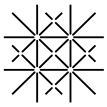


University of Basel

Process-to-Node Mapping Strategies for the HAEC Box

Bachelor Thesis

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

Examiner: Prof. Dr. Florina M. Ciorba
Supervisor: Prof. Dr. Florina M. Ciorba

Daniel Besmer
daniel.besmer@unibas.ch
2009-926-262

June 30, 2016

Acknowledgments

Foremost, I would like to express my gratitude to Prof. Dr. Florina M. Ciorba for offering me the topic for this thesis and her encouraging supervision of the work. I would like to thank Thomas Ilsche for providing me with the communication traces for the NAS parallel benchmarks and the support with the HAEC simulator. Furthermore I would like to thank my friends and especially my family for all the motivation during this time. Without their support this study would hardly have been completed. A special thanks goes to my dear friend Tim Allen who has proofread this thesis.

Abstract

Today's High-Performance Computing (HPC) systems are featuring millions of cores to cope with the increasing demand for computing power in science and technology [1]. While computation time of an application can be reduced by massive parallelization, inter-process communication has become the bottleneck for scaling of parallel applications [2]. In fact, a large portion of the energy consumption in current HPC systems and data centers are related to the transfer of information [1, 3]. To reduce the overall execution time and therefore the energy consumption, parallel applications have to be laid out carefully regarding the inter-process communication patterns and the interconnecting network of the target platform.

Researchers at the Technische Universität Dresden founded the HAEC project to address the problem of high energy dissipation of current systems by developing a new embedded computing architecture [4]. The focus of this thesis lies on the implementation and evaluation of different processes-to-node mapping strategies for the HAEC platform in order to reduce latencies related to communication and therefore increase the energy efficiency of the framework.

Table of Contents

1 Introduction

Today's steadily increasing demand for computing power in science and technology leads to the development of High-Performance Computing (HPC) systems with ten thousands of nodes [5] featuring millions of cores.[1] Since the current trends in processor and architecture design are towards increasing the numbers of computing cores on a chip [1], this number is not very likely to diminish in the near future.

As the diameter of a parallel system grows, communication over the network between computation units (e.g. processes or threads) becomes a non-negligible factor affecting the overall performance of the parallel application [1, 2, 6]. While computation becomes manageable by distributing the workload among thousands of processes, inter-process communication turns into the bottleneck for scaling of large parallel applications [2]. In order to reduce latencies related to communication, the communication pattern of the computing units (virtual topology) as well as the topology of the underlying system (physical topology) has to be taken into account. By assigning processes on the target platform in favor for data locality network contention can be minimized and is therefore resulting in a shortened execution time of the parallel application [1, 2].

The desire to reduce the time-to-market of a product in industry or towards refined models for longer and finer-grained simulations in science [1] comes with a high price. As HPC systems are getting bigger and more powerful, the energy consumption for operation and cooling becomes a major issue [4]. In fact energy efficiency is one of the main problems we are faced with in information and communication technology [1, 3]. A group of researchers at the Technische Universität Dresden founded the collaborative research center HAEC (Highly Adaptive Energy-Efficient Computing) to address this problem. Funded by the German Research Foundation (DFG) the project's aim is to develop a novel computing platform called the HAEC Box with the objective to reduce the high energy consumption of current architectures [4]. The HAEC Box is an embedded parallel computing framework consisting of a hardware and software layer. It provides a platform for running parallel

[1] China's Supercomputer Tianhe-2 consists of $3'100'000$ cores (http://www.top500.org).

applications while dynamically adapting its energy usage based on the workload [3, 4].

As mentioned above, since the high energy consumption of current systems can be assigned to a large portion to the transfer of information, a strategy for process-to-node assignment in favor for data locality should contribute to the goal of energy efficiency in the HAEC Box. Therefore the aim of this thesis is to implement, adapt and evaluate different mapping strategies - proposed in literature or what we have developed ourselves - advancing the HAEC Box for maximization of parallelism and reduction of the overall communication cost. The process-to-node mappings generated by the different mapping strategies then can be used in the HAEC simulator for verification of increased energy efficiency due consideration of communication patterns in parallel applications.

The work is organized as follows: Chapter 2 addresses the problem of mapping processes to nodes in the context of the HAEC framework. Different definitions of the mapping problem found in literature are discussed in common with the general setup. Chapter 3 gives an overview of the selected mapping strategies and their characteristics. The algorithmic performance with respect to the mapping quality as well as the run-time for the different strategies are listed in Chapter 4. Conclusions from the results and future work are given in Chapter 5.

2
Problem Definition

The following sections give a short introduction into the topic of the process-to-node mapping problem addressed in this thesis.

2.1 Inter-Process Communication

The execution time t of a parallel application with assignment A over a set of processors P is determined by the processor $p \in P$ with the highest turnaround time t_p [7]:

$$t(A) = \max_{p \in P} t_p(A) \tag{2.1}$$

In order to reduce the total running time of the parallel application the time of the processor with the highest turnaround time has to be minimized, also referred to as the *minimax criterion*, leading to an optimal assignment A_0 [7, 8]:

$$t(A_0) = \min_{A} t(A) \tag{2.2}$$

$$t(A_0) = \min_{A} \max_{p \in P} t_p(A) \tag{2.3}$$

During the execution of a parallel application the turnaround time t_p for any processor can be split into one of the following time periods [8]:

1. *Time Spent Calculating*: Denotes the time the processor p actually spends for computation.

2. *Time Spent Communicating*: Consists of the time processor p spends initiating and terminating communication as well as the time spent on routing of messages. It should be noted that the overhead associated with the transfer of messages can be eliminated by dedicated hardware. Nonetheless, the influence of the interconnecting architecture becomes noticeable in the time processor p spends both housekeeping and idle.

3. *Time Spent Housekeeping*: Summarizes the time the processor spends associated with the case when more than one process is assigned to processor p. Examples are the time the processor spends scheduling the different processes or buffering incoming messages for a process while another is being processed.

4. *Time Spent Idle*: The time the processor p spends idling can be divided into the time the processor has to wait for the others to finish and the time the processor has to wait before associated messages are sent or in transit.

Therefore the two main objectives in minimizing the total execution time of parallel applications are [6]:

1. *Load balance*: The workload should be balanced among the processors as evenly as possible, so that not one single processor is running idle, waiting for the others to complete processing.

2. *Communication costs*: Reduce the amount of communication delays, i.e. the time processors spend idling, due to message latencies.

2.2 The Mapping Problem

In its most general form the mapping problem can be seen as a minimization problem of an associated cost metric related to the assignment of processing entities (processes or threads) onto a target platform (network of processors or computing nodes). As it happens, a wide range of the generic optimization problems can be formulated as a quadratic assignment problem, which falls into the category of the strongly NP-hard problems [1]. An exact polynomial time algorithm for the generic mapping problem is therefore unlikely to be found [9].

