c.64`, and get as a result a trace file of the application.

2. Now one is able to run the HAEC-SIM simulation with the mapping files created on the pre-simulation step. To do so, please, specify a configuration file. The file consists of the target system parameters and is stored as a JSON file [7], [14].

3. To run the simulation, use the command : `mpirun -n 64 haec_sim -o <outputtracefolder>/ -p <positionsmapfile> -c <configurationfile> --module static_network_model -V <level> <inputtracefile.otf2>`

   (a) `-o <outputtracefolder>/`: path to output trace file.

   (b) `-p <positionsmapfile>`: path to mapping file.

   (c) `-c <configurationfile>`: path to configuration file.

   (d) `-V <level>`: verbosity of the messages during the simulation, where level is one of the followings: `trace`, `debug`, `info` [default], `warn`, `error`, or `fatal` [7].

4. The specified above command could be run on a KNL Xeon Phi node, that is able to run 64 threads simultaneously. If one wants to run the command with Xeon nodes, one need to specify a machinefile and submit the script file to the scheduler.

5. To get a statistics file for the chosen trace file, one has to run the following command: `mpirun -n 64 haec_sim -c <configurationfile> --module trace_stats <inputtracefile.otf2>`

HAEC-SIM configuration files

- **connections-MESH.conf**

```
{"platform": {
    "link_types":
    [ { "name" : "infiniband",
        "bandwidth" : 9375000000,
        "latency" : 1270,
        "bit_error_rate": 1e-12,
        "packet_attack_rate": 0.01,
        "idle_power": 1,
        "active_power": 0 } ],
    "topology": {
        "shape": [ 4, 4, 4 ],
        "link_types": [ "infiniband", "infiniband", "infiniband" ],
        "connectivity": [ "mesh", "mesh", "mesh" ] },
    "nodes": {
      "computing": {
        "power": "MIPS",
        "cores": 16,
        "usage": [ "exclusive", "shared" ] }
             }
    },
 "modules":
   { "cpu_resource_model": {},
     "static_network_model": {
       "routing": "shortest_path",
       "parameter_search_folders": ["share/modules/
                  static_network_model/parameters"],
       "communication_model": "DOR",
       "size_packet": 1500,
       "size_finitefield": 8,
       "size_window": 5,
       "delay_processing": 100,
       "delay_mpi": 500,
       "size_windowid": 4,
       "size_packetid": 2,
       "size_signature": 256,
       "size_generationid": 4 },
     "metric_reader": {
       "output": "channel.csv",
       "channel": "PAPI_FP_OPS" },
     "power_estimator": {
       "model": "foobar42" },
     "phase_profile": {
       "output": "output.txt",
       "channels":
       [ "localhost/watts",
         "apollo/s0/core/watts",
         "apollo/s1/core/watts",
         "apollo/s3/core/watts",
         "apollo/s0/ram/watts",
         "apollo/s1/ram/watts",
         "apollo/s3/ram/watts",
         "apollo/s0/nb/watts",
         "apollo/s1/nb/watts",
         "apollo/s3/nb/watts" ] },
     "comm_matrix": {
      "output": "comm_matrix" }
   }
}
```

- **connections-TORUS.conf**

```
{"platform": {
    "link_types":
    [ { "name" : "infiniband",
        "bandwidth" : 9375000000,
        "latency" : 1270,
        "bit_error_rate": 1e-12,
        "packet_attack_rate": 0.01,
        "idle_power": 1,
        "active_power": 0 } ],
    "topology": {
        "shape": [ 4, 4, 4 ],
        "link_types": [ "infiniband", "infiniband", "infiniband" ],
        "connectivity": [ "torus", "torus", "torus" ] },
    "nodes": {
      "computing": {
        "power": "MIPS",
        "cores": 16,
        "usage": [ "exclusive", "shared" ] }
             }
    },
 "modules":
   { "cpu_resource_model": {},
     "static_network_model": {
       "routing": "shortest_path",
       "parameter_search_folders": ["share/modules/
                  static_network_model/parameters"],
       "communication_model": "DOR",
       "size_packet": 1500,
       "size_finitefield": 8,
       "size_window": 5,
       "delay_processing": 100,
       "delay_mpi": 500,
       "size_windowid": 4,
       "size_packetid": 2,
       "size_signature": 256,
       "size_generationid": 4 },
     "metric_reader": {
       "output": "channel.csv",
       "channel": "PAPI_FP_OPS" },
     "power_estimator": {
       "model": "foobar42" },
     "phase_profile": {
       "output": "output.txt",
       "channels":
       [ "localhost/watts",
         "apollo/s0/core/watts",
         "apollo/s1/core/watts",
         "apollo/s3/core/watts",
         "apollo/s0/ram/watts",
         "apollo/s1/ram/watts",
         "apollo/s3/ram/watts",
         "apollo/s0/nb/watts",
         "apollo/s1/nb/watts",
         "apollo/s3/nb/watts" ] },
     "comm_matrix": {
      "output": "comm_matrix" }
   }
}
```

