TECHNISCHE UNIVERSITÄT DRESDEN

ZENTRUM FÜR INFORMATIONSDIENSTE
UND HOCHLEISTUNGSRECHNEN
PROF. DR. WOLFGANG E. NAGEL

Komplexpraktikum “Paralleles Rechnen”

Virtuelle Topologien

Friedrich Zahn
(Mat.-Nr.: 3966130)

Hochschullehrer: Prof. Dr. Wolfgang E. Nagel
Betreuer: Dr.-Ing. Robert Schöne

Dresden, 18. März 2021

Inhaltsverzeichnis

1 Aufgabenstellung 3

2 Benchmark-Umgebung 3
2.1 Software . . . 3
2.2 Hardware . . . 3

3 Lösungsansatz 4

4 Implementierung 5
4.1 Bedienungshinweise . . . 5

5 Ergebnisse 6
5.1 Kommunikation innerhalb einer Node . . . 6
5.2 Kommunikation zwischen Nodes . . . 9

Literatur 13

A Programmcode 15

B SLURM SBATCH-Skripte 16

1 Aufgabenstellung

Bauen Sie eine skalierbare virtuelle Topologie in Form eines 2-D-Torus auf, und testen Sie alle vier Kommunikationsrichtungen auf dem Bull HPC Cluster (Taurus)!

- Es sind die MPI-Funktionen zum Aufbau kartesischer Strukturen sowie die Kommunikationsrichtungen up, down, left, right zu verwenden. In einer ersten Messreihe soll für alle parallelen Kommunikationskanäle in up-Richtung ein voller Durchlauf über alle beteiligten Cores realisiert werden, wobei der Rank jedes Cores aufsummiert und die Gesamtdurchlaufzeit gemessen wird.
- Testen Sie Ihr Programm bis zu einer Coreanzahl von 16 sowohl Inter-Node als auch Intra-Node!
- Wiederholen Sie die Messungen für die Kommunikationsrichtungen down, left und right!

2 Benchmark-Umgebung

2.1 Software

Die von mir entwickelten Programme wurden auf dem Taurus-Cluster mit den dort vorhandenen Tools kompiliert. Zum Einsatz kamen GCC Version 7.3.0 mit OpenMPI 3.1.1 und OpenMP 4.5. Die build-Umgebung wurde mit cmake 3.11 generiert.

2.2 Hardware

Alle Messungen wurden auf dem HPC-Cluster „Taurus“ des ZIH durchgeführt. Um die Vergleichbarkeit von Messungen sicherzustellen, wurde für alle Benchmarks die „haswell“-Partition genutzt, in welcher alle Nodes über 2 Intel Xeon CPUs, Modell E5-2680 v3, mit jeweils 12 Kernen verfügen[1]. Untereinander sind die Nodes per Infiniband verbunden. Hyperthreading ist auf diesen Nodes deaktiviert, darüber hinaus wurde mittels SLURM die exklusive Nutzung der Nodes angefordert, um Verzerrungen durch anderweitig mitgenutzte Nodes auszuschließen.

3 Lösungsansatz

N-dimensionale Tori sind eine gängige switch-freie Netzwerktopologie im Bereich des verteilten Hochleistungsrechnen. Sie können dabei als Kompromiss zwischen hierarchischen Topologien (Stern o.ä.) und Vollvermaschung gesehen werden, die Vorteile beider Ansätze miteinander verbinden[3].
Ein eindimensionaler Torus ist dabei schlicht eine geschlossener Ring aus Nodes. Ein 2D Torus-Interconnect kann hergestellt werden, indem alle Nodes des Netzwerkes zunächst in einem Rechteck-Grid angeordnet werden, und zunächst alle Nodes mit ihren direkten Nachbarn verbunden werden. Darüber hinaus werden alle Nodes die an den Kanten liegen, jeweils mit dem gegenüber liegenden Ende ihrer Zeile oder Spalte verbunden - an den Ecken ist beides vonnöten. Diese Anordnung ist in Abbildung 1 visualisiert.