The communication between n processes in a parallel application can be modeled as a graph $G_p = \langle V_p, E_p \rangle$, where the vertices V_p represent the set of processes and the edges $E_p \subseteq V_p \times V_p$ the corresponding communications pairs [7, 9]. Since communication between processes may not be uniform, a more precise representation is the formulation $G_p = \langle V_p, w_p \rangle$, where $w_p(u, v)$ denotes a metric of communication (message volume or message count) between any two processes $u, v \in V_p$. The weight $w_p(u, v)$ is equal to zero if no such communication occurs between vertices u and v [1, 5]. Similarly the platform architecture of m processors can be represented as topology graph $G_t = \langle V_t, E_t \rangle$, where V_t denotes the set of processors and $E_t \subseteq V_t \times V_t$ the interconnecting links [7, 9]. Again, if the connecting links between the processors are not equal, the topology graph can be defined as $G_t = \langle V_t, w_t \rangle$, where $w_t(i, j)$ denotes the link quality between any two processors $i, j \in V_t$ and $w_t(i, j) = 0$ if no such link exists [1, 5].

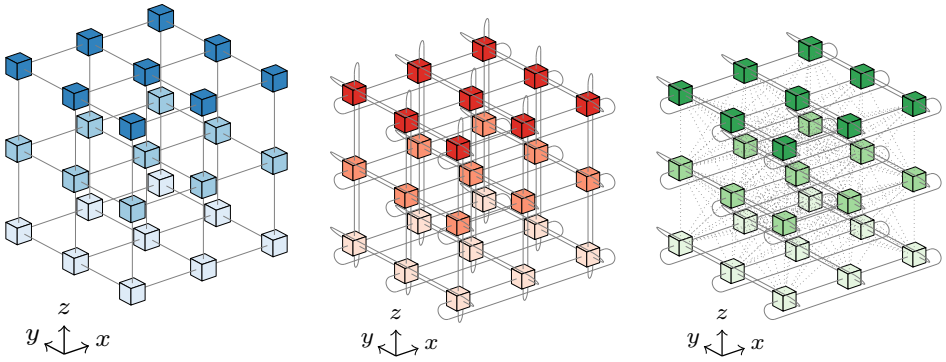
In literature the mapping problem is often introduced as a graph embedding problem [1]. Bokhari defined the mapping problem as the task of finding a injective function $M : V_p \to V_t$ which maps the vertices of the communication graph V_p (processes) to the vertices of the topology graph V_t (processors), s.t. as many edges E_p in the process graph G_p directly fall on edges E_t in the topology graph G_t. He referred to the number of overlaying edges resulting from the mapping as the cardinality $|M|$ of the mapping [9]. Shen et al. formulated the mapping problem as the finding of an optimal weak graph homomorphismus from V_p to V_t, in regard to the objectives listed in Section 2.1. The communication graph G_p is said to be weakly homomorphic to the topology graph G_t if there exists a mapping $M : V_p \to V_t$

s.t. for any edge $\langle u, v\rangle \in E_p$ there is an edge $\langle M(u), M(v)\rangle \in E_t$ including the case where $M(u) = M(v)$ [7]. Hoefler et al. state in their comprehensive review of process mapping techniques that the mapping function $M : V_p \to V_t$ neither has to be injective nor surjective. Therefore it is maybe the case that multiple vertices in V_p are mapped onto the same vertex in V_t and/or some of the vertices in V_t may remain unassigned [1].

In the scope of this work, the emphasis lies on static process-to-node mapping based on the point-to-point communication matrices obtained by processing the communication traces of parallel applications. The objective is to reduce inter-processor communication, while not diminishing the benefit of spreading the workload as evenly as possible among all processors.

2.3 Hardware Topologies

For the different mapping strategies (see Chapter 3) three node/processor topologies were considered, namely a three dimensional mesh (k-ary 3-mesh), a three dimensional torus (k-ary 3-cube) and the HAEC topology. The 3-D mesh and 3-D torus are fairly common in HPC systems, whereas the HAEC topology is unique to the HAEC Box (see Section 2.4). A graphical representation of the studied topologies is given in Figure 2.1. For better visibility a 3×3×3 version instead of the 4×4×4 version of the HAEC Box is shown. It should be noted that in particular the 3-D mesh represents a subset of the HAEC topology.



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

Figure 2.1: Schematic representation of a 3×3×3 mesh (3-ary 3-mesh), torus (3-ary 3-cube) and the HAEC topology. The represented HAEC topology consists of layers of 3×3 computing nodes connected to a 2-D torus with optical links (*solid lines*). The three tori are interconnected by wireless links (*dotted lines*).

2.4 HAEC Box

The HAEC (Highly Adaptive Energy-Efficient Computing) project is a research program funded by the German Research Foundation focusing on hardware and software adaptivity to increase energy efficiency in parallel computing. The collaborative research center con-

sists of 16 chairs from the departments of Computer Science and Electrical Engineering at Technische Universität Dresden [4]. The project is subdivided into two main research focus areas: The underlying hardware with it's technological difficulties and the self-adaptive software stack. The target platform called HAEC Box is a new type of computing architecture, consisting of 4 boards of 4×4 computing nodes cross-linked as 2-D torus by high transmission optical connections. Adjacent boards are fully interconnected by high-speed wireless links [3, 10]. Part of the technological research focus is the event trace-based simulator HAEC-SIM. It allows the study of energy-aware computation by emulation of the platform, since the actual hardware is still in development [10]. Because minimizing communication overhead in parallel applications is fundamental to energy efficiency, communication aware mapping strategies for the HAEC topology are of great interest [3].

2.5 Communication Traces

For evaluation of the mapping strategies a communication-intensive parallel benchmark from the NPB3.3[2] (NAS Parallel Benchmark) suite was used. Communication traces for the Lower-Upper Gauss-Seidel sovler (LU) with 64, 512 and 4096 MPI processes and the problem class C respectively D (denoted as LU.C.64, LU.D.512, LU.D.4096) were recorded on the HPC system Taurus at Technische Universität Dresden.[3] The OTF2 traces were then processed with the communication-matrix module of the HAEC simulator (HAEC-SIM) to obtain the unicast (point-to-point) communication matrices.

2.6 Performance Metrics

In order to determine the quality of a mapping $M : V_p \to V_t$ and hence to optimize towards it, a cost metric has to be defined. Although there is no general agreement, two different cost metrics are commonly found in literature: *dilation* and *congestion*. The dilation of a mapping M is either defined as the maximum or the sum of the weighted distance of two vertices $u, v \in V_p$ mapped to two vertices $i, j \in V_t$ [1]:

$$\sum_{u,v \in V_P} d_t(M(u), M(v)) \times w(u, v) \tag{2.4}$$

Where d_t denotes the length of the (shortest) route between vertices i, j in the topology graph. In contrast, the congestion is defined as the maximal number of communication pairs which use a certain link in the network. Either of the two communication metrics are dependent on the used routing algorithm [1].

For the purpose of this work, the quality of the different mapping strategies are evaluated by using the following metrics [3]:

1. Inter-process logical communication (IePLC): Defines the number of exchanged mes-

[2] http://www.nas.nasa.gov/publications/npb.html
[3] https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus

sages between any two processes. The IePLC is independent of the mapping strategy and a property of the communication graph G_p.

2. Intra-node logical communication (IaNLC): Corresponds to the number of messages processes exchange who are mapped onto the same node.

3. Inter-node logical communication (IeNLC): The sum of exchanged messages between nodes resulting from the mapping of the communication pairs but neglecting the network topology.

4. Inter-node physical communication (IeNPC): The number of messages two nodes exchanged regarding the topology of the system and the routing algorithm.