- **connections-HAEC-BOX.conf**

```
{"platform": {
    "link_types":
    [ { "name" : "optical",
        "bandwidth" : 1875000000000,
        "latency" : 10,
        "bit_error_rate": 1.0e-12,
        "packet_attack_rate": 0.01,
        "idle_power": 1,
        "active_power": 0 },
      { "name" : "wireless",
        "bandwidth" : 125000000000,
        "latency" : 100,
        "bit_error_rate": 1.0e-8,
        "packet_attack_rate": 0.01,
        "idle_power": 1,
        "active_power": 0 } ],
    "topology": {
        "shape": [ 4, 4, 4 ],
        "link_types": [ "optical", "optical", "wireless" ],
        "connectivity": [ "torus", "torus", "crossbar" ] },
    "nodes": {
      "computing": {
        "power": "MIPS",
        "cores": 16,
        "usage": [ "exclusive", "shared" ] }
            }
    },
 "modules":
   { "cpu_resource_model": {},
     "static_network_model": {
       "routing": "haec_box",
       "parameter_search_folders": ["share/modules/
                  static_network_model/parameters"],
       "communication_model": "DOR",
       "size_packet": 1500,
       "size_finitefield": 8,
       "size_window": 5,
       "delay_processing": 100,
       "delay_mpi": 5,
       "size_windowid": 4,
       "size_packetid": 2,
       "size_signature": 256,
       "size_generationid": 4 },
     "metric_reader": {
       "output": "channel.csv",
       "channel": "PAPI_FP_OPS" },
     "power_estimator": {
       "model": "foobar42" },
     "phase_profile": {
       "output": "output.txt",
       "channels":
       [ "localhost/watts",
         "apollo/s0/core/watts",
         "apollo/s1/core/watts",
         "apollo/s3/core/watts",
         "apollo/s0/ram/watts",
         "apollo/s1/ram/watts",
         "apollo/s3/ram/watts",
         "apollo/s0/nb/watts",
         "apollo/s1/nb/watts",
         "apollo/s3/nb/watts" ] },
     "comm_matrix": {
      "output": "comm_matrix" }
   }
}
```

Appendix B

Numerical Values

Mapping Statistics

The following tables represent the numerical values of the mapping algorithms, discussed in the present thesis. In each table one can find the name of NAS benchmark parallel application, the underlying 3-D topology, the name of the algorithm, the total, average, minimum and maximum values of the "Inter-node logical communications" metric (IeNLC), the total, average, minimum and maximum values of the "Inter-node physical communications" metric (IeNPC), the default (from a consecutive $1:1$ mapping, as proposed in [21] and discussed in Section 4.1) and total values of the "Weighted task average distance" metric (WTAD).

LU.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					14225946	88913	80829	242488		6947
MinMD					18994893	141101	80829	969951		9277
Bipartition					18024923	130766	80829	727465		8803
PaCMap					14630085	125613	80829	484975		7145
TopoAware	9052876	80830	80829	80830	12609362	108130	80829	404145	8527	6158
Greedy					25057044	216232	80829	808292		12237
GreedyALLC					10992778	88178	80829	161659		5368
FHGreedy					20692274	143289	80829	808291		10106
UDFS					19075674	140684	80829	646632		9316

LU.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					12609356	94100	80829	242487		7816
MinMD					18509915	117152	80829	242490		11566
Bipartition					17054971	120106	80829	323319		11961
PaCMap					13983449	113687	80829	323317		7145
TopoAware	9052876	80830	80829	80830	12124382	102750	80829	242488	8527	7974
Greedy					22632174	141452	80829	484974		12790
GreedyALLC					9052876	80830	80829	80830		6316
FHGreedy					16812480	115154	80829	323318		10737
UDFS					16650804	120659	80829	404145		10658