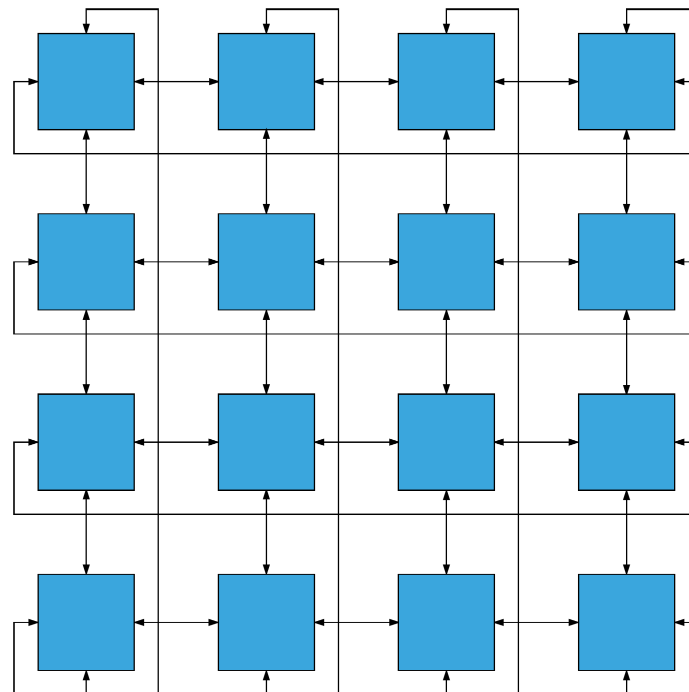

Abbildung 1: Visualisierung eines 4x4 2D-Torus. Übernommen aus [3].

Dieses Verbindungsschema bietet verschiedene Vorteile, so verfügen etwa - je nach gewählter Dimension - alle Nodes über die selbe Anzahl an Verbindungen, es kann also homogene Hardware in allen Positionen eingesetzt werden. Bei parallelisierbaren Datenübertragungen entlang einer Dimension steht von jeder Nodes zu ihrem Nachbarn in dieser Richtung die volle Bandbreite der Verbindung zur Verfügung, es gibt keine Flaschenhälse wie in hierarchischen Strukturen. Obwohl eine Torustopologie eine hohe Ausfallunempfindlichkeit bietet - von jeder Node gibt es eine Vielzahl möglicher Wege zu jeder anderen - ist sie deutlich skalierbarer als eine in dieser Hinsicht vergleichbare vollvermaschte Anordnung, welche sehr schnell eine praktisch nicht handhabbare Anzahl an Node-Verbindungen erfordert.
Die MPI-Funktionalitäten zum Aufbau kartesischer Strukturen bieten alle nötigen Funktionalitäten, um den geforderten 2D-Torus aufzubauen. MPI-Ranks werden in einer virtuellen Topologie die entsprechenden Koordinaten zugeordnet, und Nachbarknoten lassen sich unter Vorgabe einer Dimension und Schrittweite direkt ermitteln. Mittels periodischer Randbedingungen lassen sich dabei die geforderten Torus-Eigenschaften herstellen.

4 Implementierung

Das in C geschriebene Programm ist in der Lage, beliebige zweidimensionale Tori darzustellen und darin zu kommunizieren. Unter der Vorraussetzung, dass genügend MPI-Ranks allokiert wurden, wird zunächst mittels `MPI_Cart_create` die vom Nutzer angeforderte Topologie hergestellt. Dabei ist reordering deaktiviert, um eine deterministische Rank-Koordinaten-Zuordnung zu erhalten und mittels `periods = [1, 1]` werden die periodischen Randbedingungen aktiviert.
Die Ring-Kommunikation in der jeweiligen geforderten Richtung wird pro Zeitmessung so oft ausgeführt, wie in der entsprechenden Dimension Nodes vorhanden sind. Dabei schicken in jedem Takt alle Nodes die kumulierte Summe der Ranks an ihren jeweiligen Nachbarn, jedoch wird nur auf den Nodes einer Kante die Summe tatsächlich ausgewertet und die Zeit für den Durchlauf erfasst. Die Zeiterfassung und -speicherung erfolgt auf Nanosekunden genau mittels `clock_gettime(CLOCK_MONOTONIC, ...)`. Um ein Warm-Up der Kommunikationskanäle und -hardware zu erlauben, werden die ersten 10 Messungen verworfen.
Der gesamte Programmcode findet sich im Anhang als Listing 1.