As to reduce network contention and related communication latencies, an optimal mapping strategy should therefore strive for reducing the number of IeNPC and increase the number of IaNLC.

3
Mapping Strategies

Starting with Bokhari's MAPPER [9] algorithm in 1981, a wide range of algorithms have been suggested for the mapping problem in its various formulations [11]. These can be approximately divided into three main categories [7]:

1. *Heuristics*: Heuristic approaches often provide fast but suboptimal solutions where an optimal solution can not be computed in a reasonable amount of time.
2. *Mathematical programming*: The problem is formulated as an optimization problem and solved by mathematical programming techniques.
3. *Graph theoretic methods*: A minimal-cut algorithm is applied to the task graph to get a mapping with minimal inter-processor communication.

A more fine grained classification of proposed mapping algorithms in literature by Kafil et al. [11] is given in Figure 3.1.

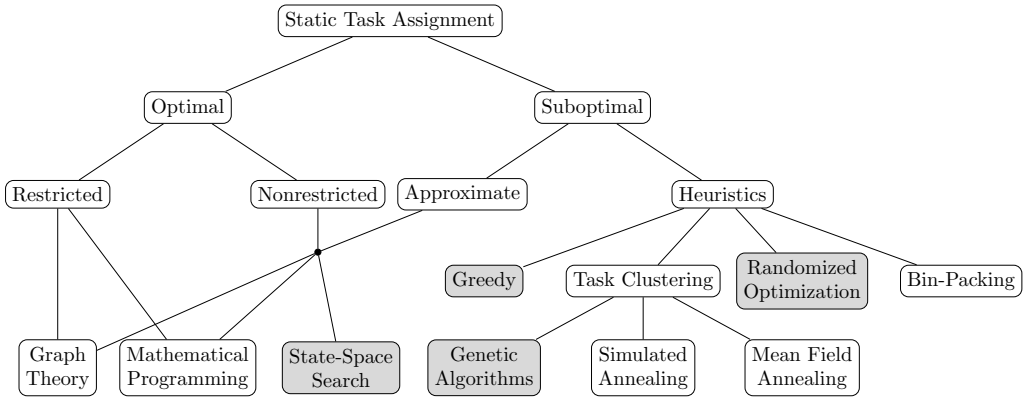


Figure 3.1: Classification of algorithms for static task assignment. Considered strategies are shaded in gray.

In the following the different mapping algorithms studied in this work are described. They can be classified into four of the given categories: Randomized Optimization (Bokhari),

Greedy (Minimum Manhattan Distance), Genetic Algorithms and Space-Filling Curves (SFC). The mapping strategies based on the SFC do not fall into any of listed categories since they are communication oblivious approaches. As indicated by the highlighting in Figure 3.1, initially, another algorithm based on state-space search was taken into consideration, but the algorithmic performance of the implementation limited the applicability to the given context (see Section 5.1).

3.1 Randomized Optimization

Bokhari proposed a mapping algorithm called MAPPER for the assignment of parallel applications solving structural problems on the finite element machine (FEM). The FEM was an array of microprocessors interconnected in a 8 nearest-neighbor fashion at NASA Langley Research Center. His heuristic algorithm uses pairwise exchanges combined with probabilistic jumps in order to improve an initial (e.g. randomly generated) mapping $M : V_p \to V_t$. He defines the number of overlapping edges in the process graph G_p with the edges in the processor graph G_t as the cardinality $|M|$, which determines the quality of a given mapping M. The cardinality is therefore used as the objective function in the randomized optimization approach [9]. The listing for the MAPPER algorithm is given below:

Algorithm 1: MAPPER

Data: G_p, G_t, $M : V_p \to V_t$ (initial mapping)
Result: $B : V_p \to V_t$ (improved mapping)

```
1  B ← M;
2  done ← false;
3  while not done do
4      repeat
5          improved ← false;
6          foreach u ∈ V_p do
7              1: examine pairwise exchange of u with every v ∈ V_p \ {u};
8              2: select pair ⟨u, v⟩ with largest gain in |M|;
9              3: if largest gain ≥ 0 then exchange pair ⟨u, v⟩;
10             4: if largest gain > 0 then improved ← true;
11         end
12     until improved = false;
13     if |M| < |B| then
14         done ← true;
15     else
16         B ← M;
17         randomly interchange n pairs in M;
18     end
19 end
```

Bokhari does not concern the case if the number of processes is higher than the number of

processors, i.e. $|V_p| > |V_t|$. Although the approach can easily modified to suit this case. The algorithm starts with an initial mapping which can be a random assignment or in our case the distribution of the processes with increasing index along the x, y and z in a sweeping motion (see Sweep SFC, Section 3.3). The algorithm then proceeds by pairwise exchanges of vertices which lead to the largest gain in the cardinality of the mapping. If the mapping can not be improved any further, for example in a (local) minima, the algorithm interchanges randomly n pairs in the current mapping, where n corresponds to the size of the $n \times n$ FEM array. The run time complexity of the algorithm is considered to be $\mathcal{O}(n^3)$ [9].

3.2 Minimum Manhattan Distance

The idea behind the minimum Manhattan distance (MinMD) algorithm is that the most communicating pairs are mapped first and as close as possible with respect to the number of hops. The greedy algorithm proceeds as follows: The communication pairs are added to a priority queue ordered by decreasing edge weight (total exchanged message size or message count). The first communication pair is mapped to the center of the architecture with a node distance of 1 hop. The processes of the following pairs are either mapped to a free node as close as possible (cf. minimal Manhattan distance) to an already assigned process, if it happens that one of the processes in the communication pair is already assigned, or as close as possible to the center. If there is no free node left in the search of the nearest-neighbor, the process-per-node limit is increased by one. The listing of the algorithm is given below, where the function Nearestneigbor is implemented as a breadth-first search for the next free node.

Algorithm 2: MinMD

Data: E_p (desc. ordered by weight), G_t
Result: $M : V_p \to V_t$

```
1  M ← { };
2  start ← (⌊d_x/2⌋, ⌊d_y/2⌋, ⌊d_z/2⌋);
3  foreach (u, v) ∈ E_p do
4      switch (u,v) do
5          case u ∈ M, v ∉ M
6              M(v) ← NEARESTNEIGHBOR(u);
7          case u ∉ M, v ∈ M
8              M(u) ← NEARESTNEIGHBOR(v);
9          case u ∉ M, v ∉ M
10             M(u) ← NEARESTNEIGHBOR(start);
11             M(v) ← NEARESTNEIGHBOR(u);
12     endsw
13 end
```

3.3 Space-Filling Curves

Space-filling curves (SFC) discovered by Peano and later extended by Hilbert in the nineteenth century [12, 13] allow a mapping of a linear array onto a two- or three-dimensional mesh. SFC are used in many applications in computer science, such as databases, image processing or parallel computing, since they allow for proximity-improving mappings, i.e. two points with distance d in the linear array, are are at most $\sqrt[2]{d}$ or $\sqrt[3]{d}$ apart in the 2-D respectively 3-D mesh [14].

In this work, five space-filling curves described by Mokbel et al. [13] were used, namely Sweep, Scan, Peano, Gray and Hilbert SFC. Three-dimensional representations of the curves are given in Figure 3.3. It should be taken into account that while the construction of the Hilbert SFC in two dimensions is unique, for the three-dimensional case there exist multiple variants. The interested reader is referred to Bader's work about SFC [15].