LU.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					9214534	80830	80829	80830		14685
MinMD					13660147	96881	80829	323316		12001
Bipartition					10750291	87401	80829	323316		10777
PaCMap					11235261	91344	80829	161659		7026
TopoAware	9052876	80830	80829	80830	9133705	80830	80829	80830	8527	11329
Greedy					13983468	98476	80829	404145		13264
GreedyALLC					9295364	82995	80829	161660		10856
FHGreedy					10588631	88981	80829	242487		10303
UDFS					13255986	93353	80829	242487		11606

Table B.1: Statistics for LU.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

BT.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					1293646	7682	3218	51488		631
MinMD					1570384	9308	3218	51488		766
Bipartition					1576820	9738	3218	77232		770
PaCMap					1255020	8201	3218	38616		612
TopoAware	617856	3218	3218	3218	1171352	7519	3218	38616	880	572
Greedy					1872876	12385	3218	86886		914
GreedyALLC					1209968	7548	3218	41834		590
FHGreedy					1589692	8829	3218	64360		776
UDFS					1879312	9595	3218	70796		917

BT.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					1029760	5987	3218	19308		722
MinMD					1390176	6917	3218	19308		892
Bipartition					1357996	7073	3218	22526		917
PaCMap					1113428	6327	3218	19308		612
TopoAware	617856	3218	3218	3218	1055504	6032	3218	19308	880	568
Greedy					1576820	7301	3218	22526		905
GreedyALLC					1094120	6592	3218	25744		628
FHGreedy					1370868	6889	3218	22526		889
UDFS					1551076	8037	3218	38616		943

BT.C.64	**IePLC**				**IeNPC**				**WTAD**	
$4 \times 4 \times 4$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					685434	3445	3218	9654		876
MinMD					920348	4383	3218	16090		870
Bipartition					878514	3958	3218	12872		880
PaCMap					881732	4819	3218	16090		612
TopoAware	617856	3218	3218	3218	740140	3559	3218	6436	880	854
Greedy					933220	4321	3218	16090		946
GreedyALLC					762666	3776	3218	12872		864
FHGreedy					936438	4524	3218	12872		873
UDFS					1020106	4701	3218	16090		980

Table B.2: Statistics for BT.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

LU.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					112837457	166427	80829	1859070		2881
MinMD					185987964	275538	80829	3475648		3820
Bipartition					177177419	227151	80829	3475652		2915
PaCMap					113564840	171549	80829	1293264		2168
TopoAware	38797980	80830	80829	80830	107826005	158103	80829	1293265	2644	1822
Greedy					257764006	567763	80829	6951295		4637
GreedyALLC					126820803	187883	80829	1697410		2276
FHGreedy					162870715	242368	80829	4849745		3095
UDFS					163517132	268061	80829	5900517		4366

LU.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					85517198	120788	80829	404145		3628
MinMD					100389820	144446	80829	484976		4635
Bipartition					85921361	124344	80829	404145		4050
PaCMap					68623896	125685	80829	646632		2168
TopoAware	38797980	80830	80829	80830	68543081	122181	80829	484974	2644	2540
Greedy					125770042	169502	80829	565803		5111
GreedyALLC					63046690	111588	80829	323318		2299
FHGreedy					94085104	135180	80829	484974		3742
UDFS					112109885	157017	80829	565803		4504

LU.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					40252904	81156	80829	161658		6416
MinMD					67573189	110958	80829	484974		4598
Bipartition					53670536	93179	80829	323316		5293
PaCMap					52134782	99494	80829	323318		2074
TopoAware	38797980	80830	80829	80830	40414561	82987	80829	242487	2644	5296
Greedy					79778340	136374	80829	404145		5303
GreedyALLC					43405240	87335	80829	242487		5764
FHGreedy					75009404	126920	80829	484975		4400
UDFS					65067405	114556	80829	404145		4894

Table B.3: Statistics for LU.C.256 on 3-D Mesh, 3-D Torus, and HAEC Box

BT.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					20131986	22672	6434	411776		463
MinMD					27801314	36106	6434	591928		482
Bipartition					23143098	37268	6434	341002		567
PaCMap					21470258	25469	6434	302398		337
TopoAware	4941312	6434	6434	6434	23078758	26108	6434	295964	521	307
Greedy					29319738	46912	6434	772080		709
GreedyALLC					21277238	24373	6434	295964		342
FHGreedy					26617458	33066	6434	701306		492
UDFS					27724106	38721	6434	643400		567