4.1 Bedienungshinweise

Das ausführbare Programm erwartet vom Nutzer 3 Argumente: Die Ausdehnung des Torus in der Left-Right-Dimension (Zeilen), die Ausdehnung in der Up-Down-Dimension (Spalten), sowie zuletzt die gewünschte Kommunikationsrichtung kodiert als 0 (right), 1 (left), 2 (up) oder 3 (down). Dabei wird vom Programm erwartet und geprüft, dass mindestens so viele MPI-Ranks wie das Produkt der Dimensionen vorhanden sind.
Das Programm führt die einkompilierte Anzahl (10^6) an Messungen durch, und schreibt die gemessene Zeit sowie die Summe der Ranks in als CSV in eine Datei mit dem Namen `kppr_3_rank_<n>.out`, sofern der jeweilige Rank in der gewählten Richtung auf der „unteren“ Kante liegt, also einen der verfügbaren parallelen Kommunikationskanäle repräsentiert.
Ist eine bestimmte Verteilung der virtuellen Topologie auf Nodes im Cluster gewünscht, muss dies über eine entsprechende SLURM-Konfiguration sichergestellt werden. Die für die Messungen hier verwendeten SBATCH-Skripte befinden sich im Anhang als Listing 2 für die Intra-Node-Konfiguration und als Listung 3 für die Inter-Node-Konfiguration.

