Automatic Analysis of a Supercomputer's Topology and Performance Characteristics

— Bachelorarbeit —

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Alexander Bufe
E-Mail-Adresse:	1bufe@informatik.uni-hamburg.de
Matrikelnummer:	6316267
Studiengang:	Computing in Science, SP (Bio-)Chemie
Erstgutachter:	Dr. rer. nat. Julian Kunkel
Zweitgutachter:	Professor Dr.-Ing. Stephan Olbrich

Hamburg, den 18. März 2014

Abstract

Although knowing the topology and performance characteristics of a supercomputer is very important as it allows for optimisations and helps to detect bottleneck, no universal tool to determine topology and performance characteristic is available yet. Existing tools are often specialised to analyse either the behaviour of a node or of the network topology. Furthermore, existing tools are unable to detect switches despite their importance. This thesis introduces an universal method to determine the topology (including switches) and an efficient way to measure the performance characteristics of the connections.

The approach of the developed tool is to measure the latencies first and then to compute the topology by analysing the data. In the next step, the gained knowledge of the topology is used to parallelise the measurement of the throughput in order to decrease the required time or to allow for more accurate measurements. A general approach to calculate latencies of connections that cannot be measured directly based on linear regression is introduced, too

At last, the developed algorithm and measurement techniques are validated on several test cases and a perspective of future work is given.

Contents

1. Introduction and Background

First, a short introduction to High Performance Computing is given to motivate this work (Section 1.1). Then, an overview of the structure of this thesis is given (Section 1.2). This is followed by an explanation of topology (Section 1.3) and performance characteristics (Section 1.4) and ways to treat measured values (Section 1.5). At last, existing tools (Section 1.6) and aims of this thesis (Section 1.7) are presented.

1.1. Motivation

The computing power of a single processor with a single core is often insufficient for many computational demanding applications in science or engineering. Since the possibilities to improve the performance of a single core are limited by the current capabilities in microelectronics, the logical step to increase computing power is to use several CPU cores simultaneously (multi-core processing), to use several CPUs (multiprocessing) or to use several computers (nodes) building a cluster after all. The most powerful systems of a given time period are called high-performance computers or supercomputers and are listed in the top 500 list[1], which is currently led by the Tianhe-2 of the China's National University of Defense Technology providing around 3.120.000 cores.

Coordinated execution of an application requires synchronisation and communication between the distributed cores. Usually an application runs one or multiple processors on each node. Due to extreme costs it is not feasible to connect each core (or even node) directly to all others, so that one has to find a sophisticated way to connect them balancing costs and performance. The arrangement in which the cores or nodes are connected is called the topology.

In parallel programs multiple CPUs collaborate to compute the intended result. There are different approaches to share the work among the CPUs, either they perform different tasks (task/function parallelism[2]) or they perform the same operation on different data (data parallelism[3]).

It is very important to know the topology of a supercomputer because it allows for optimisations. For example climate simulations are typically divided into several cells (data parallelism) which are calculated by one computing unit but have to communicate with some of the other computing units. Since not all CPUs have the same connection, it is important to distribute the task in such a way on the computing units so that the time needed for communication is minimised.

[1] `http://www.top500.org`
[2] `http://en.wikipedia.org/wiki/Task_parallelism`
[3] `http://en.wikipedia.org/wiki/Data_parallelism`

This thesis introduces a method to determine the topology and an efficient way to measure the performance characteristics of the connections.

1.2. Structure of the Thesis

This thesis consists of four chapters, each having a short summary at the beginning.

This chapter gives a short motivation and introduces the most important terms. Measurement methods are introduced and a short overview of existing tools and their abilities is provided. At last the aims of the work are presented.

The second explains the developed algorithm by first giving an overview and the main idea. Then the algorithm is developed and explained step by step. At last, the developed output file format is displayed and a way to connect models of different levels is discussed.

In the third, different methods to measure latency are compared, the developed algorithm is evaluated on several systems and a simple model to calculate the probability of finding the correct topology is developed.

In the fourth and last chapter, the results are summarised and a perspective of possible future work is given.

A short manual of the developed tool can be found in the appendix.

1.3. Topology

The topology of a network describes how its components like CPUs or nodes are connected to each other. Usually, there are nodes connected with other nodes containing at least one socket . Every socket can contain several CPUs and each CPU several cores. There are several connection technologies between the sockets like Intels QuickPath Interconnect (QPI). The topology can be represented as a graph, where every CPU is a vertex. Nodes can either be connected directly or indirectly using a switch which can connect several nodes and other switches. Sometime redundant or alternative, thus multiple paths between two CPUs could exist. Typical connection technologies between nodes are infiniband and ethernet. [Inf]

The most efficient topology is a fully connected (or fully meshed) one, where every node is connected to all other nodes. Since for n nodes a fully meshed network requires $\frac{n(n-1)}{2}$ communication links, the cost of this topology does not scale for clusters with thousands of nodes. Common topologies are hierarchical topologies with switches (e.g the fat tree topology) or hypercubes. [PD07]

1.4. Performance Characteristics

There are two major methods of measuring the time needed to transfer a message between two processes.

One-Way-Delay (OWD): A message is sent between the two processes. The time when the message is sent and the time when the message is completely received are measured. The OWD is the difference between the times. The problem with this approach is that perfectly synchronised clocks are needed which is very difficult in practice.

Round-Trip-Time (RTT): A message is sent from process A to process B. When it is completely received, it is sent back and when it is received, the time is measured again. Since the time is always measured by the same process, no synchronisation is needed. Under the assumption that the time needed to sent a message one way is the same time needed to sent the message the other way, we can calculate the OWD by dividing the RTT by two. In typical HPC networks the assumption can be made and this is a much simpler way for measuring the OWD.

If $OWD(s)$ was plotted against the size of the message for messages of various sizes, a graph can be obtained like in Figure 1.1.

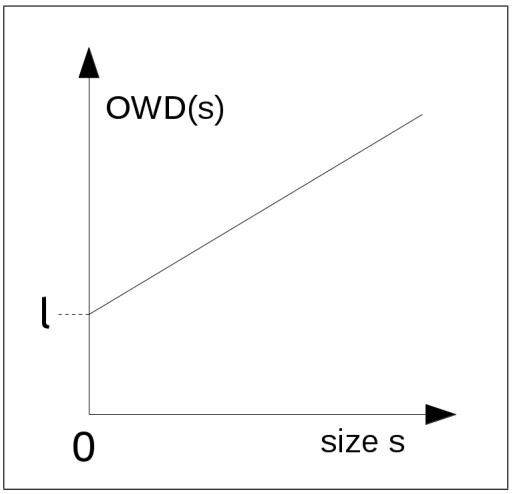

Figure 1.1.: OWD as function of the size of the massage

As expected, the needed time is a function of the size. $OWD(s)$ is linear and can be written as

$$OWD(s) = \frac{s}{t} + l. \tag{1.1}$$

l is usually called **latency** and t is usually called **throughput**. One might think, that if the size becomes zero, $OWD(s)$ will be zero, too, meaning there is no latency, but even the transfer of the smallest message will take some time due to reasons like limited speed of electricity. When many small messages are send, latency is very important. Since it is not possible to send an empty message, latency cannot be measured directly. An approximation according to (1.1) would be

$$l = \lim_{s \to 0} OWD(s) \tag{1.2}$$

because of $\lim_{s \to 0} \frac{s}{t} = 0$, but there is a lower bound of s as control data is always transferred, so that the potential of this approximation is limited. An alternative way to determine latency is to determine throughput first and then reconstruct the latency of a message with

$$l = OWD(s) - \frac{s}{t}. \tag{1.3}$$

Throughput can be measured as

$$t = \frac{s}{OWD(s) - l} = \frac{1}{\frac{OWD(s)}{s} - \frac{l}{s}} \approx \frac{s}{OWD(s)} \tag{1.4}$$

for large s.

A different way to calculate latency and throughput would be to measure OWD(s) for at least two sizes and calculate l and s directly with (in case of two different sizes)

$$t = \frac{s_2 - s_1}{OWD(s_2) - OWD(s_1)} \tag{1.5}$$

$$l = OWD(s_1) - \frac{s_1}{t} \tag{1.6}$$

or to perform a linear regression (in case of more different sizes). [PD07]

Only latency and throughput between processes can be measured with the described methods. If latency and throughput between e.g. cores have to be measured, the processes have to be pinned to the cores.

An important characteristic of the topology is the **bisection bandwidth**. If the network is divided into two parts which contain approximately the same amount of nodes so that as few cuts as possible of connections between the parts are necessary, then the bisection bandwidth is the summed bandwidth of the cut connections. If topology and performance characteristics are known the bisection bandwidth can directly be computed from them.

1.5. Measured Values

The latency (and also throughput) is - especially for small messages - subject to fluctuations (see Section 3.2.1). To reduce measurement errors, measurements have to be repeated. The received collection of values can be analysed using descriptive statistics. A typical way to describe a distribution is the five-number summary containing:

- sample minimum (the lowest measured value)
- lower quartile (the lowest value such that 25% of the measured values are lower or equal)
- median (the lowest value such that the half of the measured values is less or equal and the other half of the values is more than or equal to it)
- upper quartile (the lowest value such that 75% of the measured values are lower or equal)
- sample maximum (the highest measured value).

Another descriptor is the arithmetic mean of n values $x_1, ..., x_n$ which is defined as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{1.7}$$

Later in the developed algorithm only a single value is needed. The median should be preferred instead of the mean, because the median is much more resistant against outliers. Another possibility would be the minimum. Since the latency is limited by physical limitations, fluctuation downwards is more improbable than fluctuation upwards.

If the available methods to measure the time are very inaccurate e.g. the accuracy of the timer is in the range of the latency, it is better to measure the time of batches containing a certain number of repetitions and divide the determined time by the number of repetitions (which is equal to calculating the mean) to reduce measurement errors and use median or minimum for the means of the batches.

Even these descriptors always have a measurement error so that two measured values will hardly ever be exactly equal. Due to that, it is reasonable to compare measured values roughly. There are two major possibilities:

First, in an absolute comparison one can consider a measured value a will be less than b if

$$(b - a) > \epsilon \tag{1.8}$$

for an $\epsilon > 0$.

Second, in a relative comparison one can consider a measured value a will be less than b if $(b-a)$ is positive and relatively to the mean greater than a constant epsilon or formally if

$$\frac{b-a}{\frac{a+b}{2}} = \frac{2(b-a)}{a+b} > \epsilon \tag{1.9}$$

for an $\epsilon > 0$.

The advantage of the second method is that the tolerance depends - like the measurement error - on the level of the values. When high latencies are measured, the measurement error will probably be higher than if low latencies are measured. Due to that, only one epsilon is needed for all cases.

The disadvantage is that if the sum of two values a and b is compared to a value c, and a and c are similar values and b is much smaller then a or c, the sum and c will always be considered equal, nearly independent of the amount of b. This is a problem when one tries to measure the distance of nodes to the network interface because the value of intra node latencies will be much smaller than the value of inter node latencies, but the precision of intra node latencies is needed.

1.6. Existings Tools

There are several tools for obtaining information of intra-node or inter node topology and determining performance characteristics:

- One type of tools like Carbo or Likwid[4] are able to find information about the internal structure of nodes by reading hardware information, but cannot build up a valid model of inter-node topology nor can they characterise the connectivity.
- There are several tools to measure the performance characteristics of a certain connection like iperf[5], nuttcp[6] and traceroute[7].
- Furthermore there is a group of tools - often with a background of network administration - which are able to determine the network topology, but which are typically unable to detect switches.