BT.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					10628968	13558	6434	51472		464
MinMD					13704420	15717	6434	70774		602
Bipartition					14302782	19513	6434	83642		567
PaCMap					10275098	15359	6434	57906		337
TopoAware	4941312	6434	6434	6434	9599528	13483	6434	45038	521	327
Greedy					19192622	21663	6434	77208		748
GreedyALLC					10371608	14506	6434	57906		376
FHGreedy					13800930	16161	6434	77208		550
UDFS					15203542	18165	6434	96510		644

BT.C.256	**IePLC**				**IeNPC**				**WTAD**	
$8 \times 8 \times 4$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					5784166	6789	6434	19302		775
MinMD					8923958	11240	6434	51472		665
Bipartition					11439652	16437	6434	77208		567
PaCMap					7759404	10444	6434	51472		350
TopoAware	4941312	6434	6434	6434	5983620	7132	6434	32170	521	653
Greedy					12359714	14339	6434	70774		757
GreedyALLC					5951450	7188	6434	38604		666
FHGreedy					10390910	13054	6434	64340		666
UDFS					9052638	10973	6434	57906		691

Table B.4: Statistics for BT.C.256 on 3-D Mesh, 3-D Torus, and HAEC Box

LU.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					595548529	225076	80829	4122286		774
MinMD					1072117023	400492	80829	27239392		1248
Bipartition					1022891623	551127	80829	5658035		1455
PaCMap					900758650	304311	80829	6627978		673
TopoAware	160364860	80830	80829	80830	236020849	97369	80829	323318	705	335
Greedy					1639859871	920236	80829	45021756		1772
GreedyALLC					842804232	292641	80829	6627978		763
FHGreedy					1093213077	399567	80829	30149241		1009
UDFS					989832065	418357	80829	28209321		1452

LU.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					336329731	129259	80829	323317		871
MinMD					492491514	161314	80829	727461		1484
Bipartition					559822037	257153	80829	889119		1455
PaCMap					342310983	141393	80829	646632		673
TopoAware	160364860	80830	80829	80830	333904816	134694	80829	565803	705	858
Greedy					679448820	224019	80829	1050777		1744
GreedyALLC					332611496	143244	80829	646632		696
FHGreedy					444317336	148552	80829	1050777		1139
UDFS					591345098	189292	80829	969948		1595

LU.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					211206351	85302	80829	242488		2643
MinMD					250650977	102432	80829	565803		1704
Bipartition					503403339	236229	80829	808291		1455
PaCMap					248306842	108007	80829	404145		627
TopoAware	160364860	80830	80829	80830	165376267	82689	80829	323316	705	2259
Greedy					389919341	167492	80829	808290		1881
GreedyALLC					172650878	85092	80829	404147		2180
FHGreedy					391050904	149542	80829	889119		1580
UDFS					382482954	131483	80829	808290		1781

Table B.5: Statistics for LU.C.1024 on 3-D Mesh, 3-D Torus, and HAEC Box

BT.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					245817796	75497	12866	5197864		268
MinMD					336857612	109334	12866	7295022		305
Bipartition					275370998	132518	12866	2676128		388
PaCMap					259790272	78274	12866	2303014		183
TopoAware	39524352	12866	12866	12866	260742356	75842	12866	2830520	268	160
Greedy					366423680	167700	12866	9675232		503
GreedyALLC					260201984	78045	12866	2303014		189
FHGreedy					322885136	99411	12866	8929004		279
UDFS					319076800	118484	12866	8440096		413

BT.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					92712396	27943	12866	154392		273
MinMD					133626276	36751	12866	205856		398
Bipartition					156682148	58333	12866	244454		388
PaCMap					89856144	32640	12866	154392		183
TopoAware	39524352	12866	12866	12866	83899186	28832	12866	205856	268	165
Greedy					191832060	56772	12866	347382		499
GreedyALLC					88479482	31828	12866	154392		189
FHGreedy					127244740	35093	12866	231588		373
UDFS					166846288	47293	12866	257320		466

BT.C.1024	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 8 \times 8$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					67649428	16622	12866	64330		341
MinMD					72860158	19940	12866	154392		470
Bipartition					130293982	49712	12866	205856		388
PaCMap					66813138	21707	12866	141526		194
TopoAware	39524352	12866	12866	12866	48131706	14291	12866	102928	268	473
Greedy					119100562	36681	12866	257320		534
GreedyALLC					51078020	14437	12866	90062		482
FHGreedy					113465254	34259	12866	308784		482
UDFS					118264272	31929	12866	218722		505