5 Ergebnisse

5.1 Kommunikation innerhalb einer Node

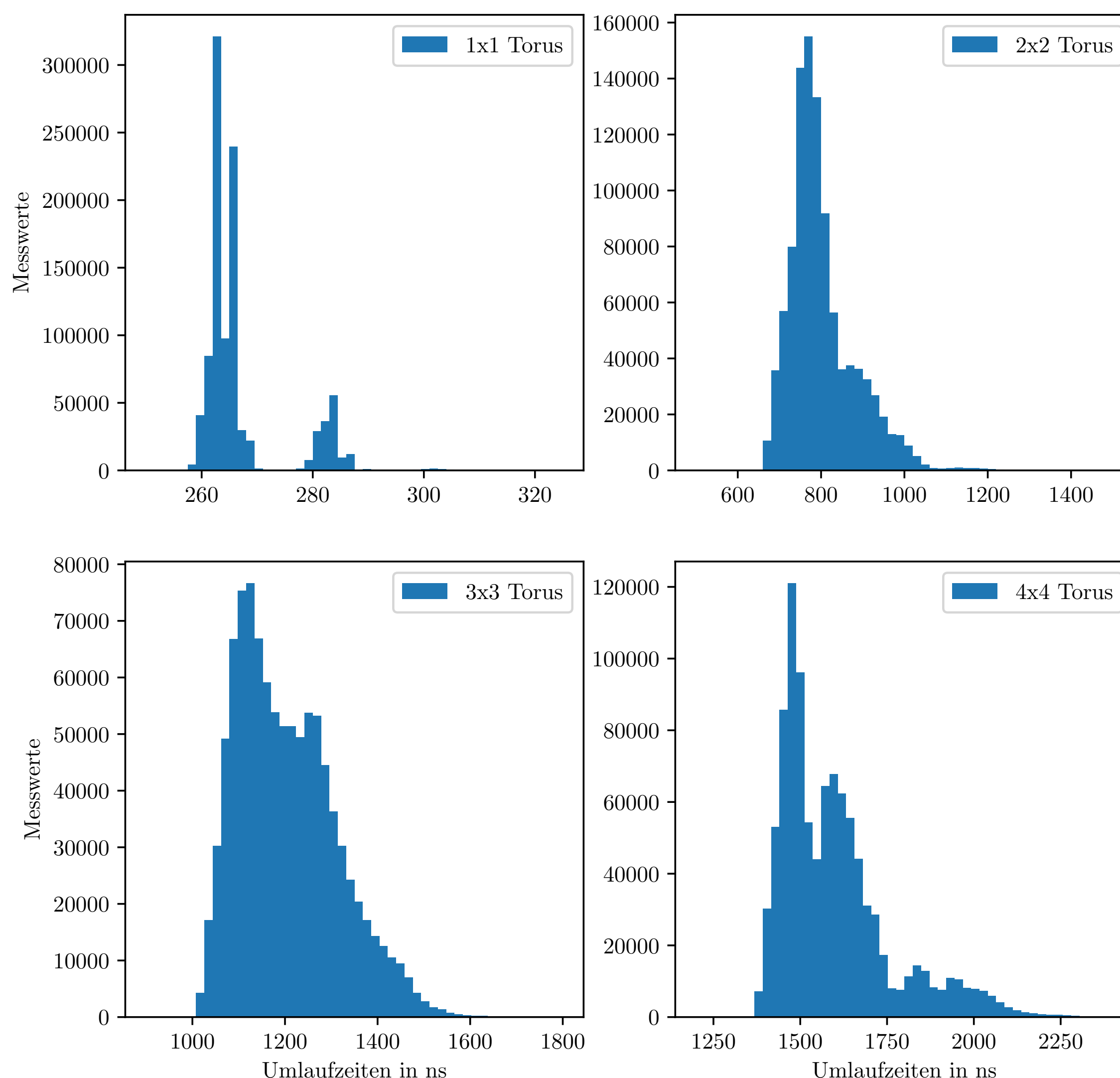

Abbildung 2: Histogramme der Umlaufzeiten für verschiedene Torus-Größen bei Kommunikation in Up-Richtung innerhalb einer physischen Node. Es wurden jeweils 10^6 Messwerte am Eck-Knoten mit Rank 0 erfasst. Nicht dargestellt sind einzelne Ausreißer nach rechts.

Die Verteilung der Messwerte für die Kommunikation Up-Richtung über 1, 2, 3 und 4 Hops in der Topologie (entsprechend der quadratischen Struktur des gewählten Torus) sind in Abbildung 2 dargestellt. Wie zu erwarten steigen die Umlaufzeiten mit wachsender Zahl an Knoten, die traversiert werden müssen, im Schnitt um ca. 400 ns pro zusätzlichem Hop. Betrachtet man die Verteilungen, ist jedoch auffällig, dass es sich augenscheinlich nicht um eine Gauß-Verteilung o.ä. handelt. Physikalisch bedingt existiert eine gewisse Mindest-Latenz, die nicht unterschritten werden kann. Dementsprechend ist die Verteilung

nach links klar abgegrenzt, und eine Vielzahl der Messwert liegt nur wenig über diesem Minimum. Im Gegensatz dazu gibt es nach oben kein Limit, und die Verteilung läuft nach rechts deutlich sanfter aus. Einzelne Ausreißer unter den Messungen liegen zum Teil mehrere Größenordnungen über dem Minimum. Damit lassen sich die Messwerte nicht sinnvoll als Gauß-Verteilung oder ähnlichen Modellen charakterisieren und die Angabe üblicher statistischer Größen wie insbesondere der Standardabweichung ist wenig aussagekräftig. Eine mögliche Beschreibung wäre die Pareto-Verteilung[2], welche ebenfalls einen minimalen Wert definiert, und abfallend nach rechts ausläuft. In diesem Sinne werden die Messungen für die jeweiligen Parameter (Größe des Torus, Kommunikationsrichtung, Intra- vs Inter-Node-Kommunikation) im Weiteren hauptsächlich durch die Angabe des Minimums, des Durchschnitts und des Medians charakterisiert. Je nach Anwendungsfall wären auch andere Prozentile von Interesse, die vollständigen Messdaten finden sich anbei.
Ein weiteres Feature, das ins Auge fällt, ist die Herausbildung von Neben-Maxima in der Verteilung, hier scheint eine Überlappung von mindestens zwei kategorisch unterschiedlichen Übertragungsverhalten vorzuliegen.