1.7. Aims of the Work

The main aim of this thesis is to create a valid model of the topology so that the model can be used to detect bottlenecks and unexpected behaviour of more complex communication patterns. The topology - including switches - has to be recognised correctly and

[4] `http://www.code.google.com/p/likwid`
[5] `http://www.iperf.fr`
[6] `http://www.nuttcp.net`
[7] `http://www.tracert.org/traceroute`

the throughput and the latency of the connections have to be measured.

Since CPU time on a supercomputer is very expensive, a functional requirement is to reduce the time needed on the cluster. There are two major methods to achieve this goal:

1. The method of measuring has to be optimised in such a way that the time needed for the measurement is reduced by measuring only needed connections and by using already gained informations.

2. The measurement and the model creation are separated, so that the model creation must not be performed on the cluster. Due to that, the time needed for the model creation is less important than the time needed for the measurement.

2. Design and Implementation

In this chapter the algorithm for determining the topology as well as latency and throughput of the links is developed.

Since latency can approximately be measured faster than throughput using small sizes of messages like one byte, only latency is used to build up a model of the topology. First, the basic assumptions are introduced (Section 2.1). Then a basic model is created and the switches are identified using clique search (Section 2.2). This model is used to schedule the measurement of throughput: Since a model is available, throughput of different connections can be measured simultaneously, because it is known which connections will not influence each other (Section 2.3). A way to remeasure the latencies more accurately in an efficient way using the knowledge of the topology is shown, too (Section 2.5). The measured throughputs can be used to recalculate the true latencies according to Equation (1.3)*. A method to determine the latency of the links of a switch (which cannot be measured directly) based on a linear regression is introduced (Section 2.4).*

Even if only the approach to create the inter-node model is described, the intra-node-model (or a inter-socket-model) can be created in the same way. The inter-node and intra-node models are created one after another and eventually connected.

If the nodes consist of multiple sockets which all have the same topology, this information can be used to minimise the costs for building the intra-node model by building an intra- and an inter-socket model and connecting them.

At last, the output format (Section 2.6) and a method to connect models (Section 2.7) is described.

2.1. Basics of the Model

The model is based on the following assumptions that

1. latency and throughput are symmetric, meaning that the latency or throughput between two nodes A and B is the same as the latency/throughput between B and A so that the graph of the topology will be undirected.

2. latencies are additive, meaning that if three nodes A, B and C are connected in such a way that a is connected to B, B is connected to C and A and C are not connected, the latency between A and C will be the sum of the latency between A and B and the latency between B and C.

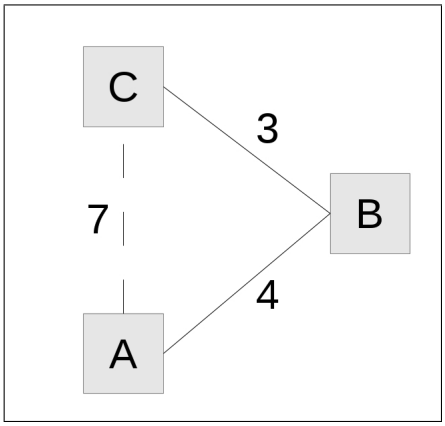

Figure 2.1.: Assumption: Additivity of latency in a path A to B, B to C

3. if three nodes are connected in the same way, the throughput between A and C will be the minimum of the throughput between A and B and the throughput between B and C.

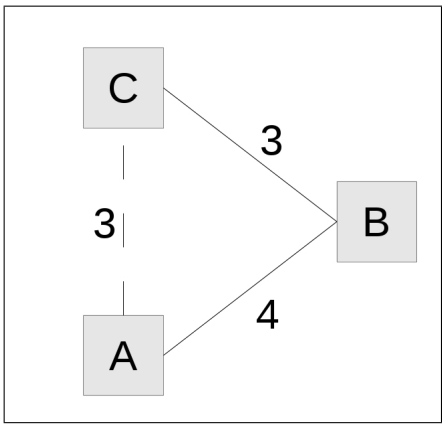

Figure 2.2.: Assumption: Throughput in a path A to B, B to C

4. if the latency between two nodes is measured, it will be assumed that the measured latency is the latency on the shortest path between A and B.

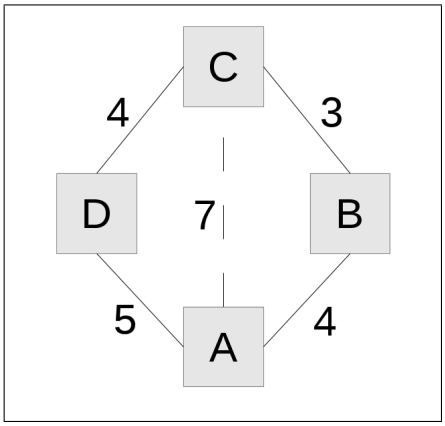

Figure 2.3.: Assumption: Latency is measured on the shortest path

2.2. Building the Model

2.2.1. Generation of the Basic Model

```
E ← ∅ ;
C ← measurePointToPointLatencies();
sortAscending(C);
foreach (a,b) ascending ∈ C do
    if latency(a,b) < shortestPathLength(a, b) then
        E ← E ∪ (a, b)
    end
end
```

Algorithm 1: Pseudocode: Generation of the basic model

To generate a basic model the following two steps, which are also sketched in Algorithm 1, are performed:

1. First, the latency of all possible node-to-node connections is measured. Let n be the number of nodes. The first node has n-1 possible connections to other nodes, the second node has n-2 due to symmetry of latency and so on. In total there will be $\frac{n(n-1)}{2}$ connections (see Equation 2.1).

$$n-1+n-2+\cdots+1 = \sum_{i=1}^{n-1} i = \sum_{i=1}^{n} i - n = \frac{n(n+1)}{2} - n = \frac{n^2+n}{2} - \frac{2n}{2} = \frac{n(n-1)}{2} \tag{2.1}$$

   All possible connections (the candidates) have to be measured, because for every possible connection information is needed whether a direct connection exists.

   Since it is not known if and which connections influence each other in this state of the algorithm, the connections cannot be measured simultaneously without the risk

of measurement errors. One can try to measure the latencies in parallel (with the scheme later described in Section 2.3.2). Single measurements have to be repeated to reduce measurement error. If different pairs are measured in every repetition simultaneously, there will be a chance that some of the measurements are not influenced by others. One can take advantage of this fact by using the minimum of the repeated measurements as the value for model creation. With parallel execution, there is no guarantee that the model is not corrupted by measurement errors but the costs for the measurements are reduced to linear time which is a great advantage especially for large clusters.

2. The initial model only consists of the nodes. The candidates are sorted ascendingly. A candidate connecting two nodes A and B will be added, if the latency of the candidate is shorter than the latency of the shortest path between A and B. Since we assume that latency is measured on the shortest path, the latency of the candidate can only be lower than or equal to the latency of the shortest path between A and B. If the latencies are equal, it will be assumed that there is no direct connection between A and B even if it were possible. Note that this condition can lead to errors (see Figures 2.4 and 2.5).

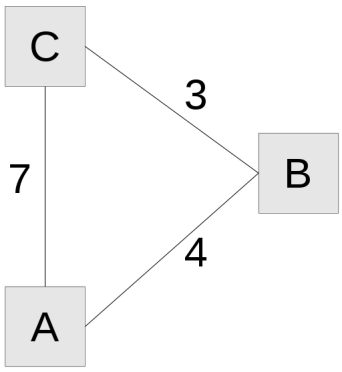

Figure 2.4.: Actual Topology

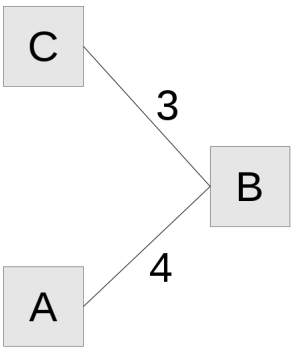

Figure 2.5.: Model of Topology

If the shortest path between A and B contains more than three nodes (A and B included), one could try to perform an additional test whether the direct connection exists by „blocking“ the shortest path: The throughput between A and B is measured. Then the throughput is measured again, but while it is measured, data is transferred between the nodes on the path. If there is a direct connection, the throughput will not change, if not, the throughput might be reduced.

Example

For example, the latencies in table 2.1 are measured:

	B	C	D
A	3	3	3
B		3	6
C			3

Table 2.1.: Example: measured latencies

In the following, the steps of the algorithm are performed on this example data.

1. Initially, the latencies are sorted and the algorithm tries to insert the edge (A,B) (Since every connections except of (B,D) have the same latency, every edge except of (B,D) could be chosen). There is no path between A and B in this state of the algorithm so that the shortest path length between A and B is ∞ and the edge is inserted. In the same way the edges (A,C) and (A,D) are inserted. The topology after this step is drawn in Figure 2.22.

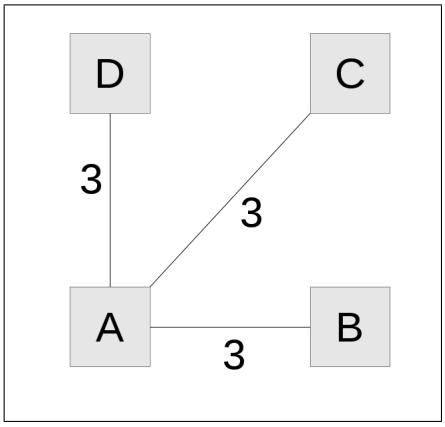

Figure 2.6.: Example: Topology after the insertion of three edges

2. The edges (B, C) and (C, D) are inserted, too, because the latency of the shortest path is 6 in this state of the algorithm and the latency of the edges is 3, which is less than 6. The topology after this step is drawn in Figure 2.7.

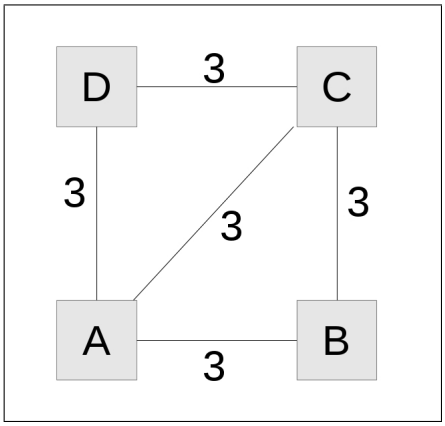

Figure 2.7.: Example: all edges inserted

3. At last the algorithm tries to insert the edge (B, D). Since the latency of the shortest path is 6 which is equal to the latency of the edge, the edge is not inserted.

2.2.2. Shortest Path Length Calculation

Since all node-to-node latencies on the shortest path are known (the measured latencies), the shortest path length between a and b can be computed using the following lemma. [CSRL01, Lemma 24.1]

> Given a weighted, directed graph G = (V, E) with weight function w : E $\rightarrow$ R, let $p = (v_1, v_2, \ldots, v_k)$ be a shortest path from vertex v_1 to vertex v_k and, for any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ be the subpath of p from vertex v_i to vertex v_j . Then, p_{ij} is a shortest path from v_i to v_j.