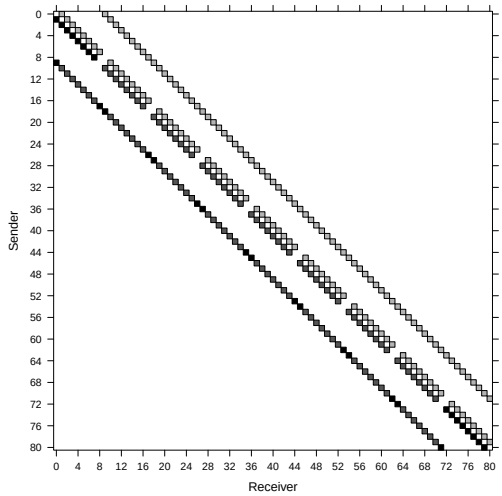
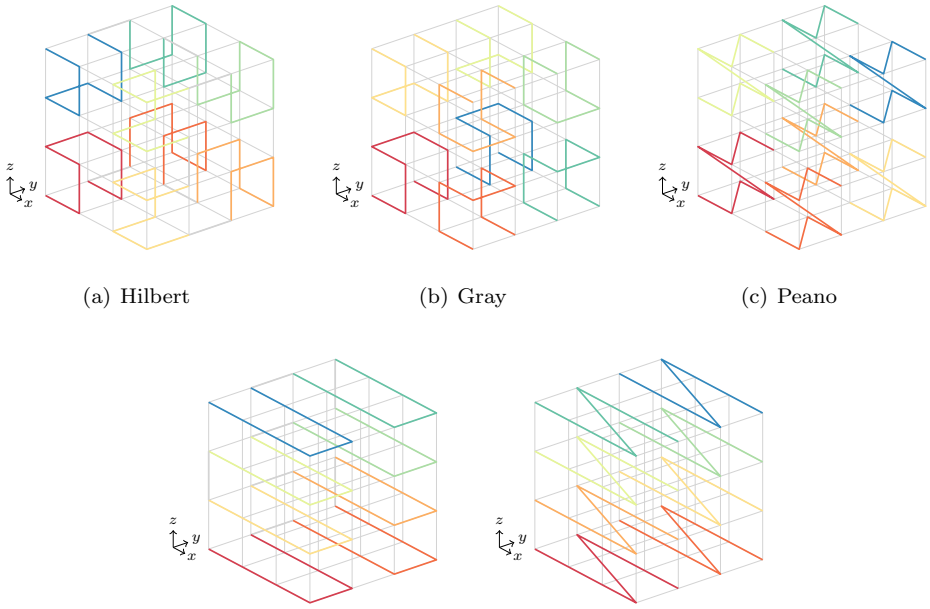


Figure 3.2: Spy plot of the communication matrix for the point-to-point messages of the LU NAS parallel benchmark with problem class C and 81 MPI processes (lu.C.81).

SFC belong to the communication oblivious mapping strategies, since the communication graph is not directly taken into account by the approach. Nonetheless as shown in Figure 3.2 the communication matrix for the MPI processes in the LU benchmark is not dispersed or in other words, a process does only communicate with a few other processes in its direct neighborhood. Since SFC allow for proximity-improving assignments, a mapping along a SFC therefore may allow for data locality between processes.

3.4 Genetic Algorithm

Inspired by biological evolution, genetic algorithms (GA) have been successfully applied to many practical optimization problems like optimization of gas pipelines or VLSI layout [16]. As population-based optimization technique GA require the formulation of the problem in a genetic representation, i.e. a sequence of characters also referred as chromosome, combined



(a) Hilbert (b) Gray (c) Peano

(d) Scan (e) Sweep

Figure 3.3: Three-dimensional space-filling curves (SFC) used in the context of this work. The curves proceed along the colors red-orange-yellow-olive-green-blue.

with a fitness function. The strategy then uses the algorithmic abstractions of the mechanism found in genetics to optimize an initial (random) population [17]:

1. *Selection*: Give preference to fitter offspring.
2. *Crossover*: Recombination through mating between two sequences.
3. *Mutation*: Pointwise alteration of the sequence.

While most genetic algorithms found in literature follow a general structure (see Algorithm 3), the implementation of the three operators, i.e. selection, crossover and mutation differ based on the problem [16]. For the implementation of the genetic algorithm in this thesis a fitness proportionate selection also referred to as roulette wheel selection strategy combined with the ordered crossover (OX) operator (see Section A.1) was chosen [16, 18]. A crossover operator which preserves the relative ratios between the characters in the string is important, otherwise a penalty for letting too many processors unassigned has to be introduced. Mutation of the string is based on exchange of two randomly chosen elements in the string. As fitness function the cardinality (see Section 3.1) was used. An initial population of 100 candidates was optimized over 10 generations.

Algorithm 3: Genetic Algorithm

Data: G_p, G_t, *size*

Result: $M : V_p \to V_t$

```
initialize population;
while not done do
    while |generation| < size do
        select parents (based on fitness or random);
        apply crossover;
        apply mutation;
        evaluate offspring for fitness;
        add offspring to generation if accepted;
    end
end
```

4 Experiments

In order to compare the mapping strategies three communication traces for the LU parallel benchmark with a different numbers of MPI processes were used (LU.C.64, LU.D.512 and LU.D.4096). The extracted point-to-point communication matrices were then taken as input for the mapping onto the selected topologies (3D-Mesh/3D-Torus/HAEC). The number of nodes in the target topology are chosen to be equal to the number of processes in the benchmark, allowing for a one-to-one mapping of the processes to the nodes. The execution time for each of the mapping algorithm was recorded using the profiler in Python's standard library (`cProfile`).[4] To determine the quality of the mapping strategies with respect to minimizing the inter-node communication, the statistics for the mapping metrics explained in Section 2.6 were evaluated. As communication model between non-adjacent nodes in the topology dimension-order routing (DOR) was chosen.

4.1 Results

The results from the run time analysis and the mapping statistics are given below. In addition to the mapping algorithms presented in Chapter 3, the best candidate from a random sampling of 10'000 configurations was taken for comparison.

Table 4.1 lists the algorithmic run time in seconds for the different topologies and problem sizes. Since Bokhari's as well as the genetic algorithm (GA) did not finish in a reasonable amount of time by mapping 4096 MPI processes to any of the three studied topologies, computation was stopped after 3 hours and the best mapping found so far was returned.