		Rank mod n			
Torus-Größe nxn		0	1	2	3
	Min	254			
1x1	Median	264			
	Mean	267			
	Min	657	675		
2x2	Median	782	799		
	Mean	803	825		
	Min	983	1004	1004	
3x3	Median	1188	1171	1125	
	Mean	1208	1210	1210	
	Min	1339	1328	1296	1387
4x4	Median	1563	1517	1625	1596
	Mean	1605	1574	1596	1627

Tabelle 1: Statistische Größen der gemessenen Durchlaufzeiten in Up-Richtung für die Kanten-Ranks über die betrachteten Torus-Größen hinweg, in Nanosekunden. Es wurden für jede Torus-Größe und Rank jeweils 10^6 Durchlaufzeiten erfasst.

In Tabelle 1 sind die charakteristischen Größen für die Messungen an allen Kanten-Knoten in Up-Richtung über die geforderten verschiedenen Torus-Größen hinweg angegeben.
Die Umlaufzeiten ausgehend von Rank 0 in die vier verschiedenen Richtungen für die vier betrachteten Torus-Größen sind in Tabelle 2 gelistet. Für die drei kleineren Tori ist keine signifikante Abhängigkeit von der Richtung erkennbar, anders jedoch beim 4x4-Torus. Hier ist klar sichtbar, dass sich für left/right deutlich höhere Median- und Durchschnitswerte ergeben, als in up/down-Richtung. Die naheliegende Erklärung hierfür ist, dass die verwendeten Haswell-Nodes über zwei Sockets zu je 12 Kernen verfügen, die $4 \times 4 = 16$ virtuellen Knoten des größten Torus müssen also auf beide Sockets verteilt werden. Bei fixer Knoten-Core-Zuordnung dürfte eine komplette Spalte des Torus auf dem zweiten Socket liegen, sodass die Kommunikation in Spalten (up/down) schneller passiert als Socket-überschreitend in Zeilen (left/right).

Richtung		Torus-Größe			
		1x1	2x2	3x3	4x4
left	Min	227	645	968	1562
	Median	238	744	1171	2139
	Mean	241	775	1197	2286
right	Min	211	643	955	1480
	Median	225	755	1116	2155
	Mean	229	782	1153	2290
up	Min	254	657	983	1339
	Median	264	782	1188	1563
	Mean	267	803	1208	1605
down	Min	224	670	986	1360
	Median	234	789	1178	1621
	Mean	238	825	1205	1646

Tabelle 2: Statistische Größen der gewonnenen Messwerte für den Prozess mit MPI-Rank 0 (Eck-Knoten „unten links“) über die betrachteten Torus-Größen und Kommunikationsrichtungen hinweg, in Nanosekunden. Es wurden für jede Größen-Richtungs-Kombination jeweils 10^6 Durchlaufzeiten erfasst.