Let $p_{AB} = (A, \ldots, C, B)$ be the shortest path between A and B, then p_{AC} and p_{CB} will be shortest paths and the length of the shortest path will be $W(p_{AB}) = W(p_{AC}) + W(p_{CB})$. It is not known which node is C, but it is known that C has to be adjacent to b so that we compute the shortest path length by determining

$$\min_{d \in adj(b)} (W(p_{ad}) + W(p_{db})). \tag{2.2}$$

The number of nodes, n, is an upper bound for the number of nodes adjacent to B, so that the shortest path length can be computed in $\mathcal{O}(n)$. Another advantage is that the additivity of latencies is only an assumption and only one addition is needed when the length of the shortest path is calculated this way instead of the many additions if an algorithm like the djikstra algorithm would have been used.

2.2.3. Switch Detection and Replacing

```
repeat
    change ← false;
    D ← getGroupsOfLatencies(E);
    foreach l ∈ D do
        cliques ← findCliques(G, l) ;     /* where all edges have latency l */
        foreach c ∈ cliques do
            replaceBySwitch(G, c);
            change ← true;
        end
    end
until change = false ;
```

Algorithm 2: Pseudocode: Switch detection (allowing only cliques with a size of at least two)

If the topology contains switches, the nodes they connect will seem to be fully meshed. The property of a switch is to connect nodes adding the same latency to every connection. If the cluster is homogeneous, this will be able to be used to detect the switches, because the latency of all connections of nodes to the switch will be equal, which means that a switch appears as a fully meshed area where all connections have the same latency. Such areas can be detected easily using clique search like the Bron-Kerbosch algorithm. [BK73]

For each group of similar latencies is searched, whether a clique exists where all nodes are connected with latencies belonging to the group. If a clique is found, all edges of the clique are deleted and a new vertex, the switch, is inserted to which all nodes are connected with a latency of half of the latency the clique was searched. If a node has edges to all members of the clique with the same latency, these edges are deleted and a new edge to the switch with a latency of the old edge minus the latency between the nodes of the clique and the switch is inserted. Since new cliques can be created by this procedure, it has to be searched until no new clique is found. A pseudocode of the algorithm can be found in Algorithm 2.

Cliques of size two are only replaced by a switch if both nodes in the clique are connected to at least one other node and the connection between the first node of the clique and the other node is equal to the connection between the second node of the clique and the other node, so that the switch has at least three links.

Example

As an example, assume the latencies in table 2.2 are measured.

	B	C	D	E	F	G	H	I
A	2	2	4	4	4	4	4	4
B		2	4	4	4	4	4	4
C			4	4	4	4	4	4
D				2	2	4	4	4
E					2	4	4	4
F						4	4	4
G							2	2
H								2

Table 2.2.: Example: measured latencies

The algorithm is performed on data:

1. All edges are inserted so that the basic model in Figure 2.8 is received.

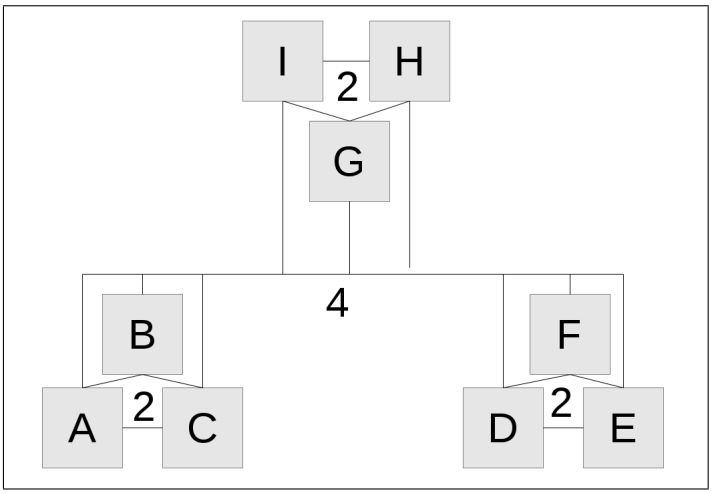

Figure 2.8.: Example: the basic model (simplified from the fully meshed)

2. Now cliques where all edge have the latency 2 or 4 are searched. There is no clique with latency 4, but there are three cliques with latency 2 - {A, B, C}, {D, E, F} and {G, C, I} - so that switches connecting them are inserted (see Figures 2.9 and 2.10).

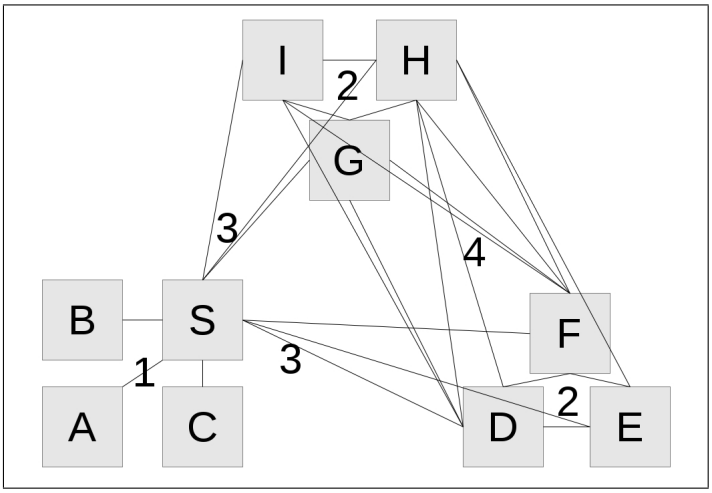

Figure 2.9.: Example: One clique replaced by switch

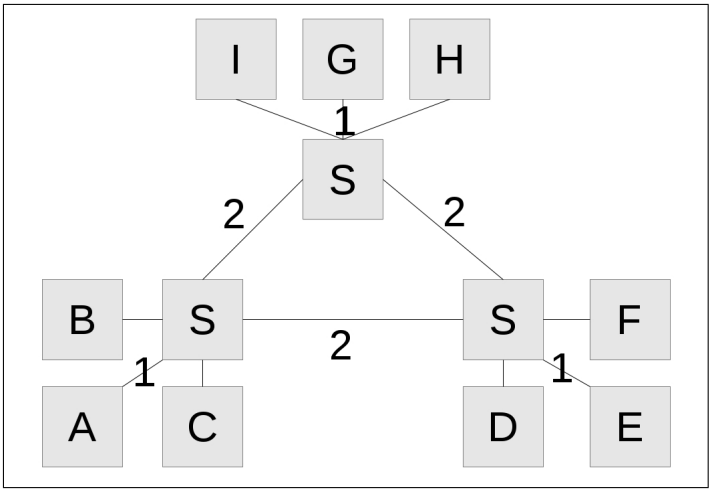

Figure 2.10.: Example: Three cliques replaced by switches

3. Now a new clique with the latency 2 is found consisting the switch vertices (see Figure 2.11).

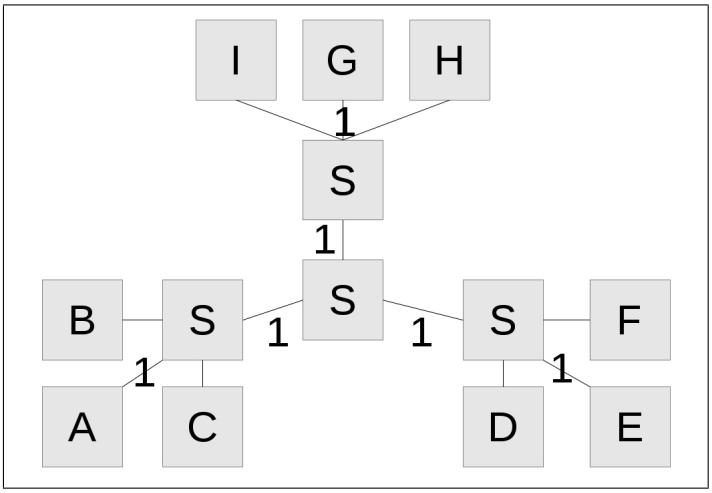

Figure 2.11.: Example: Finished topology

Another condition is that cliques must either consist only of nodes or only of switches. Mixed cliques must no be allowed, as this can lead to errors. In the example above, let the measured latency between node of different switches be 3. After the first switch insertion, a clique containing switches and nodes will be found. Due to that, the edges of the switch to the third group will be deleted and transferred to the new switch, so that the three switches will not be fully-meshed, but have a line topology.

An easy way to get the types of latencies is to put them into a set, but if a set which compares roughly is used, it will depend on the order of insertion how many elements are inserted. For example three latencies A, B and C where A is equal to B and B is equal to C, but A and C are not equal. If they are inserted in the order A, B, C, the set will contain A and C, but if they are inserted in the order B, A, C or B, C, A, the set will contain only B. Another disadvantage is that an edge can belong to more than one type of latency at a time so that the result of the algorithm can depend on the order of execution.

A simple way to only find cliques where all edges have the same latency is to copy the graph and delete all edges not having the latency for which is searched and use a standard clique search algorithm. One might think that clique search is np-complete and therefore that multiple clique searches will need a lot of time, but it has to be recognised that the graph often decomposes into several parts by this procedure so that the effective size of the graph will be smaller and calculation will be sped up.

The idea of this step is that all node-to-switch edges basically have the same latency and that a switch adds the same latency to all connections. Due to that this might fail if the cluster is not homogeneous. The limitations of heterogenous clusters are shown in Section 2.2.5.

A case where the algorithm will fail is a topology where there are several “direct“

connections, for example node-to-node-edge and a connection including a switch, between two nodes (see Figure 2.12). This problem cannot be recognised by only using latencies.

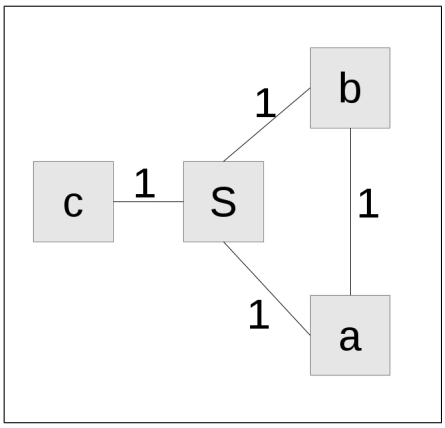

Figure 2.12.: A topology, the algorithm cannot detect

2.2.4. Elimination of virtual Switch-to-Switch Edges

```
D ← ∅;
foreach (u,v) ∈ E do
    if isSwitch(u) and isSwitch(v) then
        if Latency(u,v) = min_{w∈adj(v)}(Latency(u, w) + Latency(w, v)) then
            D ← D ∪ (u, v);
        end
    end
end
E ← E \ D;
```

Algorithm 3: Pseudocode: Elimination of virtual switch-to-switch edges

At last, it has to be checked if the switch-to-switch connections are actual connections by checking if the switch to switch connection has the same latency as the shortest path from the switch to the other switch without the connection. The algorithm is sketched in Algorithm 3.

For example, if we have a topology consisting of four switches connected as a cycle (Figure 2.13), the model for the switches will be fully connected (Figure 2.14) because when the switch-to-switch edges are created, the algorithm does not check if the latency of the edge is shorter than the latency of the shortest path. This has to be checked now and the mistakenly added edges have to be deleted.