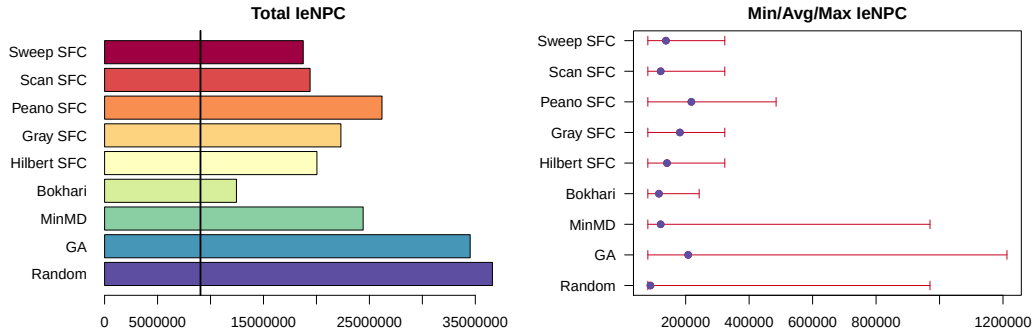
The statistics of the inter-node physical communications (IeNPC, see Section 2.6) are shown in Figure 4.1, 4.2 and 4.3. The corresponding numerical values are given in the Appendix (Section A.2). Due to the one-to-one mapping of processes and nodes, the values for the inter-node logical communications (IeNLC) are equal to the numbers of the inter-process

[4] It should be noted that the profiler itself affects the algorithmic performance, but the time differences resulting from parsing of large input files are not taken into account as it would be using time statistics provided by the unix `time` command.

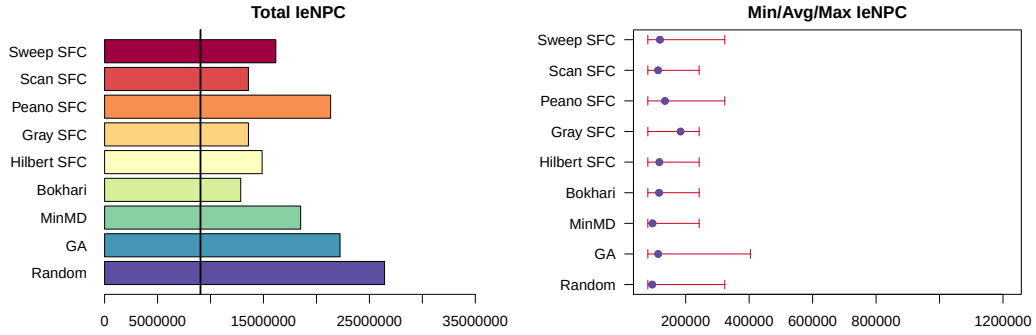
Algorithm	**Run Time [s]**	**Run Time [s]**	**Run Time [s]**
LU.C.64	4×4×4 3-D Mesh	4×4×4 3-D Torus	4×4×4 HAEC
Scan SFC	0.000	0.000	0.000
Sweep SFC	0.000	0.000	0.000
Peano SFC	0.014	0.012	0.012
Gray SFC	0.012	0.010	0.010
Hilbert SFC	0.013	0.012	0.012
Bokhari	22.625	16.810	47.462
MinMD	0.003	0.005	0.007
Genetic Algorithm	166.493	153.431	171.072
Random Sampling	9.008	9.893	19.716
LU.D.512	8×8×8 3-D Mesh	8×8×8 3-D Torus	8×8×8 HAEC
Scan SFC	0.002	0.002	0.002
Sweep SFC	0.002	0.002	0.002
Peano SFC	0.099	0.098	0.098
Gray SFC	0.079	0.081	0.079
Hilbert SFC	0.121	0.094	0.096
Bokhari	273.349	678.054	3206.106
MinMD	0.143	0.071	0.040
Genetic Algorithm	881.562	1029.892	1737.342
Random Sampling	76.412	82.441	457.194
LU.D.4096	16×16×16 3-D Mesh	16×16×16 3-D Torus	16×16×16 HAEC
Scan SFC	0.004	0.004	0.004
Sweep SFC	0.004	0.004	0.004
Peano SFC	0.214	0.211	0.214
Gray SFC	0.178	0.176	0.178
Hilbert SFC	0.214	0.210	0.208
Bokhari	*	*	*
MinMD	1.116	0.221	0.845
Genetic Algorithm	*	*	*
Random Sampling	664.814	708.230	*

Table 4.1: Run time in seconds for a single execution of the different mapping algorithms. All measurements were performed on a 2.3 GHz Intel Core i5-5300U processor, with 8 GB of memory, running Fedora Linux 23.

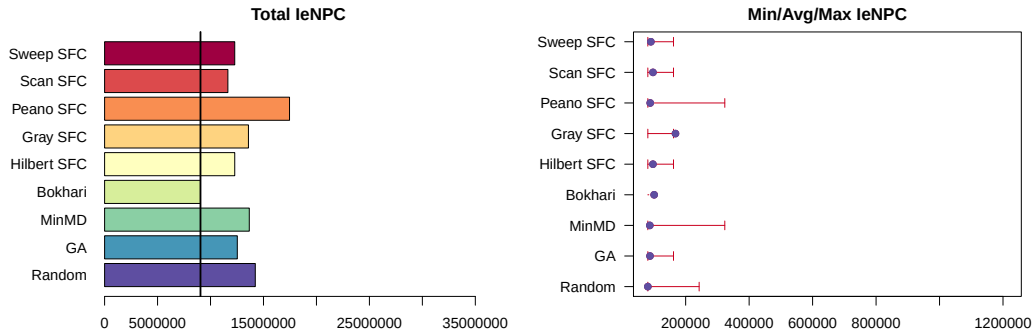
logical communications (IePLC). For the same reason the intra-node logical communications (IaNLC) are equal to zero. Hence, the values for IaNLC and IeNLC are not listed. The limits of the abscissae in the plots were set equally to allow better comparison between the topologies. A vertical bar in the graphics for the total number of IeNPC represents the value of total IeNLC. The corresponding minimal, maximal and average values overlay with the minimal values for the IeNPC plot and are therefore not shown.