5.2 Kommunikation zwischen Nodes

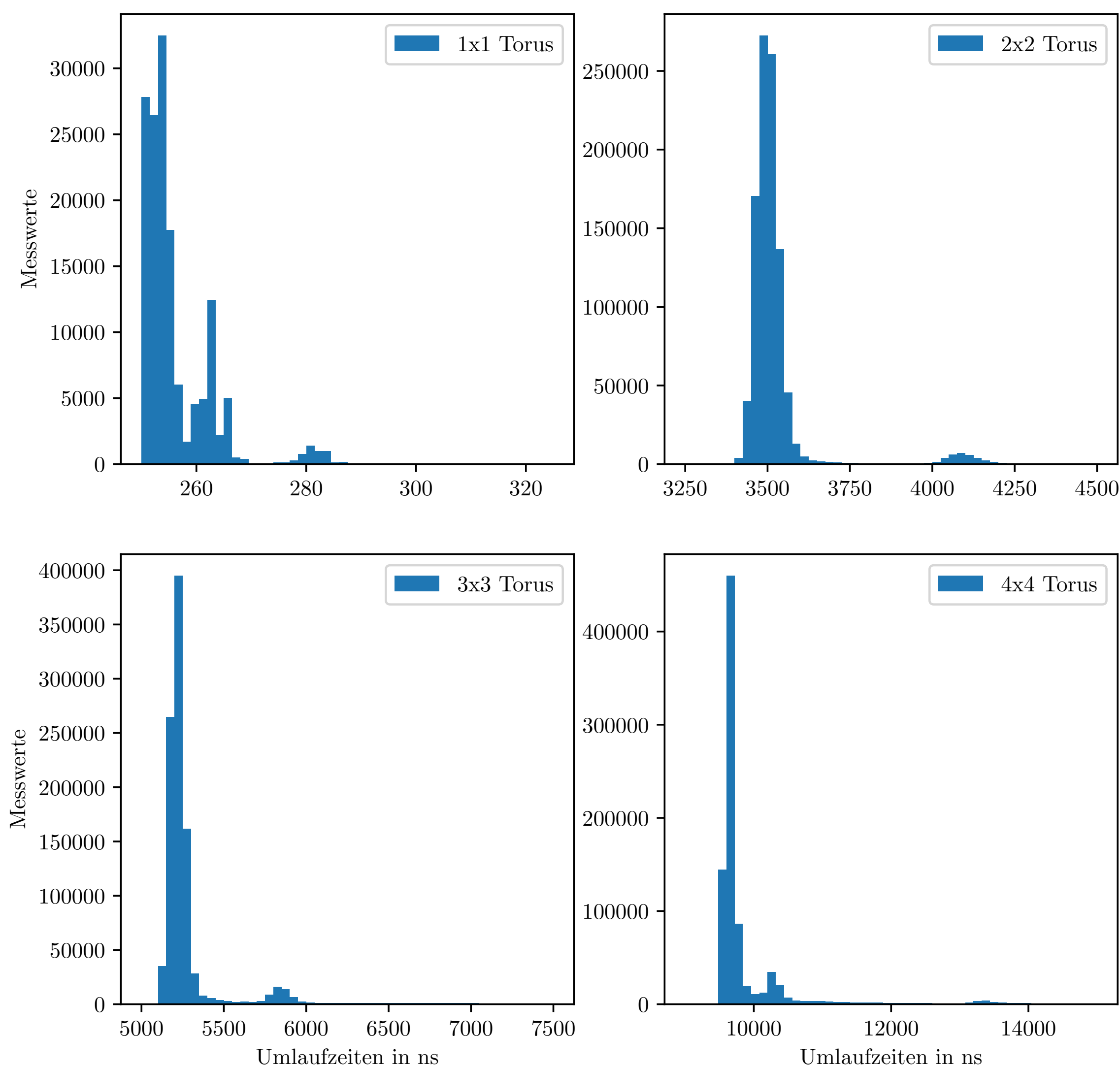

Abbildung 3: Histogramme der Umlaufzeiten für verschiedene Torus-Größen bei Kommunikation in Up-Richtung zwischen räumlich getrennten physischen Nodes. Es wurden jeweils 10^6 Messwerte erfasst. Nicht dargestellt sind einzelne Ausreißer nach rechts.

Für die Histogramme bei Kommunikation in Up-Richtung zwischen separaten Nodes im Cluster, dargestellt in Abbildung 3, ergibt ein ähnliches Gesamtbild wie bei der Kommunikation innerhalb einer physischen Node, nur dass die Latenz pro zusätzlichem Hop fast eine Größenordnung über den Umlaufzeiten innerhalb einer physischen Node liegt. Es ergeben sich aber wieder die bereits beobachteten nach links scharf abgegrenzten, nach rechts auslaufenden Verteilungen mit abgesetzten Neben-Peaks.
In Tabelle 3 sind die Ergebnisse für die Kanten-Ranks über die verschiedenen Torus-Größen hinweg aufgeführt. Es fällt ins Auge, dass es hier wesentlich größere Schwankungen innerhalb einer Kante im selben Torus gibt. Dies ist wenig überraschend, da die räumliche Verteilung der durch SLURM zugeteilten