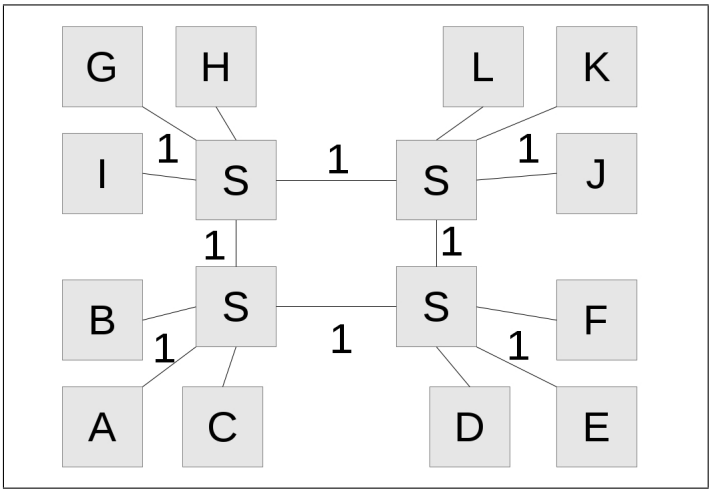

Figure 2.13.: Example: The actual topology

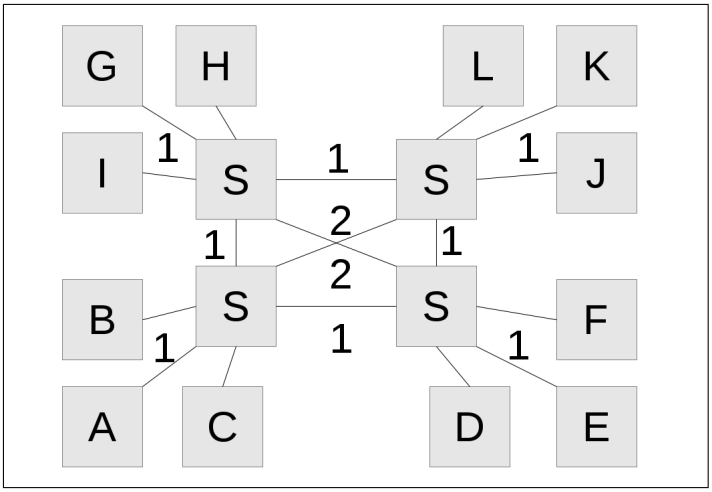

Figure 2.14.: Example: The computed toplogy after applying Algorithm 2

If cliques of size two are allowed, this step must be performed after every insertion of a switch. Otherwise, the diagonal edges connecting switches in 2.14 would be mistaken for a clique.

2.2.5. Heterogeneous Clusters

Although switch detection using clique search works primary on homogeneous clusters, there are many cases it works on heterogeneous clusters, too, for example, if a switch is only connected to other switches or nodes of the same kind.

Another example is a switch connected to at least two nodes of the same kind and to one different node. The different node will not be part of the clique, but as all nodes

of the clique will have the same connection to the other node, the connection will be transferred to the switch. If there are at least two nodes of one kind and at least two nodes of another kind, the algorithm will place one switch for every group and the edge connecting the switches will have the latency zero (Figure 2.15).This can be generalised to any number of groups of similar nodes leading to a clique of switches, all connected with latency 0. Due to that, such cliques should be rejoined.

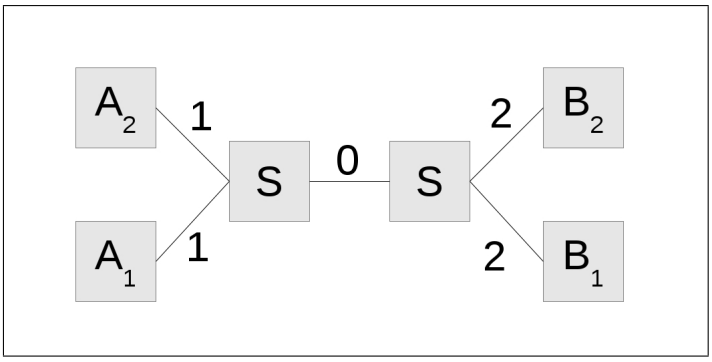

Figure 2.15.: Two groups of nodes interconnected by a single switch

As a last example, let there be four nodes connected with the latencies in table 2.3

Node	A_2	B	C
A_1	2	3	4
A_2		3	4
B			5

Table 2.3.: Example: Measured latencies

A clique of size two (A_1 and A_2) is found and replaced by a switch. This leads to the topology in Figure 2.16.

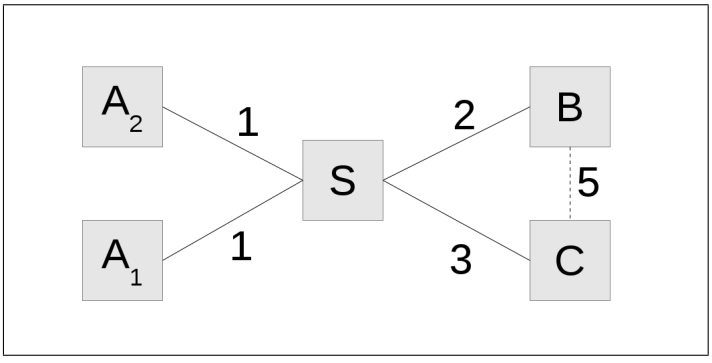

Figure 2.16.: Example: Two nodes of the same kind and two different nodes connected by a single switch

There is a false edge between B and C which can be recognised as a virtual edge, as its latency is the sum of the latencies connecting B and C to the switch. Generally, it should be checked if such edges exist and if they can be deleted assuming that they actually describe the connection to the switch, too.

2.3. Measuring Throughput

Now the model has been created, it can be used to measure the throughput of the edges simultaneously. There are three types of edges, node-to-node edges, note-to-switch edges and switch-to-switch edges, which will be treated individually because of two reasons: First, it reduces the complexity of the algorithm. Second, if a constant amount of data is used to measure, the time needed will be more similar for edges of the same type than for edges of different types which reduces waiting time.

The algorithm uses virtual time cycles are introduced as a unit for analysis: An arbitrary pairing of nodes can be measured in every time cycle so that the amount of needed time cycles has to be minimised to get best parallelisation.

2.3.1. Node-to-Node Edges

The best case scenario for this type of edges is the ring or the line topology which can be measured with $\mathcal{O}(1)$ cycles e.g. Figure 2.17. First, the connections (a,b) and (c,d) are measured and then the connections (b,c) and (d,e).

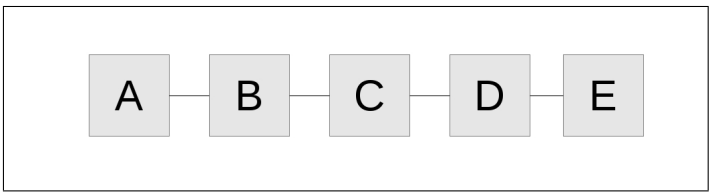

Figure 2.17.: Line topology which can be measured in $\mathcal{O}(1)$

The worst case scenario is a topology where one node is connected to all other nodes (Figure 2.18). In this scenario $\mathcal{O}(n)$ cycles are needed. Even in case of a fully-meshed topology $\mathcal{O}(n)$ cycles are needed, because this is equal to measuring the throughput of all possible pairings which can be done in $\Theta(n)$ which will be proven in the next section.

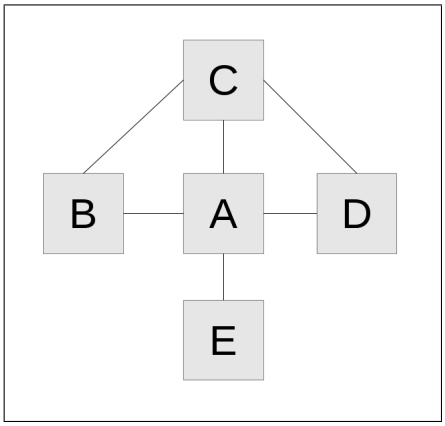

Figure 2.18.: A topology that can only be measured in $\mathcal{O}(n)$

Generally the number of needed time cycles depends on the maximum node degree so that edges connecting a node with a high node degree should be measured first. An algorithm should be able to measure the worst and best case scenario with the theoretical optimal amount of time cycles as the following algorithm:

First, the graph is copied and all switches and edges connected to switches are removed. There is an attempt to use every node as a starting point for a path which is build with an deep-first-search like algorithm. If the number of nodes of the path is odd, the last element is removed and the throughput between the first and the second element, the third and the fourth element and so on is measured. After measuring, the connections are deleted from the copy of the graph and the procedure is repeated unless all connections have been measured, which means that every edge has been deleted so that $\max_{v \in V'} degree(v) = 0$. A parameter for optimisation is the order in which the nodes are used as starting points and in which the nodes are tried to be put into the path. In both cases, the node with the highest node degree should be preferred. The algorithm is sketched in Algorithm 5.

Input: v, path, not
if $|\{w \in adj(v) | w \notin path\}| = 0$ **then**
    if *length of path is even* **then** append v **to** not;
    else append v **to** path;
end
else
    $max \leftarrow arg\, max_{w \in adj(v) | w \notin (path \cup not)} degree(v)$;
    append v **to** path;
    recursivePath(max, path, not);
end

Algorithm 4: Pseudocode: recursivePath

```
V' ← { v ∈ V | isSwitch(v) = false } ;
E' ← { (u, v) ∈ E | isSwitch(u) = false ∧ isSwitch(v) = false} ;
path ← not ← ∅;
while max_{v∈V'} degree(v) > 0 do
    max ← arg max_{v∈V'|v∉(path∪not)} degree(v);
    recursivePath(max, path, not);
    if |path ∪ not| = |V'| then
        for i ← 0 to  half length of path do
            measureThroughput( (path[2i], path[2i+1]) );
            E' \ (path[2i], path[2i + 1]);
        end
        path ← not ← ∅;
    end
end
```

Algorithm 5: Pseudocode: Measuring Throughput of Node-to-Node Edges

2.3.2. Node-to-Switch Edges

As long as each node is only connected to one switch, all node-to-switch connections of different switches can be measured simultaneously. It has to be checked first if this condition is fulfilled.

The problem in measuring the throughput of node-to-switch connections is that the throughput cannot be measured directly. It is only possible to measure the throughput with another node connected to the switch and the throughput will be the maximum of both node-to-switch connections. Due to that, one has to find the node with the maximum throughput so that the throughput of a measured connection is limited by the other edge.

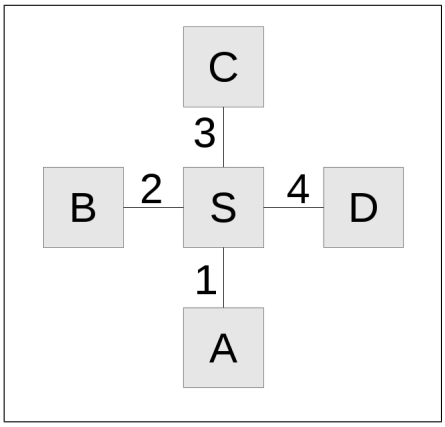

Figure 2.19.: Example Graph, throughput annotated

To find the edge with the maximum throughput in Figure 2.19 one has to measure all connections in the worst case. If, for example, all connections with the node A are measured, every result will be 1 so that the only information gained is that node A has the edge with the worst throughput to the switch (or all edges have the same throughput). If then the node B is chosen, the only information gained is that B has the edge with the second worst throughput to the switch and so on.

It would be possible to reduce the amount of needed cycles, if the maximum throughput of the switch was high enough so that connections can be measured parallel. This can be checked easily with three measurements: If the throughput of all except two nodes plus the throughput between the two nodes is equal to the throughput of all nodes measured simultaneously, the maximum throughput is high enough, otherwise the connections have to be measured sequentially.