(a) 4×4×4 3-D Mesh

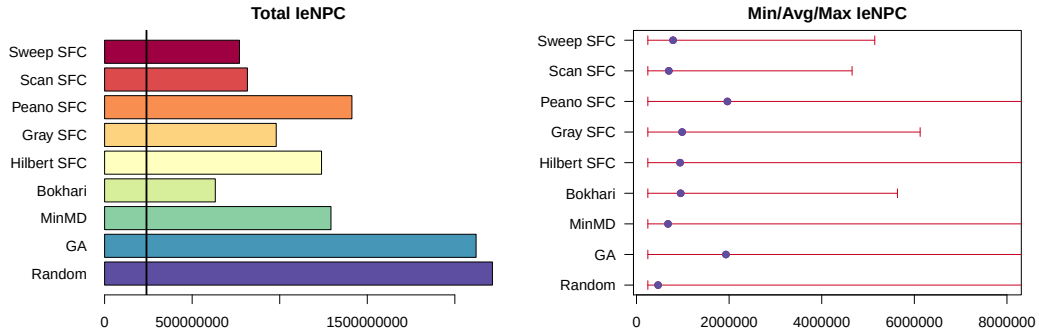


(b) 4×4×4 3-D Torus

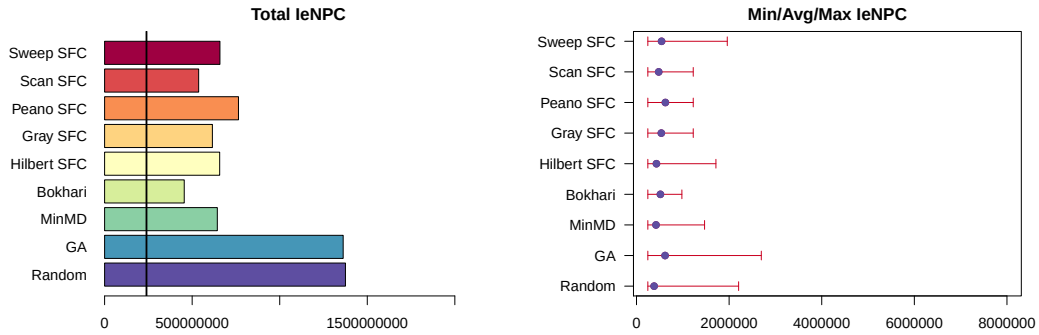


(c) 4×4×4 HAEC

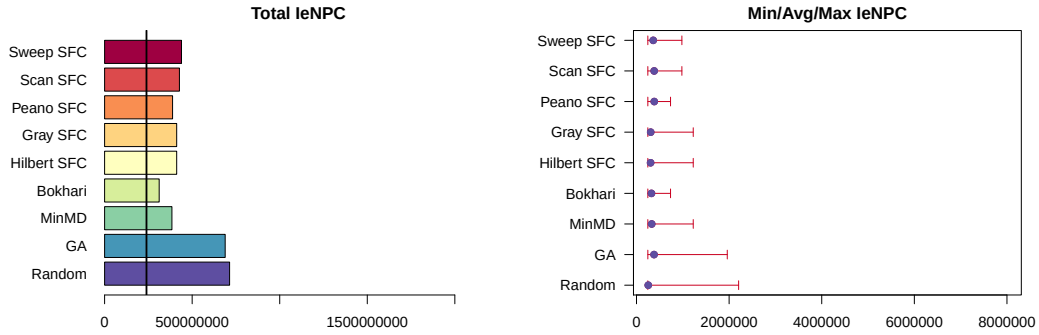
Figure 4.1: Statistics for the number of inter-node physical communications (IeNPC) obtained by mapping the communication graph of the benchmark **LU.C.64** onto 4×4×4 3-D mesh, 3-D torus and the HAEC topology. The vertical line in the plots for the total IeNPC represent the total values for the IePLC. The numerical values are given in Table A.1 in the Appendix.



(a) 8×8×8 3-D Mesh

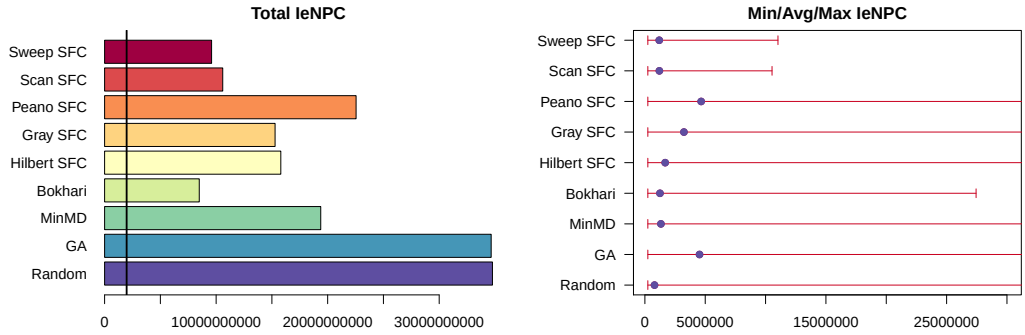


(b) 8×8×8 3-D Torus

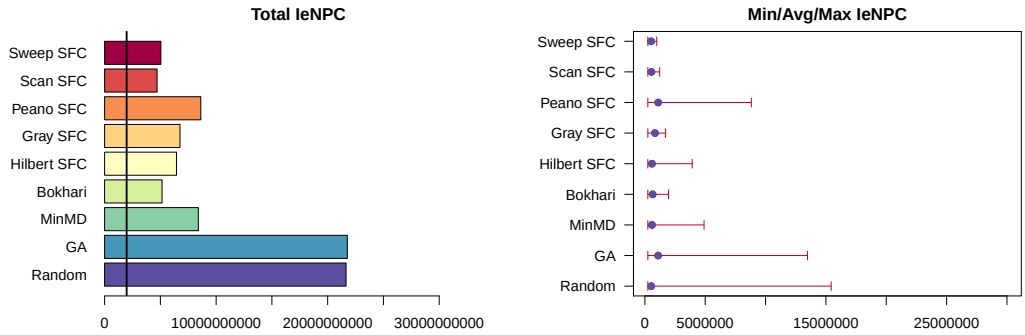


(c) 8×8×8 HAEC

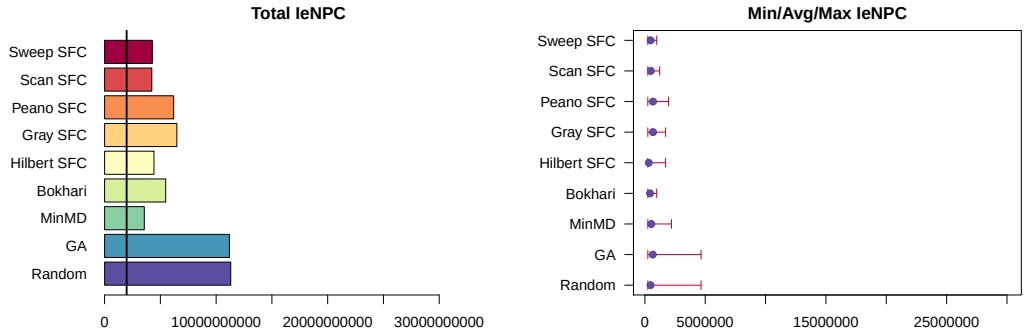
Figure 4.2: IeNPC statistics for mapping **LU.D.512** onto a 8×8×8 3-D mesh, 3-D torus and the HAEC topology. The numerical values are given in Table A.2.



(a) 16×16×16 3-D Mesh



(b) 16×16×16 3-D Torus



(c) 16×16×16 HAEC

Figure 4.3: IeNPC statistics for mapping **LU.D.4096** onto a 16×16×16 3-D mesh, 3-D torus and the HAEC topology. The numerical values are given in Table A.3.

4.2 Discussion

As shown in Figure 4.1, 4.2 and 4.3, a general trend in all of the three different problem sizes (LU.C.64, LU.D.512, LU.D.4096) is that the number of total inter-node physical communications (IeNPC) is diminishing from the 3-D mesh, over the 3-D torus to the HAEC topology for all of the considered algorithms. This is associated with the increased number of links between nodes along the topologies and therefore the decreased average distance between any two nodes. In the case of mapping of LU.C.64 to the 4×4×4 HAEC topology Bokhari's algorithm yields a perfect mapping, i.e. the number of total IeNPC is equal to the inter-process logical communications (IePLC).
While Bokhari's algorithm performs best with respect to the total number of IeNPC in almost all of the cases, the maximum value for the number of IeNPC in case of mapping to a 16×16×16 topology is higher as with the Sweep SFC, which was used as initial mapping in Bokhari's algorithm. This could be justified by the fact that the algorithm was stopped after 3 hours execution time. The genetic algorithm (GA) did not provide satisfactory results, as the number of total IeNPC is only slightly less lower than the random sampling strategy and always higher for the average number of IeNPC compared to all strategies. This may be attributed to the relative small number of generations or the implementation of the crossover strategy. Since the length of the mating sequence was chosen randomly, the ordered crossover may lead to a completely new configuration rather than an optimization of a local region. Therefore a strategy with constrained crossover sequences may should be considered in the future.