Table B.6: Statistics for BT.C.1024 on 3-D Mesh, 3-D Torus, and HAEC Box

LU.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					19380429440	1681016	245021	440057738		1214
Bipartition					14855869613	1889099	245021	73506311		1128
PaCMap					14778687429	1146969	245021	99233507		556
TopoAware	1975849596	245022	245021	245022	*	*	*	*	925	*
Greedy					26942756560	3763481	245021	481466272		1499
GreedyALLC					14092873326	1105411	245021	97028318		577
FHGreedy					18338598678	1515712	245021	417760838		892
UDFS					18216086508	1853301	245021	518219415		1550

LU.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					8404956438	586284	245021	4900420		1565
Bipartition					7350385784	737251	245021	3185273		1128
PaCMap					4462567878	453421	245021	3185273		556
TopoAware	1975849596	245022	245021	245022	*	*	*	*	925	*
Greedy					10216886160	815460	245021	7595653		1749
GreedyALLC					4374360289	444008	245021	2205189		574
FHGreedy					6908367878	521190	245021	3430294		1226
UDFS					9575175921	691449	245021	5390462		1650

LU.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					3558685601	319796	245021	2205191		1677
Bipartition					6574159166	706519	245021	2940252		1128
PaCMap					3443035424	366671	245021	1960168		556
TopoAware	1975849596	245022	245021	245022	*	*	*	*	925	*
Greedy					5418639916	549057	245021	8575735		1774
GreedyALLC					2104240619	257180	245021	1470127		2523
FHGreedy					5588684537	485129	245021	2940253		1439
UDFS					6046138454	422689	245021	2940252		1756

Table B.7: Statistics for LU.D.4096 on 3-D Mesh, 3-D Torus, and HAEC Box

BT.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ 3-D Mesh	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					4715463060	358700	32130	96936210		242
Bipartition					3238157790	357098	32130	21655620		255
PaCMap					3776399550	253365	32130	33126030		122
TopoAware	394813440	32130	32130	32130	*	*	*	*	263	*
Greedy					4705085070	548123	32130	111973050		446
GreedyALLC					3734116470	254594	32130	31455270		121
FHGreedy					4802310450	358783	32130	101562930		223
UDFS					4633467300	399231	32130	113386770		332

BT.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ 3-D Torus	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					1623593160	101178	32130	642600		305
Bipartition					1661185260	140517	32130	642600		255
PaCMap					944686260	80571	32130	546210		122
TopoAware	394813440	32130	32130	32130	*	*	*	*	263	*
Greedy					2518413660	174624	32130	1574370		420
GreedyALLC					924155190	79655	32130	449820		121
FHGreedy					1538769960	103978	32130	867510		282
UDFS					2070296550	138993	32130	1156680		363

BT.D.4096	**IePLC**				**IeNPC**				**WTAD**	
$16 \times 16 \times 16$ HAEC	total	avg	min	max	total	avg	min	max	default	total
Mapper					*	*	*	*		*
MinMD					770316750	47989	32130	385560		400
Bipartition					1436339520	126073	32130	642600		255
PaCMap					661717350	53035	32130	514080		122
TopoAware	394813440	32130	32130	32130	*	*	*	*	263	*
Greedy					1498928760	109379	32130	1285200		443
GreedyALLC					527446080	37215	32130	289170		366
FHGreedy					1366103340	92888	32130	867510		396
UDFS					1388979900	86703	32130	674730		391

Table B.8: Statistics for BT.D.4096 on 3-D Mesh, 3-D Torus, and HAEC Box

HAEC-SIM statistics

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	2.42857	1.57143	1.7619	1.78571	1.28571	4.14286
336 B	2.42857	1.57143	1.14286	1.14286	1.28571	1.71429
352 B	1.42857	1.57143	1.69048	1.21429	1.14286	1.04762
368 B	1.42857	1.57143	1.42857	1	1.14286	1
760 B	1.92857	1.5	1.57143	1.64286	1.21429	2.42857
800 B	1.92857	1.57143	1.63095	1.40476	1.21429	2.38095
840 B	1.92857	1.64286	1.57143	1.07143	1.21429	1.57143
1296 B	1.92857	1.5	1.28571	1.35714	1.21429	1.78571
259200 B	1.92857	1.57143	1.72619	1.5	1.21429	2.59524
272160 B	1.92857	1.57143	1.28571	1.07143	1.21429	1.35714
Average # of hops	1.92857	1.57143	1.61607	1.39286	1.21429	2.28571