To measure the throughput, we have two alternatives:

1. If we assume that the throughput of an edge is correlated with the latency of the edge, meaning that edges with a lower latency have a higher throughput, we can use our knowledge of the latencies to optimize the measurement of the throughput by sorting the nodes by latency and then measure the throughput between the first and second node, third and fourth node and so on and after that measure the throughput between the second and third, fourth and fifth node, and so on. In this case the throughput can be measured in $\mathcal{O}(1)$ (parallel) or $\mathcal{O}(n)$ (sequential).

2. We measure the throughput of all possible pairings. Let t_{ij} be the measured throughput between node i and j. Then the throughput between node i and the switch is $\min_j t_{ij}$. This can be done in $\mathcal{O}(n^2)$ (sequential) or in $\mathcal{O}(n)$ (parallel) with the following scheme:

   All nodes connected to a single switch are put into a group. The group is divided into two groups, the left containing nodes 1 to $\lfloor \frac{n}{2} \rfloor$, the right containing nodes $\lfloor \frac{n}{2} \rfloor + 1$ to n. The throughput of all possible pairings - there are $\lceil \frac{n}{2} \rceil$ possible pairings - of the two groups is measured. The procedure is repeated for each group until the groups only contain one or two nodes. If a group contains two nodes, the throughput between the two members is measured.

Example

This is an example with ten nodes 1...10. In every step the throughput between nodes that are closely written on the same height is measured. The skip indicates the next step of the algorithm. Altogether there are 11 steps needed. (In the last step the throughput between nodes 4 and 5 and nodes 9 and 10 is measured.)

5	10	5	6	5	7	5	8	5	9
4	9	4	10	4	6	4	7	4	8
3	8	3	9	3	10	3	6	3	7
2	7	2	8	2	9	2	10	2	6
1	6	1	7	1	8	1	9	1	10

	5		10				4		9			3		8	
2	4	7	9			2	3	7	8		2	5	7	10	
1	3	6	8			1	5	6	10		1	4	6	9	

			5				10					4			9
1	2	3	4	6	7	8	9			1	2	3	5	6	7	8	10

Analysis

The amount of needed cycles is described by the following recurrence:

$$T(n) = \begin{cases} 0, & \text{if } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + \lceil \frac{n}{2} \rceil, & \text{if } n > 1. \end{cases} \tag{2.3}$$

The recurrence of the cases n > 1 has the form $T(n) = aT(\frac{n}{b}) + f(n)$, where a = 1, b = 2 and f(n) = $\lceil \frac{n}{2} \rceil$ so that we can use the Master theorem. [CSRL01] Since f(n) = $\Omega(n^{\log_b a+\epsilon})$ for $0 < \epsilon < 1$ and $af(\frac{n}{2}) \leq cf(n)$ for $\frac{1}{2} \leq c < 1$, we can apply the third case of the Master Theorem so that $T(n) = \Theta(f(n)) = \Theta(\lceil \frac{n}{2} \rceil) = \Theta(n)$.

2.3.3. Switch-to-Switch Edges

The problem measuring switch-to-switch edges is that these edges cannot be measured directly. One has to measure the throughput between nodes so that the switch-to-switch edge which has to be measured is on the path between the nodes. Since the throughput of node-to-switch edges is usually lower than than the throughput of switch-to-switch edges, the throughput between several nodes has to be measured. It has to be guaranteed that the edge which has to be measured is on the used path between the nodes, but it is not known in cases of several possible paths which one will be used.

For a given edge, there are two possibilities:

1. If the graph decomposes into two unconnected sub-graphs by removing the edge, then as many nodes as possible of the two sub-graphs can be paired. This can be checked easily using a search algorithm, such as depth-first search.

2. If the sub-graphs are connected even without the edge, the pairings will have to be chosen carefully, for example by using only the nearest nodes.

   For example, if in Figure 2.20 the edge between S_3 and S_4 shall be measured, one would use the nodes J, K and L as one group and the nodes D, F, E as the other group. Adding node I to group one and for example node B to the other group would be unsafe, because it is not known on which path the data between the nodes is transferred.

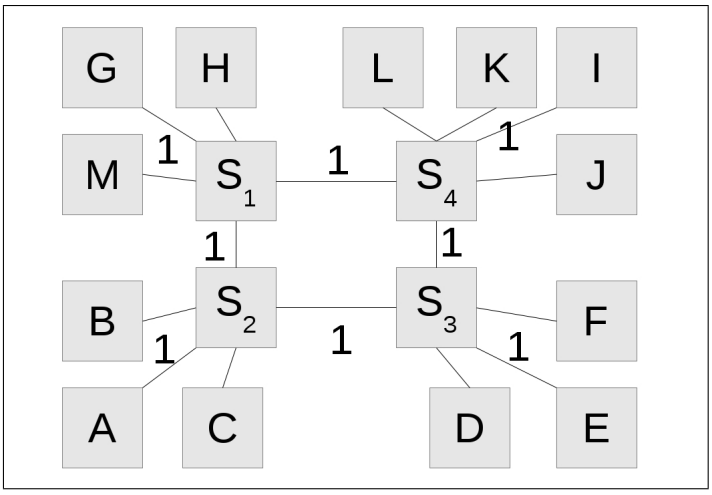

Figure 2.20.: Example for measurement of switch-to-switch edges

   Another possibility would be to assume that the data will be transferred on the shortest path and calculate all unique shortest paths.

Since it is not known how many nodes are needed to measure the maximum throughput of the edge, one has to use as many nodes as possible which makes the parallelisation difficult. A simple parallelisation would be a greedy scheme where first an edge which has to be measured is chosen and the needed nodes and involved switches are determined. Then there is an attempt to find another edge which has to be measured and where no node or switch needed for measurement is needed for the measurement of the first edge, too.

If we assume that all edges of the switches will have the same throughput, we only have to measure one switch-to-switch edge per switch. Ideally, we should choose the edge for a switch so that the minimum of the two groups between which we measure the throughput is maximised. Sometimes we will choose the same edge for different switches so that the number of edges needed to be measured will be smaller than the number of switches.

2.3.4. Maximum Throughput of Switches

The maximum throughput of a switch can be measured similarly. If the graph decomposes in more than two parts by removing the switch, one will have to consolidate the parts so that two groups are containing approximately the same number of nodes which corresponds to the partition problem. The partition problem is np-complete, but in practice the amount of groups is small so that the solution can be computed fast. If the graph does not decompose by removing the switch, the nearest nodes have to be chosen for the measurement.

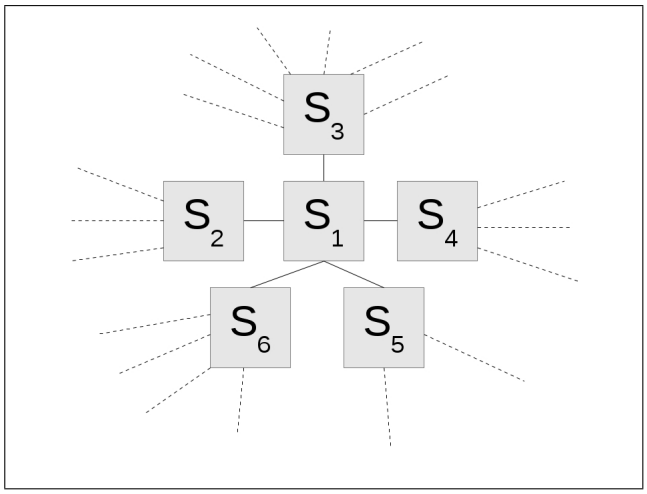

Figure 2.21.: Example for maximum throughput of switch - the dashed lines represent the nodes

Example

In the topology sketched in Figure 2.21 let there be no connection between the switches except of S_1. To find the best pairing one has to solve the partition problem for the multiset (2, 3, 3, 4, 6). One solution is (2,3,4) for the first group and (3, 6) for the other group so that all nodes of the switches S_2 and S_3 should be put into one group and all nodes of the switches S_4, S_5 and S_6 should be put in the other group.

2.4. Correcting Latencies

When the throughput has been measured, the measured latencies can be corrected. Since the latencies of node-to-switch and switch-to-switch edges cannot be measured directly, but they can be calculated from the latencies between the nodes.

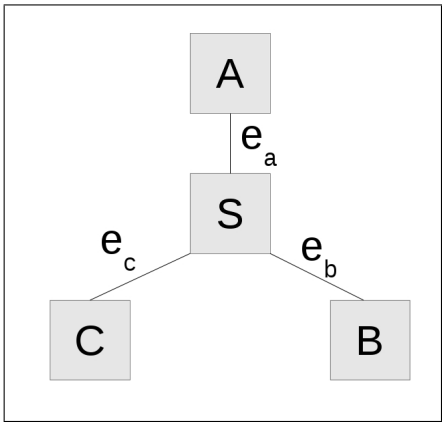

Figure 2.22.: Example: Three nodes connected to a switch

Because of the additivity of latencies the latencies of the edges (e_a, e_b and e_c) in Figure 2.22 have to satisfy the following equations, where l_{ab} is the latency between the nodes a and b:

$$\begin{aligned} e_a + e_b &= l_{ab} \\ e_a + e_c &= l_{ac} \\ e_b + e_c &= l_{bc} \end{aligned}$$

In this case, we can calculate the latencies of the edges by solving this system of linear equations, which can also be written as a matrix equation:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} e_a \\ e_b \\ e_c \end{pmatrix} = \begin{pmatrix} l_{ab} \\ l_{ac} \\ l_{bc} \end{pmatrix} \tag{2.4}$$

Generally we can write

$$Ax = b \tag{2.5}$$

where A is the matrix as described containing all possible paths between nodes, x the vector of the latencies of the edges that have to be calculated and b the vector of the latencies.

By using this method, we can use most of the measured latencies we had to measure for basic model generation. For every connection which is not a direct node-to-node edge, we can add the path to matrix and the measured latency to the vector b. In cases where several possible paths exist, there are two possibilities: Either all possible paths whose summed latency are roughly the measured latency are added or only unique paths are added.

To determine b, we have to correct the measured latencies by subtracting the time required to transfer the data of the message. There are two possibilities:

1. According to the assumption that the throughput on a path is the minimum throughput of the edges on the path. The throughput can be calculated by determining the minimum of the throughput of all edges in the path . The actual latency l is $l = l' - \frac{s}{t\,min}$ where l' is the measured latency, s the overall size of the massage and t_{min} the minimum throughput of the path belonging to l'.

2. A store-and-forward model is assumed. The message is first stored at every node and then sent forward. Let n be the number of edges of the path. The actual latency l is $l = l' - \sum_{i=1}^{n} \frac{s}{t\,i}$, where t_i is the throughput of edge i. [Kun13]

To compute x there are three possibilities depending on the number the number of rows of A and the number of edges. Note that the number of rows of A must not be $\frac{n(n-1)}{2}$ e.g. the latencies could be remeasured with fewer measurements.

1. The number of rows of A is smaller than the number of edges. The linear equation system cannot be solved. The algorithm does not create topologies of that kind.