Overall, the strategies based on space-filling curves (SFC) - expect for a few cases of the Peano SFC - generally led to lower total (and minimum, maximum and average) values for the number of IeNPC compared to either the genetic algorithm or randomized sampling strategy. This results from the fact that the used communication matrices for the LU benchmark are not dispersed, i.e. a process only communicates with a few processes in its direct neighborhood. The mapping statistics of the MinMD strategy did not show the desired results, which may also are related to the regular communication pattern of the LU parallel benchmark. As shown in Figure 3.2 a lot of the communication pairs have an equal number of exchanged messages and therefore the greedy behaviour of the algorithm leads to an imbalanced assignment of the processes onto the nodes, underpinned by the large difference between minimal and maximal values in the IeNPC.

While the mapping statistics for Bokhari's algorithm show good results for the most part, the run time performance increased dramatically from a 8×8×8 topology to a 16×16×16 topology (remember the time complexity $\mathcal{O}(n^3)$) so that execution had to be conducted time-constrained to guarantee termination within a reasonable amount of time. A test run of 8 hours for mapping LU.D.4096 onto a 16×16×16 3-D mesh did not yield a better cardinality after a certain amount of time, hence the algorithm either got stuck on a plateau or the effect of the randomized interchanges of the n process pairs hitting a (local) minimum was revoked in the subsequent optimization step. The run time within a problem size for algorithms based on the cardinality as optimization funcion was observed to be considerably

higher for the HAEC topology than the 3-D mesh and torus, which is a side effect from the implementation to calculate the cardinality. As expected, the execution times of the SFC did not increase significantly compared to the other algorithms, which results from the $\mathcal{O}(n)$ time complexity of the algorithms.

5 Conclusions

Process placement onto the physical computing units of a High-Performance Computing (HPC) system plays an important role to improve performance and energy efficiency while executing a parallel application [1, 3]. In this work different strategies for mapping processes onto computing nodes in favor for data locality were presented and analyzed for their run time performance as well as the mapping performance according to the defined statistics. As the results indicate, a careful placement of processes of a parallel application can have a significant effect on the number of inter-node physical communications (IeNPC). Nevertheless, the reduced run time of the parallel application and the resulting increase in energy efficiency yet has to be proven by the emulation on the HAEC simulator.
The benefit of mapping processes onto nodes in favor for data locality may not justify a mapping algorithm which takes considerably more time to complete than the application itself. Hence, the run time of the mapping algorithm plays an important role in selecting a mapping strategy. While Bokhari's heuristic based on randomized optimization showed good results in minimizing the number of IeNPC, execution time was bounded by the run time complexity of the algorithm.

5.1 Future Work

As mentioned in Chapter 3 an additional mapping strategy based on the well-known $A^\star$ algorithm in artificial intelligence was considered. Since $A^\star$ search was applied successfully to many NP-hard problems and guarantees to find and optimal solution given an admissible and consistent heuristic, it seemed to be an ideal mapping strategy. While Shen et. al [7] showed that their algorithm based on $A^\star$ leads to an optimal assignment with heterogeneous processing units and nonidentical communications links, the presented heuristic did not yield acceptable results in our context, leading to an exhaustive search in the state-space. The definition of an appropriate heuristic function for partial assignments in the HAEC Box may lead to a improvement on the method. In order to reduce complexity of the mapping task and at the same time allow for parallelization [1], approaches based on recursive partitioning should be considered in the future.

Bibliography

[1] Hoefler, T., Jeannot, E., and Mercier, G. An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. In Jeannot, E. and Zilinskas, J., editors, *High Performance Computing on Complex Environments*, pages 75–94. Wiley (2014).

[2] Bhatele, A. *Automating Topology Aware Mapping for Supercomputers*. Ph.D. thesis, Dept. of Computer Science, University of Illinois (2010). http://hdl.handle.net/2142/16578.

[3] 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*. Springer International Publishing, Cham (2014).

[4] Fettweis, G., Nagel, W., and Lehner, W. Pathways to servers of the future. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1161–1166 (2012).

[5] Hoefler, T. and 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).

[6] Mohan, R., Gopalan, N. P., Prasanth, S. H. D., Sanyam, S., and Varma, S. R. Parallel heuristic graph matching algorithm for task assignment problem in distributed computing systems. In *Computer Information Science (ICCIS), 2012 International Conference on*, volume 2, pages 575–579 (2012).

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

[8] Norman, M. G. and Thanisch, P. Models of Machines and Computation for Mapping in Multicomputers. *ACM Comput. Surv.*, 25(3):263–302 (1993).

[9] Bokhari, S. H. On the Mapping Problem. *IEEE Transactions on Computers*, C-30(3):207–214 (1981).

[10] Bielert, M., Ciorba, F., Feldhoff, K., Ilsche, T., and Nagel, W. HAEC-SIM: A Simulation Framework for Highly Adaptive Energy-Efficient Computing Platforms. ACM (2015).

[11] Kafil, M. and Ahmad, I. Optimal Task Assignment in Heterogeneous Distributed Computing Systems. *IEEE Concurrency*, 6(3):42–51 (1998).

[12] Mokbel, M. F., Aref, W. G., and Kamel, I. Performance of Multi-dimensional Space-filling Curves. In *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems*, GIS '02, pages 149–154. ACM, New York, NY, USA (2002).

[13] Mokbel, M. F. and Aref, W. G. Irregularity in High-dimensional Space-filling Curves. *Distrib. Parallel Databases*, 29(3):217–238 (2011).

[14] Ahmed, M. and Bokhari, S. Mapping with Space Filling Surfaces. *IEEE Transactions on Parallel and Distributed Systems*, 18(9):1258–1269 (2007).

[15] Bader, M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing (Texts in Computationproblemal Science and Engineering)*. Springer, 2013 edition (2012).

[16] Moscato, P. On Genetic Crossover Operators for Relative Order Preservation. In *Caltech Concurrent Computation Program, Report C3P-778* (1989).

[17] Chockalingam, T. and Arunkumar, S. Genetic Algorithm Based Heuristics for the Mapping Problem. *Comput. Oper. Res.*, 22(1):55–64 (1995).

[18] Lee, W. and Kim, H.-Y. Genetic algorithm implementation in Python. In *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 8–11 (2005).

A
Appendix

A.1 Ordered Crossover Operator

For illustration of the ordered crossover (OX) operator, the following example of a string encoding the Traveling Salesman Problem was taken from [16]:

1. The crossover starts with two parent strings A and B. In the following A is considered as donor and B as receptor string. The matching section is given in bold type.

$$\begin{aligned} \text{A} &= 984\mathbf{567}1320 \\ \text{B} &= 871\mathbf{230}9546 \end{aligned}$$

2. Since the resulting offspring should have the same set of numbers, the numbers in the matching section of the donor string are deleted in the receptor string:

$$\begin{aligned} \text{A} &= 984\mathbf{567}1320 \\ \text{B} &= 8{*}1\mathbf{230}9{*}4{*} \end{aligned}$$

3. To preserve the relative order, the 'holes' in the receptor string are filled by a sliding motion starting at the end of the matching section:

$$\begin{aligned} \text{A} &= 984\mathbf{567}1320 \\ \text{B} &= 230\mathbf{***}9481 \end{aligned}$$

4. Finally the matching section of the donor string is inserted to B resulting in the offspring C:

$$\begin{aligned} \text{A} &= 984\mathbf{567}1320 \\ \text{C} &= 230\mathbf{567}9481 \end{aligned}$$

The offspring sequence D resulting by mating B with A is obtained analogue.