Torus-Größe nxn		Rank mod n 0	1	2	3
1x1	Min	224			
	Median	234			
	Mean	238			
2x2	Min	3380	6096		
	Median	3501	6237		
	Mean	3536	6306		
3x3	Min	5063	4591	7685	
	Median	5223	4734	7874	
	Mean	5320	4834	8232	
4x4	Min	9430	9383	5735	6257
	Median	9680	9631	5941	6486
	Mean	13173	10158	6073	6623

Tabelle 3: Statistische Größen der gemessenen Durchlaufzeiten in Up-Richtung für die Kanten-Ranks über die betrachteten Torus-Größen hinweg, in Nanosekunden. Es wurden für jede Torus-Größe und Rank jeweils 10^6 Durchlaufzeiten erfasst. Die virtuellen Knoten liegen alle auf verschiedenen, räumlich getrennten physischen Nodes.

physischen Nodes im Cluster heterogen ausfallen kann, entlang bestimmter Zeilen oder Spalten können also unterschiedliche Wege innerhalb des Clusters zurückzulegen sein.
Der selbe Effekt ist auch bei der Messung über die verschiedenen Kommunikationsrichtungen hinweg zu beobachten, welche in Tabelle 4 aufgeführt sind.

Richtung		Torus-Größe			
		1x1	2x2	3x3	4x4
left	Min	216	3347	7722	11603
	Median	231	3464	7957	11947
	Mean	234	3497	10599	16080
right	Min	213	3353	7719	11597
	Median	225	3476	7948	12001
	Mean	229	3508	11319	15867
up	Min	224	3380	5063	9430
	Median	234	3501	5223	9680
	Mean	238	3536	5320	13173
down	Min	222	3378	5058	9490
	Median	234	3497	5221	9705
	Mean	237	3532	5315	9831

Tabelle 4: Statistische Größen der gewonnenen Messwerte für den Prozess mit MPI-Rank 0 (Eck-Knoten „unten links“) über die betrachteten Torus-Größen und Kommunikationsrichtungen hinweg, in Nanosekunden. Die virtuellen Knoten liegen alle auf verschiedenen, räumlich getrennten physischen Nodes. Es wurden für jede Größen-Richtungs-Kombination jeweils 10^6 Durchlaufzeiten erfasst.

Literatur

[1] *Hardware-Informationen zum Taurus-Cluster im ZIH-Kompendium.* https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareTaurus, . – [Online; Stand 18. Januar 2021]

[2] WIKIPEDIA: *Pareto-Verteilung — Wikipedia, Die freie Enzyklopädie.* https://de.wikipedia.org/w/index.php?title=Pareto-Verteilung&oldid=209259133, 2021. – [Online; Stand 18. März 2021]

[3] WIKIPEDIA CONTRIBUTORS: *Torus interconnect — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/w/index.php?title=Torus_interconnect&oldid=986566932, 2020. – [Online; Stand 17. März 2021]