2. The number of rows of A is equal to the number of edges. Equation 2.5 is a linear equation system. If A is regular, it can be solved (for example by using LU-decomposition). The question to be asked is if A is always regular, meaning it is always possible to solve the corresponding linear equation system or if there might be cases in which no unique solution exists. It is known from mathematics that a matrix is regular if and only if the matrix has full rank. For a topology similar to Figure 2.22, but with n nodes, the matrix can be written as the following:

$$\begin{pmatrix} 1 & 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & 1 & 0 & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & 1 & 1 \\ 1 & 0 & \dots & \dots & 0 & 1 \end{pmatrix} \tag{2.6}$$

   Let $M_{(i)}$ the i^{th} row vector of the matrix. It can be easily seen that if n is even, then

$$A_{(n)} = \sum_{i=1}^{n} (-1)^{i+1} A_{(i)} \tag{2.7}$$

   meaning that in this case the last row is linear dependent of the other rows/columns, meaning that the matrix has not full rank so that A is not regular and no unique solution of the linear equation system exists. In that case there are two possibilities: Either one could measure additional connections and perform a regression or one could measure different connections. However, the first method increases the

required amount of measurements and the needed computation time. The second method has the problem that all edges are no longer measured with the same time and accuracy.

3. The number of rows of A is greater than the number of edges. In general, there is no x so that $Ax - b = 0$ thus we have to find

$$\min_{x \in \mathbb{R}} \|Ax - b\| \tag{2.8}$$

for a convenient norm, meaning we have to perform a linear regression to find the best latencies of the edges of an additive model. For the euclidean norm the problem is known as Ordinary Least Squares (OLS) regression and there are many algorithm to solve it like QR-decomposition.

A regression delivers a coefficient of determination, $R^2 \in [0, 1]$, describing how good the calculated latencies can explain the measured ones. In case of a perfect model R^2 is 1, in case of no linear dependence R^2 is 0. This can be used to validate the determined topology. If R^2 is not high, this can indicate a wrong topology. Note that a high value of R^2 does not prove a correct model. The higher the amount of variables (edges), the higher is the value of R^2. A fully meshed topology will always have an coefficient of determination of 1, as there are as many variables as observations (measured latencies). [MPV12]

2.5. Remeasuring Latencies

If the latencies for the model creation have been measured shortly, intending to reduce the time needed for the measurement, or if the latencies were determined simultaneously so that the measurements might have influenced each other, it might be of interest to remeasure the latencies. As it is no longer required to determine the latencies of all possible pairings, the time of a single measurement can be increased. Also it can be profited by the knowledge of the topology by using similar methods as they were used to measure the throughput.

The procedure for measuring the latency of **node-to-node edges** is exactly the same as the procedure for measuring the throughput of those edges.

In case of node-to-switch edges, the **node-to-switch edges** of different switches can be measured simultaneously but the node-to-switch edges of each switch should be measured sequentially to avoid measurement errors.

Since there are only two nodes needed for the measurement of a **switch-to-switch edge** (and not as many as possible like for the measurement of the throughput) the parallelisation scheme will be more efficient.

2.6. Output

Since graph drawing is very complex, no graphical output is implemented. There are several free graph drawing programs which can be used so that the graph of the model can be outputted as a .tgf file. The Trivial Graph Format is an easy graph format which is supported by nearly all graph programs and has the following structure:

A list of the unique vertex id and the label is followed by the hash symbol and a list of the edges with the pair of connected vertex ids and the label of the edge. For example:

```
1 v1
2 v2
#
1 2 l: 3 t: 10
2 1 l: 3 t: 10
```

2.7. Connection of Models

As mentioned the intra node model can be build in the same way as the inter-node model where the nodes are the CPU cores and the switches are points connecting edges. To connect the models, a neighboured node has to be chosen and all connection from this node to the cores have to be measured and inserted as connection from a virtual external vertex to the candidates in the step of basic model creation. After the creation of the intra-node model, the latencies of the connection between the external vertex and the other nodes are subtracted by the minimum of the latencies between the external vertex and other vertices.

The models can connected by replacing the external vertex with the vertex representing the node. If one vertices which were used to measure the latency for the inter-node model have a connection higher than zero to the vertex representing the node, the latency between the two vertices representing the node has to be corrected by subtracting the latency of the connection between core vertex and node vertex.

Using the .tgf file format the connection can easily be done using a simple perl script.

2.8. Summary

2.8.1. Summary of the Costs

Due to the clique search, the cost for the computation of the model out of the measured data will be $\mathcal{O}(\exp(n))$, where n is the number of nodes, sockets or cores depending on which model we are creating. However, these costs are unimportant because the constant

factor is usually extremely smaller than the constant of the measurement. The cost to perform a linear regression are $\mathcal{O}(r^3) = \mathcal{O}(n^6)$, where r is the number of rows, so that the (optional) step of recalculating the latencies by a linear regression can lead to a high runtime.

The costs for measuring the data to create the basic models are $\mathcal{O}(n^2)$ (sequentially measured) or $\mathcal{O}(n)$ (parallel measured).

The time required to measure the throughput of node-to-node edges is $\mathcal{O}(n_N)$, where n_N is the number of nodes having an edge to another node. (Even if we had a fully connected topology, we would be able to measure in linear time using the scheme described in Section 2.3.2).

To measure the node-to-switch edges, we need $\mathcal{O}(n_S)$, where n_s is the number of nodes having a connection to a switch.

Finally, the time needed to measure the switch-to-switch edges is - in the case that every edge of a switch is measured - $\mathcal{O}(|E_{S-S}|)$, where $|E_{S-S}|$ is the number of switch-to-switch edges, and $\mathcal{O}(S)$, where S is the number of switches, in the case that only one edges per switch is measured.

Summed up, if the connections to create the basic model are measured parallel and the throughput of only one switch-to-switch edge is measured, the required time for the necessary measurements is proportional of the number of nodes (because the number of switches depends of the number of nodes too).

2.8.2. Summary of the Chapter

In this chapter, we have seen all major aspects off the developed algorithm:

- The latency of all possible pairings of nodes is measured and the data is used to compute a basic model in a way that switches can be detected using standard clique search algorithms.
- Although the algorithm is designed for usage on homogeneous clusters, it is shown that the algorithm can detect some cases of heterogeneous clusters.
- A schedule to measure the throughput is developed by separating the problem into three problems which are easier to solve.
- By performing a linear regression, not only the latency of edges which cannot be measured directly can be computed, but also all measured latencies can be used.

The algorithm must now be validated with several test.

3. Validation

First, the three test systems are described (Section 3.1). The distribution of the batches of a measurement on the Magny node is analysed (Section 3.2.1). Descriptors to compare measurements are defined (Section 3.2.2) and used to compare different measurement methods (Section 3.2.3). The developed algorithm is tested with four test cases (Section 3.3). At last, a simple model to estimate the probability of failure is presented (Section 3.4). The chapter ends with a short summary (Section 3.5).

3.1. Test Systems

Three different systems are used for validation:

- The WR working group hosts a cluster with heterogeneous hardware for teaching and research. Amongst others, the cluster offers 10 dual-socket Intel Westmere compute nodes and a four-socket AMD Magny-Cours node with 12 cores per socket which are used for testing. The Westmere cluster is mainly used for teaching and student projects. Each node of it contains two Intel® Xeon® X5650 processors with 6 cores each. The nodes are all connected with a single switch so that the topology can be called a star or a tree. The operating system of the cluster is Ubuntu 12.04 and OpenMPI was used to compile the tool.
- The Blizzard is the supercomputer of the DKRZ (Deutsches Klimarechenzentrum). It consists of 264 IBM Power6-compute nodes with 16 Dual Core Processors per node. The nodes are connected with an Infiniband network. The operating system of the Blizzard is AIX 6.1 and the parallel environment (PE) was used.

3.2. Comparison of Measurement Methods

3.2.1. Analysis of the Distribution of the Batches

To analyse the distribution of the batches, the latency between the first and second core of the Magny node is measured with batches consisting of a single measurement and 50,000 repetitions. The five-number summary and the mean of all of the batches are calculated with R[1] (see Table 3.1).

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.951	1.061	1.071	1.130	1.101	32.820

Table 3.1.: Five-number summary of the distribution of a single measurement

[1] `http://www.r-project.org/` version 2.14.1

The difference between the minimum and the 3rd Quartile is small, whereas the maximum is very high. Although in this case mean and median are almost equal, the extremely high maximum can increase the mean, especially if a smaller number of repetitions is used. The difference between the minimum and the median is small, so that the minimum can be used to describe the whole measurement, too.

3.2.2. Descriptors of Deviation

Different methods to measure latency (sequential and parallel measurement/size of batches) have been described and will be compared in this section. To compare two datasets A and B, where A_i is the ith value of the n values in Set A, the following descriptors are used:

- mean deviation, defined as

$$MD(A,B) := \frac{1}{n}\sum_{i=0}^{n}(A_i - B_i) = \frac{1}{n}\sum_{i=0}^{n} A_i - \frac{1}{n}\sum_{i=0}^{n} B_i = mean(A) - mean(B) \quad (3.1)$$

- mean absolute deviation, defined as

$$MAD(A,B) := \frac{1}{n}\sum_{i=0}^{n}|A_i - B_i| \quad (3.2)$$

- quadratic mean deviation (also called root mean square), defined as

$$QMD(A,B) = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(A_i - B_i)^2} \quad (3.3)$$

- maximum deviation, defined as

$$MAXD(A,B) := A_x - B_x, \text{where } x = arg\,max_x|A_x - B_x| \quad (3.4)$$

3.2.3. Comparison of Different Measurement Approaches

The following measurements are measurements of all possible pairings of cores on a single node of the Blizzard.

Comparison of Repeated Measurements

First, three measurements where every single value is the median of 10,000 measurements and all possible pairings are measured parallel are compared. The unit of all values is μs. Measured intra-cpu latency is approximately $1.15\mu s$, inter-cpu latency $1.3\mu s$ and inter-socket latency varies from $1.7\mu s$ to $2.1\mu s$.

compared measurements	MD(A,B)	MAD(A,B)	QMD(A,B)	MAXD(A,B)
1 and 2	0.0006	0.0077	0.0100	0.0350
1 and 3	-0.0053	0.0111	0.0180	-0.0840
2 and 3	-0.0059	0.0106	0.0176	-0.0970

Table 3.2.: Comparison of repetitions of a measurement

Since the MD is very small, the means of the two measurements are almost equal. The MAD and QMD are not big, too. Only the maximum deviation is not very small, which can lead to problems, as the difference of latency between inter- and intra-processor communication is only 0.15 μs.

Comparison of batched and individual measurement

Next, different approaches to measure the latency of a single batch are compared. The batches of the first measurement (A) consist only of a single measurement, whereas the batches of the second measurement (B) consist of a number of repetitions so that the batch needs approximately 100 μs.

repetition	MD(A,B)	MAD(A,B)	QMD(A,B)	MAXD(A,B)
1	0.0655	0.0655	0.0662	0.1182
2	0.0628	0.0628	0.0638	0.0965
3	0.0685	0.0686	0.0712	0.1694

Table 3.3.: Comparison of individual (A) and batched execution (B)

In all three cases the MD and MAD are almost equal indicating that the mean is simply shifted to a higher latency. A possible explanation is that the measurement of time itself needs some time and, in the case of batches consisting of many measurements, the mean of the batch is less influenced as a batch consisting only of a single measurement (remember the mean is calculated by determining the time of the whole batch and dividing by the number of measurements in the batch).