A.2 Mapping Statistics

LU.C.64	**IePLC**				**IeNPC**			
4×4×4 3D-Mesh	total	avg	min	max	total	avg	min	max
Sweep SFC					18752386	137885	80829	323318
Scan SFC					19399020	121243	80829	323320
Peano SFC					26188677	181865	80829	484977
Gray SFC					22308862	121243	80829	323320
Hilbert SFC	9052876	80829	80829	80830	20045654	115870	80829	323317
Bokhari					12447703	88912	80829	242488
MinMD					24410456	141100	80829	969951
Genetic Algorithm					34514094	207916	80829	1212439
Random Sampling					36615639	217950	80829	969951
LU.C.64	**IePLC**				**IeNPC**			
4×4×4 3D-Torus	total	avg	min	max	total	avg	min	max
Sweep SFC					16165850	118866	80829	323318
Scan SFC					13579314	113160	80829	242487
Peano SFC					21338922	183956	80829	323320
Gray SFC					13579314	95628	80829	242487
Hilbert SFC	9052876	80829	80829	80830	14872578	116192	80829	242488
Bokhari					12851853	94498	80829	242487
MinMD					18509915	117151	80829	242490
Genetic Algorithm					22228039	113408	80829	404147
Random Sampling					26431163	134852	80829	323319
LU.C.64	**IePLC**				**IeNPC**			
4×4×4 HAEC	total	avg	min	max	total	avg	min	max
Sweep SFC					12286046	90338	80829	161659
Scan SFC					11639412	96995	80829	161660
Peano SFC					17459118	167876	80829	323320
Gray SFC					13579318	87046	80829	161660
Hilbert SFC	9052876	80829	80829	80830	12286048	100705	80829	161660
Bokhari					9052876	80829	80829	80830
MinMD					13660147	96880	80829	323316
Genetic Algorithm					12528538	87612	80829	161660
Random sampling					14225961	88360	80829	242487

Table A.1: Statistics for mapping the communication pattern of the benchmark **LU.C.64** onto a 4×4×4 3-D mesh, 3-D torus and the HAEC topology. Due to the one-to-one mapping of processes and nodes, the values for the number of intra-node logical communications (IaNLC) and inter-node logical communications (IeNLC) are omitted as they are either equal to zero or identical to the number of inter-process physical communications (IePLC).

LU.D.512	**IePLC**				**IeNPC**			
8×8×8 3D-Mesh	total	avg	min	max	total	avg	min	max
Sweep SFC					770346256	789289	245021	5145448
Scan SFC					815430112	698142	245021	4655405
Peano SFC					1412301578	985555	245021	35038020
Gray SFC					980084230	680614	245021	6125526
Hilbert SFC	239140588	245021	245021	245022	1238826575	953677	245021	29892570
Bokhari					632154385	466190	245021	5635486
MinMD					1292731518	941537	245021	30627635
Genetic Algorithm					2121392613	1932051	245021	93108014
Random Sampling					2214745678	1963427	245021	91637885

LU.D.512	**IePLC**				**IeNPC**			
8×8×8 3D-Torus	total	avg	min	max	total	avg	min	max
Sweep SFC					658616650	541625	245021	1960170
Scan SFC					537086214	479541	245021	1225107
Peano SFC					764465770	536091	245021	1225108
Gray SFC					615492978	421281	245021	1225105
Hilbert SFC	239140588	245021	245021	245022	657146564	516218	245021	1715147
Bokhari					454759126	380233	245021	980085
MinMD					643670480	430548	245021	1470128
Genetic Algorithm					1362317266	617271	245021	2695231
Random Sampling					1375303423	623720	245021	2205191

LU.D.512	**IePLC**				**IeNPC**			
8×8×8 HAEC	total	avg	min	max	total	avg	min	max
Sweep SFC					439077778	361083	245021	980085
Scan SFC					427316764	381532	245021	980086
Peano SFC					388113402	307051	245021	735065
Gray SFC					411635448	328782	245021	1225105
Hilbert SFC	239140588	245021	245021	245022	411635390	322851	245021	1225105
Bokhari SFC					311666849	253800	245021	735064
MinMD					384438141	304626	245021	1225107
Genetic Algorithm					688509266	378926	245021	1960169
Random Sampling					713746455	383322	245021	2205191

Table A.2: Statistics for mapping **LU.D.512** onto a 8×8×8 3-D mesh, 3-D torus and the HAEC topology.

LU.D.4096 16×16×16 3D-Mesh	**IePLC** total	avg	min	max	**IeNPC** total	avg	min	max
Sweep SFC					9593063400	1189615	245021	11025960
Scan SFC					10592749192	1199360	245021	10535917
Peano SFC					22549775519	3235261	245021	376352298
Gray SFC					15285391679	1328933	245021	33077835
Hilbert SFC	1975849596	245021	245021	245022	15799935759	1249797	245021	252126617
Bokhari					8487528519	797850	245021	27442372
MinMD					19380429440	1681015	245021	440057738
Genetic Algorithm					34661165102	4526138	245021	1256222830
Random Sampling					34775590067	4662232	245021	1254262678

LU.D.4096 16×16×16 3D-Torus	**IePLC** total	avg	min	max	**IeNPC** total	avg	min	max
Sweep SFC					5045473068	520205	245021	980085
Scan SFC					4704403830	544491	245021	1225107
Peano SFC					8624740298	838167	245021	8820757
Gray SFC					6758660496	590482	245021	1715148
Hilbert SFC	1975849596	245021	245021	245022	6445033014	645989	245021	3920338
Bokhari					5151322161	528666	245021	1960170
MinMD					8404956438	586283	245021	4900420
Genetic Algorithm					21762767905	1095533	245021	13476157
Random Sampling					21650058341	1099322	245021	15436323

LU.D.4096 16×16×16 HAEC	**IePLC** total	avg	min	max	**IeNPC** total	avg	min	max
Sweep SFC					4281007458	474402	245021	980085
Scan SFC					4233963420	490042	245021	1225107
Peano SFC					6194131670	674448	245021	1960172
Gray SFC					6480316588	535829	245021	1715147
Hilbert SFC	1975849596	245021	245021	245022	4426059876	432190	245021	1715149
Bokhari					5480630427	466040	245021	980085
MinMD					3558685601	319795	245021	2205191
Genetic Algorithm					11190600566	655840	245021	4655399
Random Sampling					11310415833	669849	245021	4655401

Table A.3: Statistics for mapping **LU.D.4096** onto a 16×16×16 3-D mesh, 3-D torus and the HAEC topology.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

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

Author — Autor
Daniel Besmer

Matriculation number — Matrikelnummer
2009-926-262

Title of work — Titel der Arbeit
Process-to-Node Mapping Strategies for the HAEC Box

Type of work — Typ der Arbeit
Bachelor 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, 2016

Signature — Unterschrift