A Programmcode

```c
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <time.h>
#include <string.h>

#define REPS 1000000

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int dim[2];
    dim[0] = atoi(argv[1]);
    dim[1] = atoi(argv[2]);

    int mode;
    mode = atoi(argv[3]);

    int direction, displacement;
    switch (mode){
        case 0:
            direction = 0;
            displacement = 1;
            break;
        case 1:
            direction = 0;
            displacement = -1;
            break;
        case 2:
            direction = 1;
            displacement = 1;
            break;
        case 3:
            direction = 1;
            displacement = -1;
            break;
        default:
            return EXIT_FAILURE;
    }

    if(dim[0] * dim[1] > world_size){
        printf("Not enough ranks available to distribute topology.\n");
        return EXIT_FAILURE;
    }

    int period[2] = {1, 1};
    int reorder = 0;
    MPI_Comm torus;

    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &torus);
    int coords[2];
```

```
61      MPI_Cart_coords(torus, world_rank, 2, coords);
62
63      FILE *outf;
64      char filename[255];
65      strcpy(filename, "kppr_3_rank_");
66      char rank[3] = {0};
67      sprintf(rank, "%d", world_rank);
68      strcat(filename, rank);
69      strcat(filename, ".out");
70      if ((direction == 0 && coords[0] == 0) || (direction == 1 && coords[1] == 0)){
71          outf = fopen(filename, "w");
72      }
73
74      struct timespec tStart, tEnd;
75
76      int count;
77      for (count = 0; count < REPS + 10; count++) {
78          int rank_sum = 0;
79          int recv_sum;
80          int destination, source;
81
82          clock_gettime(CLOCK_MONOTONIC, &tStart);
83          for (int step = 1; step <= dim[direction]; step++) {
84              MPI_Cart_shift(torus, direction, displacement, &source, &destination);
85              MPI_Sendrecv(&rank_sum, 1, MPI_INT, destination, 0, &recv_sum,
86                           1, MPI_INT, source, 0, torus, NULL);
87              rank_sum = recv_sum + world_rank;
88          }
89
90          clock_gettime(CLOCK_MONOTONIC, &tEnd);
91
92          long timeTaken = (tEnd.tv_sec - tStart.tv_sec) * 1e9 + (tEnd.tv_nsec -
    tStart.tv_nsec); // in nanoseconds
93
94          if ((direction == 0 && coords[0] == 0) || (direction == 1 && coords[1] == 0)
    ){
95              if (count > 9) fprintf(outf, "%ld, %d\n", timeTaken, rank_sum);
96          }
97      }
98      MPI_Finalize();
99      return EXIT_SUCCESS;
100 }
```

Listing 1: Der verwendete Programmcode für die durchgeführten Messungen.

B SLURM SBATCH-Skripte

```
1  #!/bin/bash
2
3  # 1 node:
4  #SBATCH --nodes 1
5
6  #SBATCH --ntasks-per-node 24
7
8  #SBATCH --cpus-per-task 1
9
10 #SBATCH --partition=haswell
11
12 #SBATCH --exclusive
13
14 # jobname:
15 #SBATCH -J kppr_3
16 #SBATCH --output %x.%j.log
```

```
17
18 # Load required modules
19 module load CMake/3.11.4-GCCcore-7.3.0
20 module load OpenMPI
21
22 cd /home/s4690420/kppr/task3/intra
23
24 for dim in 1 2 3 4
25 do
26   mkdir $dim
27   cd $dim
28   for dir in 0 1 2 3
29   do
30     mkdir $dir
31     cd $dir
32     srun --ntasks=$((dim * dim))  ../../../kppr_3 $dim $dim $dir
33     cd ..
34   done
35   cd ..
36 done
```

Listing 2: Das verwendete SBATCH-Skript um die Messungen innerhalb einer physischen Node durchzuführen.

```
1 #!/bin/bash
2
3 # 1 node:
4 #SBATCH --nodes 16
5
6 #SBATCH --ntasks-per-node 1
7
8 #SBATCH --cpus-per-task 1
9
10 #SBATCH --partition=haswell
11
12 #SBATCH --exclusive
13
14 # jobname:
15 #SBATCH -J kppr_3
16 #SBATCH --output %x.%j.log
17
18 # Load required modules
19 module load CMake/3.11.4-GCCcore-7.3.0
20 module load OpenMPI
21
22 cd /home/s4690420/kppr/task3/inter
23
24 for dim in 1 2 3 4
25 do
26   mkdir $dim
27   cd $dim
28   for dir in 0 1 2 3
29   do
30     mkdir $dir
31     cd $dir
32     srun --ntasks-per-node=1 --ntasks=$((dim * dim))  ../../../kppr_3 $dim $dim $dir
33     cd ..
34   done
35   cd ..
36 done
```

Listing 3: Das verwendete SBATCH-Skript um die Messungen zwischen separaten, räumlich getrennten physischen Nodes durchzuführen.