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	2.42857	1.61905	1.71429	1.42857	1	2.85714
336 B	2.42857	1	1.14286	1.14286	1	2
352 B	1.14286	1.42857	1.54762	1.30952	1	1.09524
368 B	1.14286	1	1.42857	1.35714	1	1
760 B	1.78571	1.28571	1.28571	1.21429	1	2.21429
800 B	1.78571	1.47619	1.58333	1.34524	1	1.80952
840 B	1.78571	1	1.57143	1.42857	1	1.78571
1296 B	1.78571	1.14286	1.14286	1.14286	1	1.71429
259200 B	1.78571	1.52381	1.63095	1.36905	1	1.97619
272160 B	1.78571	1	1.28571	1.25	1	1.5
Average # of hops	1.78571	1.39286	1.54464	1.33929	1	1.85714

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	1.57143	1.04762	1.19048	1.02381	1.04762	1.42857
336 B	1.57143	1	1.14286	1	1	1.07143
352 B	1.14286	1	1.35714	1	1.02381	1
368 B	1.14286	1	1.14286	1	1	1
760 B	1.35714	1	1.07143	1	1.07143	1.28571
800 B	1.35714	1.02381	1.27381	1.0119	1.02381	1.16667
840 B	1.35714	1	1.21429	1	1	1.07143
1296 B	1.35714	1	1.07143	1	1.03571	1.14286
259200 B	1.35714	1.02381	1.27381	1.0119	1.03571	1.21429
272160 B	1.35714	1	1.14286	1	1	1.03571
Average # of hops	1.35714	1.01786	1.24107	1.00893	1.02679	1.16964

Table B.9: Number of hops for LU.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	2.8416e+07	3.54255e+07	3.43011e+07	3.43834e+07	3.92693e+07	1.96162e+07
336 B	2.33307e+07	2.91098e+07	3.32248e+07	3.32248e+07	3.17297e+07	2.79557e+07
352 B	5.66855e+07	5.26456e+07	4.83485e+07	6.40592e+07	6.69627e+07	6.23203e+07
368 B	5.92621e+07	5.50386e+07	5.66969e+07	7.69852e+07	7.00064e+07	5.3931e+07
760 B	1.55783e+07	1.64738e+07	1.6588e+07	1.58285e+07	1.62078e+07	1.72861e+07
800 B	2.64135e+07	2.56196e+07	2.81024e+07	2.74979e+07	2.82803e+07	2.71471e+07
840 B	3.49994e+07	3.32635e+07	3.84633e+07	3.87937e+07	3.68237e+07	3.63288e+07
1296 B	1.06156e+08	1.19074e+08	1.28412e+08	1.26914e+08	1.30502e+08	1.10201e+08
259200 B	2.82271e+08	2.92844e+08	2.91513e+08	3.10829e+08	3.22158e+08	2.72656e+08
272160 B	2.79698e+08	2.92286e+08	3.09571e+08	3.24981e+08	3.18208e+08	3.0329e+08
Average bandwidth	9.13e+07	9.52e+07	9.85e+07	1.05e+08	1.06e+08	9.31e+07

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	2.8416e+07	3.6273e+07	3.43011e+07	3.6289e+07	4.29136e+07	2.59565e+07
336 B	2.33307e+07	3.48678e+07	3.32248e+07	3.32248e+07	3.48678e+07	2.59018e+07
352 B	6.69627e+07	5.22433e+07	5.17346e+07	5.93347e+07	7.3638e+07	6.90491e+07
368 B	7.00064e+07	5.41949e+07	5.66969e+07	5.74397e+07	7.69852e+07	7.69852e+07
760 B	1.54166e+07	1.72425e+07	1.67387e+07	1.75571e+07	1.69012e+07	1.51023e+07
800 B	2.65957e+07	2.76633e+07	2.82543e+07	2.93551e+07	2.93461e+07	2.62058e+07
840 B	3.55623e+07	4.20901e+07	3.82133e+07	3.6025e+07	3.78734e+07	3.204e+07
1296 B	1.10468e+08	1.34776e+08	1.34776e+08	1.36466e+08	1.40623e+08	1.1248e+08
259200 B	2.91327e+08	3.09082e+08	2.97196e+08	3.13904e+08	3.30504e+08	2.82458e+08
272160 B	2.87733e+08	3.29948e+08	3.102e+08	3.1729e+08	3.26419e+08	2.99388e+08
Average bandwidth	9.56e+07	1.04e+08	1.00e+08	1.04e+08	1.11e+08	9.66e+07