Comparison of sequential and parallel measurement

At last sequential (A) and parallel measurement (B) of all possible pairings is compared.

compared measurements	MD(A,B)	MAD(A,B)	QMD(A,B)	MAXD(A,B)
individual execution	-0.0294	0.0306	0.0357	-0.1080
batched execution (time 100 μs)	-0.0340	0.0348	0.0406	-0.1197

Table 3.4.: Comparison of sequential (A) and parallel (B) measurement

It can be seen that parallel measurement leads to a small, but unimportant increase of latency, even maximum deviation is not significantly higher than the maximum deviation of simple repeated measurements.

3.3. Testing

3.3.1. Westmere Cluster

The measured inter-node latencies can be found in table 3.5 and the measured intra-node latencies in table 3.6.

nodes	2	3	4	5	6	7	8	9	10
1	53.253	53.252	53.263	53.377	53.362	54.658	53.457	53.647	53.335
2		53.127	53.022	53.348	53.437	53.406	54.417	53.311	53.167
3			53.394	53.409	53.176	53.954	53.318	54.457	53.453
4				53.356	53.176	54.039	53.349	53.462	54.387
5					54.263	53.632	53.634	53.343	53.403
6						54.504	54.595	54.614	54.543
7							53.522	53.377	53.875
8								53.366	53.156
9									53.328

Table 3.5.: Measured inter-node latencies of the Westmere nodes, median of 100 repetitions, batch time 10,000 μs, all values in μs

cores	2	3	4	5	6	7	8	9	10	11	12
1	0.445	0.457	0.455	0.443	0.447	0.844	0.827	0.866	0.858	0.863	0.854
2		0.452	0.460	0.451	0.447	0.892	0.875	0.910	0.914	0.905	0.906
3			0.454	0.456	0.459	0.858	0.838	0.905	0.911	0.877	0.908
4				0.464	0.460	0.858	0.846	0.886	0.887	0.879	0.880
5					0.446	0.858	0.850	0.889	0.891	0.891	0.876
6						0.853	0.845	0.881	0.886	0.879	0.878
7							0.451	0.456	0.452	0.451	0.448
8								0.445	0.441	0.438	0.437
9									0.452	0.454	0.450
10										0.439	0.446
11											0.445

Table 3.6.: Intra-node latencies of the Westmere nodes, median of 100 repetitions, batch time 10,000 μs, all values in μs

First, a model of the inter-node topology was created and then a model of one node. Finally, the intra-node model was duplicated and the two models were connected. The

result was visualised with yEd[2].

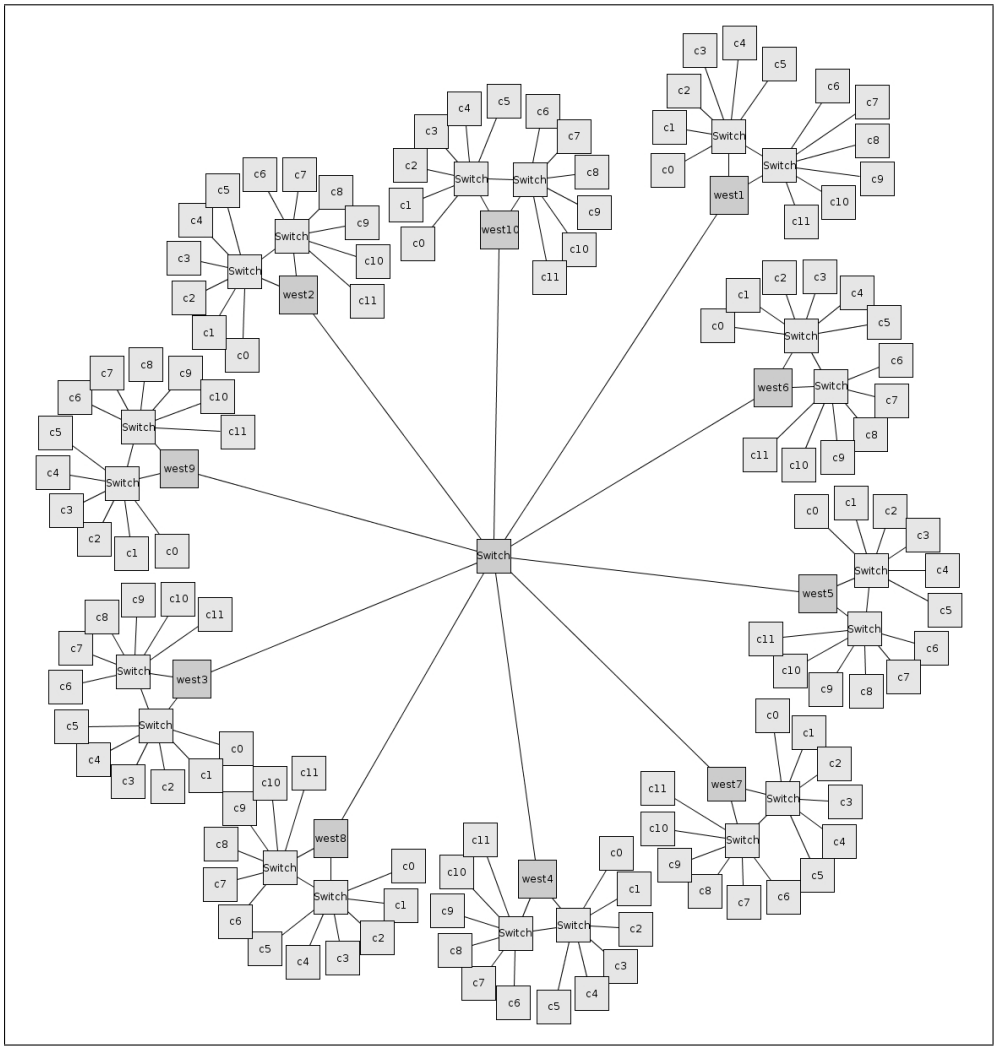

Figure 3.1.: Determined Topology of the Westmere Cluster

As it can be seen, the correct topology was found: In the middle is the central switch connecting the nodes represented by a virtual vertex. Connected to the virtual vertices are the two "switches" representing the two sockets each connected to six cores.

3.3.2. Magny Node

Since only a single node is available, only the intra-node model can be generated and tested. The problem determining the topology of the Magny node is that the latencies of inter- and intra-socket communication are very similar. (see Table 3.7 and 3.8)

[2] `http://www.yworks.com/en/products_yed_about.html`

sockets	2	3	4
1	1.204	1.232	1.171
2		1.213	1.189
3			1.171

Table 3.7.: Inter-socket latencies measured as the median of 1,000,000 single measurements, all values in μs

cores	2	3	4	5	6	7	8	9	10	11	12
1	1.1145	1.101	1.109	1.098	1.134	1.160	1.144	1.138	1.141	1.159	1.155
2		1.050	1.108	1.060	1.114	1.111	1.107	1.096	1.109	1.108	1.128
3			1.063	1.050	1.122	1.171	1.145	1.159	1.164	1.172	1.157
4				1.046	1.106	1.130	1.103	1.086	1.104	1.120	1.131
5					1.073	1.150	1.138	1.153	1.128	1.147	1.154
6						1.135	1.147	1.130	1.144	1.143	1.164
7							1.038	1.067	1.054	1.044	1.045
8								1.045	1.046	1.031	1.034
9									1.052	1.049	1.052
10										1.037	1.052
11											1.035

Table 3.8.: Intra-socket latencies (of the first socket) measured as the median of 1,000,000 single measurements, all values in μs

As it can be seen, the intra-socket latency varies from 1.035 μs to 1.172 μs and the inter-socket from 1.171 μs to 1.232 μs. Due to that, the algorithm cannot distinguish between the two kinds of connections and this leads to a false topology.

If the epsilon is chosen high, the algorithm assumes all connections have roughly the same latency, the topology will be a star with a single switch. In cases where the latencies are very similar, it can be argued that the differences are not important enough to justify placement decisions because by optimizing the distribution of task to the sockets only a small time decrease can be gained.

3.3.3. Single Blizzard Node

The intra-node topology of the Blizzard Node with 16 dual-core processors is complex (see Figure 3.2). There are four quad-node building blocks, each consisting of four fully-meshed processors and each processor is linked to two other processors in a way that it is possible to reach every other processor with at most two other processor on the path. There are four different types of edges:

- the edges between core and processor (type A)

- the edges in the block(type B)
- the edge between processors which are not in the same block (type C)

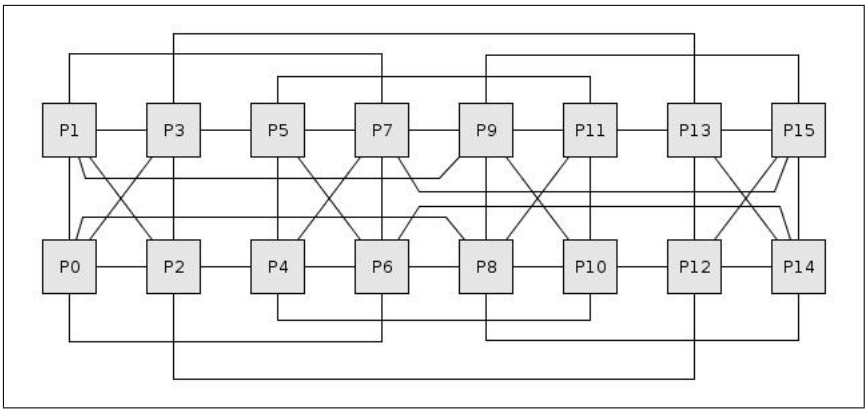

Figure 3.2.: Inter-processor topology of the Blizzard nodes [Lud11, p. 437]

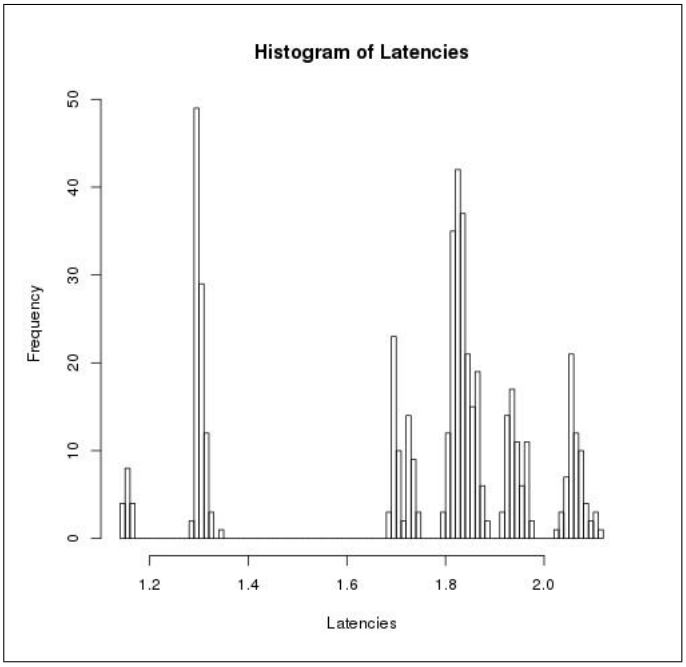

Figure 3.3.: 100-bin Histogram of measured Blizzard latencies, latencies in μs

All possible core-to-core latencies are measured. A histogramm of them can be found in Figure 3.3. The six groups (from left to the right) can be identified as

1. latencies between cores of the same processor (2 x type A)
2. latencies between cores of processors being in the same block (2 x type A + type B)

3. latencies between cores of processors not being in the same block (2 x type A + type C)

4. latencies between cores of processor with one other processor on the path (2 x type A + type B + type C)

5. same as fourth group (explanation below)

6. latencies between cores of processors with two other processors on the path (2 x type A + 2 x type B + type C)

Actually, group 4 and 5 are actual the same, because the gap between the groups is very small and the assumption that they were different would lead to contradictions:

- The latencies of the connection between processor 0 and processors 10-13 (see Figure 3.9) would belong to different groups, although they are all connected in the same way (type B + type C).
- To explain the last group, one had to assume that the connection between processors 0 and 14 is two edges of type C, but the latency between processors 0 and 14 is equal to the latency between processors 0 and 15 (see Figure 3.9). However, these processors are not connected with two edges of type C.

p	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1.31	1.30	1.31	1.82	1.84	1.71	1.83	1.73	1.84	1.87	1.95	1.86	1.94	2.10	2.05

Table 3.9.: Measured latencies between first cores of the processors (p)

The algorithm almost computes the correct topology (see Figure 3.4). The fully-meshed areas are mistakenly replaced by a switch, but a switch, efficient enough so that different nodes connected to it do not influence each other, is equivalent to a fully-meshed topology.

