technische universität
dortmund

# Hochleistungsrechnen mit LiDO3

*Andreas Blume*

2. Oktober 2018

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für praktische Informatik (LS4)
http://ls4-www.cs.tu-dortmund.de

fakultät für
informatik
Lehrstuhl für praktische Informatik

# Inhaltsverzeichnis

# Kapitel 1

# Allgemeines

In diesem Kapitel wird LiDO3 kurz vorgestellt und dabei der technische Aufbau und die Benutzung des Clusters allgemein erläutert.

## 1.1 Aufbau und technische Daten

Die 3. Generation des Linux Cluster Dortmund (LiDO3) wurde am 16.5.2018 feierlich in Betrieb genommen. Seit dem wird es vom IT & Medien Centrum (ITMC) der Technischen Universität Dortmund betrieben und steht somit Wissenschaftlerinnen und Wissenschaftlern an der TU Dortmund, der Fachhochschule Dortmund und der Universitätsallianz Ruhr für das Hochleistungsrechnen zur Verfügung.[dTUD17, Seite 17]

Bei LiDO3 handelt es sich um ein auf der Architektur „Distributed Memory Computing“ basierendes heterogenes Compute Clustersystem, dass das Betriebssystem Linux und das Batchsystem Slurm verwendet. Das Cluster besteht aus 366 Rechenknoten (8160 CPU Cores), wobei mehr als 30 TB RAM auf die Knoten verteilt sind. Dadurch kann von jedem Knoten mindestens 64 GB RAM genutzt werden. Zur Speicherung von Daten nutzt LiDO3 das parallele Dateisystem BeeGFS (Link zur Produktseite: [Bee18]) mit 1,28 PB, dass von jedem Rechenknoten sichtbar ist. Zusätzlich besitzt jeder Knoten mindestens 2 TB lokalen Speicher. Die Netzwerkanbindung von LiDO3 wurde mit Infiniband QDR (40 Gbit/s) mit höchstens 1:3 Blocking realisiert.

In der Tabelle 1.1 werden die öffentlichen Knotenarten von LiDO3 kurz vorgestellt. Darüber hinaus verfügt das High Performance Compute Cluster LiDO3 über einen entsprechend vor verkabelten Rechnerschrankbereich, um in begrenzter Anzahl Rechnerknoten nachrüsten zu können. Diese Rechenknoten werden bei Bedarf vom LiDO3-Team in LiDO3 technisch integriert und mitbetreut.

| | **2 Sockel Knoten (316 Knoten)** | **4 Sockel Knoten (30 Knoten)** | **2 Sockel Knoten (20 Knoten)** |
|---|---|---|---|
| CPU | Intel® Xeon® E5-2640v4 (10 Kerne, 2.4 GHz) | Intel® Xeon® E5-4640v4 (12 Kerne, 2.1 GHz) | Intel® Xeon® E5-2640v4 (10 Kerne, 2.4 GHz) |
| L3 Cache | 25 MB | 30 MB | 25 MB |
| RAM | 64 GB | 256 GB (davon 2 Knoten mit 1 TB RAM) | 64 GB |
| Interconnect | Infiniband QDR (davon 72 Knoten mit 1:1 Blocking) | Infiniband QDR | Infiniband QDR |
| Sonstiges | – | – | 2x NVIDIA® Tesla® K40m |

**Tabelle 1.1:** Die Knotenarten von LiDO3 (öffentlich für alle LiDO3-Nutzer). [fIICdTD18]

Eine Übersicht über das gesamte Cluster bietet die Abbildung 1.1.