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
320 B	6.56163e+07	6.47108e+07	6.5108e+07	6.58101e+07	6.54668e+07	6.54781e+07
336 B	6.35446e+07	6.27894e+07	6.39094e+07	6.37e+07	6.30146e+07	6.25705e+07
352 B	2.31183e+08	2.21979e+08	2.27651e+08	2.22121e+08	2.22121e+08	2.27564e+08
368 B	1.85337e+08	1.68432e+08	1.78462e+08	1.85337e+08	1.86091e+08	1.77136e+08
760 B	2.02305e+07	2.01382e+07	2.03035e+07	1.99984e+07	2.01123e+07	2.0286e+07
800 B	3.58649e+07	3.56318e+07	3.59215e+07	3.54428e+07	3.55831e+07	3.59843e+07
840 B	4.79491e+07	4.73682e+07	4.79743e+07	4.74855e+07	4.75586e+07	4.76215e+07
1296 B	2.15783e+08	2.13936e+08	2.15296e+08	2.15638e+08	2.15982e+08	2.14922e+08
259200 B	6.7419e+08	6.4639e+08	6.68838e+08	6.63643e+08	6.63158e+08	6.67602e+08
272160 B	6.56742e+08	6.28817e+08	6.44329e+08	6.61583e+08	6.39131e+08	6.289e+08
Average bandwidth	2.20e+08	2.11e+08	2.17e+08	2.18e+08	2.16e+08	2.15e+08

Table B.10: Bandwidth (in B/s) for LU.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	2.91667	2.09375	2.03125	1.89583	1.95833	2.57292
105840 B	2.91667	2.09375	2.03125	1.89583	1.95833	2.57292
227200 B	3.5	3	1.83333	1.66667	1.83333	3.33333
227280 B	3.5	2.16667	2	1.83333	2.16667	4
228800 B	3.1	2.13333	2	1.61667	2.15	2.6
228880 B	3.16667	2.41667	2.16667	1.5	2.08333	2
230400 B	2.73333	1.93333	2.26667	2.05	1.88333	2.9
230480 B	2.7	1.9	1.76667	2.33333	1.73333	2.2
230560 B	2.72222	2.27778	1.77778	1.94444	1.83333	1.66667
Average # of hops	2.91667	2.09375	2.03125	1.89583	1.95833	2.57292

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	2.33333	1.66667	1.80208	1.70833	1.77083	2.21875
105840 B	2.33333	1.66667	1.80208	1.70833	1.77083	2.21875
227200 B	2.5	1.66667	1.83333	2	1.66667	1.66667
227280 B	2.5	2	2	1.5	2.16667	1.66667
228800 B	2.4	1.7	1.76667	1.68333	1.75	2.31667
228880 B	2.33333	1.41667	1.83333	1.58333	1.91667	1.91667
230400 B	2.26667	1.73333	1.86667	1.68333	1.71667	2.45
230480 B	2.3	1.7	1.7	1.76667	1.76667	2.1
230560 B	2.27778	1.33333	1.77778	1.83333	1.83333	1.88889
Average # of hops	2.33333	1.66667	1.80208	1.70833	1.77083	2.21875

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	1.70833	1.10938	1.42708	1.19792	1.23438	1.51562
105840 B	1.70833	1.10938	1.42708	1.19792	1.23438	1.51562
227200 B	2.33333	1	1.16667	1	1.16667	1.33333
227280 B	2.33333	1	1.16667	1	1.5	1.33333
228800 B	1.93333	1.16667	1.4	1.13333	1.35	1.73333
228880 B	1.83333	1	1.33333	1.08333	1.33333	1.66667
230400 B	1.5	1.03333	1.55	1.28333	1.16667	1.48333
230480 B	1.46667	1.16667	1.36667	1.23333	1.16667	1.46667
230560 B	1.55556	1.22222	1.27778	1.27778	1.05556	1
Average # of hops	1.70833	1.10938	1.42708	1.19792	1.23438	1.51562