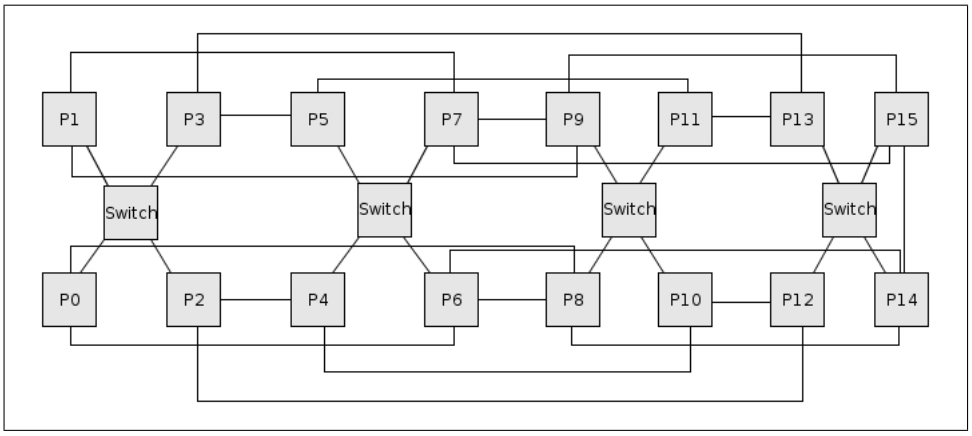

Figure 3.4.: Computed topology of the Blizzard node

3.3.4. Four Blizzard Nodes

If all possible core-to-core latencies are measured parallel and the cores are on different nodes, the intra-node measurements are influenced (see Figure 3.5). This is surprising as the parallel measurement of intra-node latencies alone does not lead to measurement errors (see Section 3.2). However, this can easily be prevented by separating the model generation of inter- and intra-node model and connecting them after the generation.

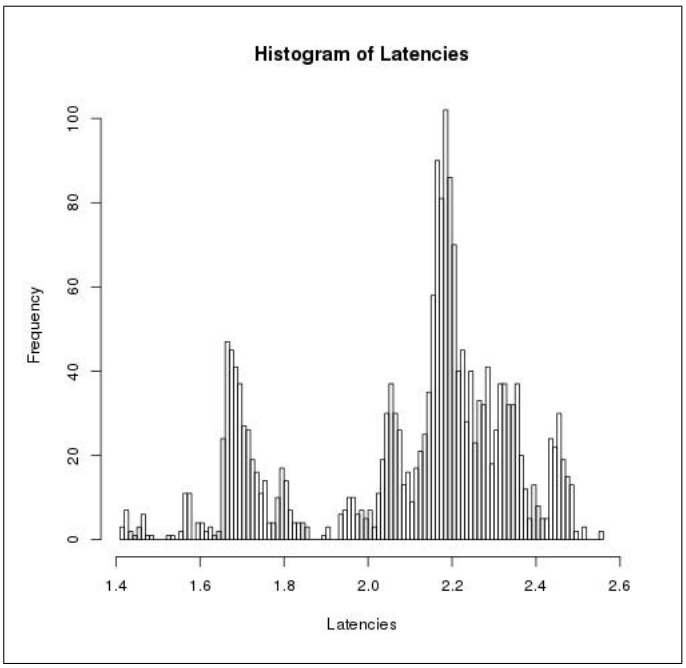

Figure 3.5.: 100-bin Histogram of intra-node latencies, all possible core-to-core latencies measured parallel, latencies in μs

3.4. Challenges

When the latencies are measured and the topology is created, a single wrongly determined latency of an edge can lead to a completely wrong topology. As simplification we can assume that every wrongly measured latency will lead to a false topology (which is untrue, because if the measured latency is compared to the shortest path latency and the measured latency is higher than the shortest path latency, which is an measurement error, this will not lead to a wrong topology).

Let p be the probability that the latency of a single connection is measured within the tolerances and n be the number of nodes (cores or sockets). The probability P that all $\frac{n(n-1)}{2}$ connections are measured without any error is

$$P = p^{\frac{n(n-1)}{2}}. \tag{3.5}$$

p \n	12	32	128	256
99%	51,5%	0,7%	0%	0%
99,9%	93,6%	60,1%	0,03%	0%
99,99%	99,3%	95,2%	44,4%	3,4%

Table 3.10.: P calculated for different error probability (p) and numbers of processors (n)

In table 3.10, it can be seen that the latency of a connection has to be determined with a very high precision for high amounts of nodes/cores.

The required precision - and therefore p - is directly dependent on ϵ. If there are very small differences and due to that only small tolerances are allowed (meaning ϵ is very small), the precision needed to provide a good probability p will be high or the probability to measure the right value with a given precision will be small and the algorithm will fail.

3.5. Summary of the Chapter

By analysing the distribution of the batches the importance of robust descriptors is made clear as high outliers are possible.

The comparison of different measurement methods showed that intra-node latencies can be determined in parallel with only a small measurement error. This is very important as this provides a linear scaling. It is also shown that the measurement with batches should be preferred, as the measurement without batches leads to a small methodical error.

The tests on the Westmere cluster and the single Blizzard node demonstrate that he correct topology can be found in many cases, but the test on the Magny node and the four Blizzard nodes demonstrate also that the accurate measurement of the latencies is vital.

Al last, it is illustrated that a single wrongly measured latency can lead to a wrong topology.

4. Conclusion and Future Work

A conclusion (Section 4.1) and a perspective of future work (Section 4.2) are given.

4.1. Conclusion

A completely new approach was tested with this work: Since both - topology and performance characteristics - are demanded, the determination is combined. First, the latencies are measured and the topology is determined by analysing them. Then, the topology is used to support the measurement of the throughput. Although in the first step the topology is not known and simply the latency of all possible edges is measured, all of the gained data can be used by performing a linear regression. The algorithm is designed highly abstractly, so that a single algorithm can be used to determine e.g. inter- or intra-node topology.

One of the main advantages of the tool - the latencies are measured and the topology is determined on the side by analysing the latencies - is also a disadvantage: A single measured latency can lead to a wrong topology, so that the schedule of the throughput measurement will be incorrect, too. However, as long as the latencies can be measured with high precision (relative to the differences of the latencies), the algorithm can find the right topology in many cases as seen on the Westmere cluster and can even detect the complex topology of the Blizzard node. The computation of the topology of the Magny-cours node only failed as the latencies could not be measured precisely enough.

Some examples of cases where the algorithm fails to determine the correct topology were shown. Nevertheless, no test went wrong due to errors of the algorithm, but there were only few test cases thus it cannot be ensured that these theoretical limitations are unimportant in practice.

Although the algorithm provides linear scaling (of measurement time), this can lead to a high runtime, if the number of cores or nodes is very high. The amount of CPUs in high-performance-computers or even desktop pc has been extremely increasing. It is an interesting fact that if the constants of algorithms (here the required time for a measurement of a connection) do not decrease, someday there will be a point where every non-constant scaling algorithm will be impractical. This leads to the challenging necessity of developing algorithms with a constant need of time.

4.2. Future Work

There are several things that should be improved:

- In this work, only point-to-point communication was regarded and collective communication (a communication with more than two participated points) was completely ignored, although collective communication is very important in practice.
- Since the probability of finding the right topology depends mainly on the probability of measuring the latencies correctly, the technique to measure latencies should be improved in order to make it more robust and accurate.
- Furthermore, the parallelisation scheme when measuring the throughput of switch-to-switch edges is very simple and often provides a non-optimal solution.
- The usage of a set to determine the groups of latencies allows to demonstrate the algorithm but has many disadvantages in real test cases as seen at the test with the Blizzard. A much better solution is to use approaches from cluster analysis to determine a strict partitioning clustering.

  For example, one could group the latencies by sorting the measured values and determine the bounds of the groups by checking whether the gap between two neighboured values is bigger than a defined constant. The latencies of the inputfile can be replaced by the mean of the group they belong to. If the latencies are corrected by using the actual latencies, this step has no further effect than to simplify the computation of the topology.
- The behaviour on heterogeneous clusters was not tested. Also a method to determine the topology of an arbitrary heterogeneous cluster should be developed.

Bibliography

[BK73] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[Inf] Interconnect analysis: Infiniband and 10gige in high-performance computing. `http://www.hpcadvisorycouncil.com/pdf/IB_and_10GigE_in_HPC.pdf`. Accessed: 2014-03-16.

[Kun13] Julian Kunkel. *Simulation of Parallel Programs on Application and System Level*. Phd thesis, Universität Hamburg, 07 2013.

[Lud11] Thomas Ludwig. Skript zur Vorlesung Hochleistungsrechnen. `http://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2011_2012/hr-1112.pdf`, October 2011. Accessed: 2014-03-17.

[MPV12] D.C. Montgomery, E.A. Peck, and G.G. Vining. *Introduction to Linear Regression Analysis*. Wiley Series in Probability and Statistics. Wiley, 2012.

[PD07] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach, Fourth Edition (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann, 4 edition, March 2007.

List of Algorithms

List of Figures

List of Tables

A. Short Manual of the Tool

A.1. all-against-all

This program can be used to measure the latencies of all possible process-to-process connections. It has to be executed parallel and it has to be ensured that the threads are bound to cores and in case core-to-core connections are measured that the thread with rank one is bound to core one and so on. The results are printed to the console. The available options are:

- -s: measure sequentially instead of parallel
- -c: prints "c[rank]" instead of hostname
- -r: how many batches are measured to determine the latency of a single connection. The measured values are the median of the batches.
- -t: set minimum time of a single batch

A.2. model

This program uses the described algorithm to generate the topology from the output of all-against-all.x. The inputfile is the output of all-against-all. The topology is printed to the console as an .tgf file. To compare values the method of a relative comparison is used.

Options are:

- -e: The value of ϵ

A.3. throughput

This program can be used to measure the throughput of edges in a topology. It has to be started parallel and the threads have to be bound to the cores/nodes in the same order as they were bound to measure the latencies. The inputfile is the .tgf file of the toplogy. A .tgf file with the measured throughput inserted is printed out to the console.

A.4. regression

The inputfiles are the .tgf file of the topology and the output file with the measured latencies. The measured latencies will be corrected by throughput recalculated using a linear regression. To link the executable, the GNU Scientific Library is required. The available options are:

- -s: the size of the data packet sent to determine the throughput
- -r: how often every measurement is repeated. The final value is the median of the measurements.
- -b: use MPI_SendReceive

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Vordorf, den 18. März 2014 .....