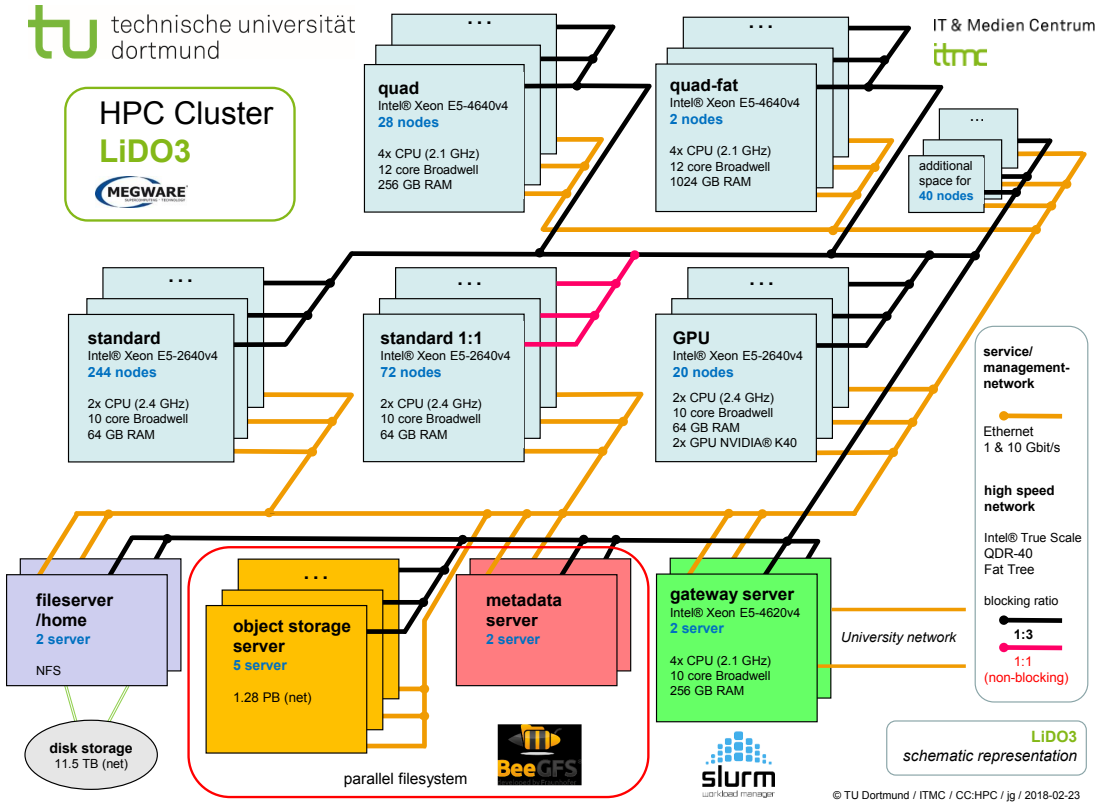


**Abbildung 1.1:** Der vereinfachte Aufbau von LiDO3. [fIICdTD18]

## 1.2 Das Dateisystem

In diesem Abschnitt werden die wichtigen Ordner und einige Möglichkeiten zur lokalen und remote Dateibearbeitung vorgestellt.

### 1.2.1 Wichtige Ordner

Im Folgenden werden die zur Verfügung stehenden Ordner eines LiDO3-Nutzers vorgestellt.

#### 1.2.1.1 Das zentrale Home Dateisystem ZFD

LiDO3 verfügt über 14 TB Festplattenspeicher für Home-Verzeichnisse. Jeder Nutzer besitzt sein eigenes Home-Verzeichnis: `/home/<username>` mit den folgenden Eigenschaften:

- Limit: 100000 Dateien und 32 GB Datenvolumen insgesamt
- Daten in diesem Ordner werden regelmäßig gesichert.

**Achtung:** Die Homeordner sind auf den Gatewayservern schreib- und lesbar gemountet und auf Rechnerknoten ausschließlich **lesbar**!

#### 1.2.1.2 Das zentrale parallele Dateisystem BeeGFS

LiDO3 verfügt über 1280 TB Festplattenspeicher für Work-Verzeichnisse. Jeder Nutzer besitzt sein eigenes Work-Verzeichnis: `/work/<username>` mit der folgenden Eigenschaft:

- Daten werden nicht gesichert, d.h. vollständiger Datenverlust möglich.

Demzufolge ist der Nutzer selbst verantwortlich für die Datensicherung. Ein Ausfall und damit einhergehender Datenverlust ist schon vorgekommen!

#### 1.2.1.3 Der Scratch Ordner

Da extrem viele und zeitintensive Dateioperationen mit besonders vielen kleinen Dateien BeeGFS stark belasten und somit zu langsameren Zugriffszeiten bzw. Ausführungszeiten eines Programms führen, gibt es auf jedem Rechenknoten den lokalen Ordner `/scratch`. Dieser Ordner befindet sich im lokalen Plattenspeicher eines Knotens und ist daher auch nur dort sichtbar. Die lokalen Platten von jedem Knoten haben eine Speicherkapazität von mindestens 2 TB, wobei dem scratch-Ordner fast die gesamte lokale Festplatte zur Verfügung steht (1,8 TB).