Table B.11: Number of hops for BT.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	5.0235e+07	4.98413e+07	5.15332e+07	5.30337e+07	5.12809e+07	5.22724e+07
105840 B	8.89292e+07	9.51563e+07	9.44781e+07	9.32147e+07	9.54262e+07	9.35432e+07
227200 B	2.26529e+08	2.45026e+08	2.72062e+08	2.30204e+08	2.79915e+08	1.93251e+08
227280 B	2.20593e+08	3.08023e+08	2.54399e+08	2.17308e+08	2.59378e+08	1.78016e+08
228800 B	2.2308e+08	2.43863e+08	2.45586e+08	2.2229e+08	2.22691e+08	2.0735e+08
228880 B	2.08495e+08	2.33625e+08	2.04029e+08	2.20493e+08	1.91244e+08	2.09129e+08
230400 B	2.16794e+08	2.06706e+08	2.11569e+08	2.06523e+08	1.87051e+08	2.04386e+08
230480 B	2.21167e+08	2.02709e+08	2.33939e+08	2.00336e+08	1.98162e+08	2.20752e+08
230560 B	2.18256e+08	1.99691e+08	2.24637e+08	2.07998e+08	1.90345e+08	2.29611e+08
Average bandwidth	1.86e+08	1.98e+08	1.99e+08	1.83e+08	1.86e+08	1.76e+08

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	5.23643e+07	5.30749e+07	5.2122e+07	5.41215e+07	5.20939e+07	5.29419e+07
105840 B	9.16702e+07	9.56017e+07	9.3999e+07	9.32396e+07	9.41448e+07	9.49204e+07
227200 B	2.48936e+08	2.34047e+08	2.67843e+08	2.3626e+08	2.31675e+08	2.67474e+08
227280 B	2.39105e+08	2.17395e+08	2.50976e+08	2.39568e+08	2.13215e+08	2.59237e+08
228800 B	2.3363e+08	2.29103e+08	2.4656e+08	2.32632e+08	2.36924e+08	2.24927e+08
228880 B	2.17799e+08	2.29774e+08	2.22611e+08	2.17558e+08	2.12498e+08	2.28194e+08
230400 B	2.15894e+08	2.22359e+08	2.20859e+08	2.18944e+08	2.41959e+08	2.04206e+08
230480 B	2.18364e+08	2.22934e+08	2.2937e+08	2.19639e+08	2.47266e+08	2.09759e+08
230560 B	2.16729e+08	2.30281e+08	2.24154e+08	2.15657e+08	2.42271e+08	2.14226e+08
Average bandwidth	1.93e+08	1.93e+08	2.01e+08	1.92e+08	1.97e+08	1.95e+08

Message Size Groups \ Mapping Algorithm	XYZ-default	Mapper	PaCMap	TopoAware	GreedyALLC	FHGreedy
17640 B	5.97684e+07	5.9687e+07	5.9839e+07	5.97273e+07	5.97439e+07	5.97104e+07
105840 B	1.13871e+08	1.13416e+08	1.13886e+08	1.13434e+08	1.1345e+08	1.13521e+08
227200 B	4.58234e+08	4.5049e+08	4.65238e+08	4.49946e+08	4.50552e+08	4.65913e+08
227280 B	4.25073e+08	4.21126e+08	4.2909e+08	4.21837e+08	4.18149e+08	4.29616e+08
228800 B	4.13177e+08	4.0639e+08	4.13452e+08	4.08383e+08	4.08965e+08	4.14914e+08
228880 B	3.77214e+08	3.73587e+08	3.75778e+08	3.78443e+08	3.73493e+08	3.75135e+08
230400 B	3.7113e+08	3.65116e+08	3.69848e+08	3.68964e+08	3.69607e+08	3.70215e+08
230480 B	3.73477e+08	3.66417e+08	3.72631e+08	3.70137e+08	3.71195e+08	3.73479e+08
230560 B	3.71859e+08	3.64445e+08	3.70761e+08	3.69204e+08	3.69587e+08	3.70499e+08
Average bandwidth	3.29e+08	3.25e+08	3.30e+08	3.27e+08	3.26e+08	3.30e+08

Table B.12: Bandwidth in (B/s) for BT.C.64 on 3-D Mesh, 3-D Torus, and HAEC Box

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author—Autor
Viacheslav Sharunov

Matriculation number—Matrikelnummer
15-059-322

Title of work—Titel der Arbeit
Optimized parallel tasks to nodes mapping in 3-D high performance interconnection topologies

Type of work—Typ der Arbeit
Master Thesis

Declaration—Erklärung
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.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, June 30, 2017

Signature—Unterschrift