#### 1.2.1.4 Zusammenfassung

Um von den Vorteilen des scratch-Ordners zu profitieren, gibt es die folgende Empfehlung:

- Bei Jobstart jeweils Daten von `/home` nach `/scratch` kopieren.
- Daten anschließend in `/scratch` verarbeiten.
- Bei bzw. vor Jobende Daten wieder zurück von `/scratch` nach `/work` kopieren (nach `/home` geht nicht, da `/home` auf den Rechnerknoten nur lesbar gemountet!).

- Wichtige Ergebnisse später auf den Gatewayservern von `/work` nach `/home` kopieren, um von der regelmäßigen Datensicherung des Home-Verzeichnis zu profitieren.

| | **Gateway-server** | **Rechen-knoten** | **Bemerkung** |
|---|---|---|---|
| **home** | lesbar & schreibar | nur lesbar | • Limit: 100000 Dateien und 32 GB Datenvolumen<br>• Daten werden regelmäßig gesichert |
| **work** | lesbar & schreibar | lesbar & schreibar | • Daten werden nicht gesichert<br>→ Datenverlust möglich |
| **scratch** | – | lesbar & schreibar | • Lokaler Festplattenspeicher des Rechenknotens<br>• Größe: ca. 1,8 TB |

**Tabelle 1.2:** Zugriffsberechtigungen der LiDO3-Ordner im Überblick.

### 1.2.2 Dateien bearbeiten

Dateien können vorbereitend innerhalb von `/work` oder von `/home` nach `/work` kopiert werden. Dazu steht die Linux Shell Bash (Konsole) zur Verfügung.

Um Dateien zwischen Rechnern zu kopieren, existieren zwei Möglichkeiten:

- via Linux mit scp (Manpage: [SCP13])
- via Windows mit Hilfe einer SCP-Software: z.B. WinSCP, Filezilla, ...

Zur lokalen Dateibearbeitung kann ein Editor verwendet werden. Im LiDO3-Cluster stehen die folgenden Editoren zur Verfügung: vi, Emacs, Gedit, Nano.

**Tipp:** Zum Kopieren oder anderen Dateioperationen und zum Editieren steht auf den Gatewayrechnern der Midnight Comander zu Verfügung. Dazu den Befehl `mc` in der Konsole eingeben. Natürlich kann man die Dateien auch lokal bearbeiten und mit `scp` übertragen.

## 1.3 Zugang und Login

### 1.3.1 Beantragung der Nutzung

Angehörige der TU Dortmund beantragen die LiDO3-Nutzung im Serviceportal. Dort kann unter `IT-Dienste` → `Hochleistungsrechnen (LiDO)` ein Neuantrag für LiDO3 gestellt werden. Der blaue Pfeil in Abbildung 1.2 zeigt den Weg zum LiDO3-Neuantrag im Serviceportal. Wählt der Nutzer diesen Link, erscheint ein Formular, dass in Abbildung 1.3 zu sehen ist. Neben einem SSH Public Key (Generierungsanleitung siehe weiter unten),


**Abbildung 1.2:** Neuantrag LiDO3. [fIICdTD18]

muss zusätzlich der Grund der Inanspruchnahme, die Nutzungsdauer, sowie der Genehmiger angegeben werden. Der Genehmiger ist in der Regel ein(e) Hochschullehrer/-in oder Institutsleiter/-in. Dem Genehmiger wird anschließend der Antrag per E-Mail vorgelegt, um die Genehmigung zu überprüfen.

Die anderen Anwender aus dem Ressourcenverbund NRW beantragen den Zugang mit Hilfe eines Antrags, den sie zunächst ausfüllen und anschließend an die aufgeführte Adresse (Service Desk des ITMC) schicken. Das Formular kann unter folgendem Link heruntergeladen werden: `https://www.itmc.tu-dortmund.de/cms/de/dienste/hochleistungsrechnen/lido3/index.html`.

Anleitung zur Erzeugung und Nutzung des öffentlichen und privaten ssh-Schlüssels unter Linux:

- in das `.ssh`-Verzeichnis wechseln: `cd /.ssh`
- das folgende Kommando ausführen: `ssh-keygen -t rsa -b 4096`
- danach gewünschten Name des Schlüssels und Passwort eingeben

Eine Anleitung für Windows mit `puttygen` ist unter [fIICdTD18, Folie 11] zu finden.

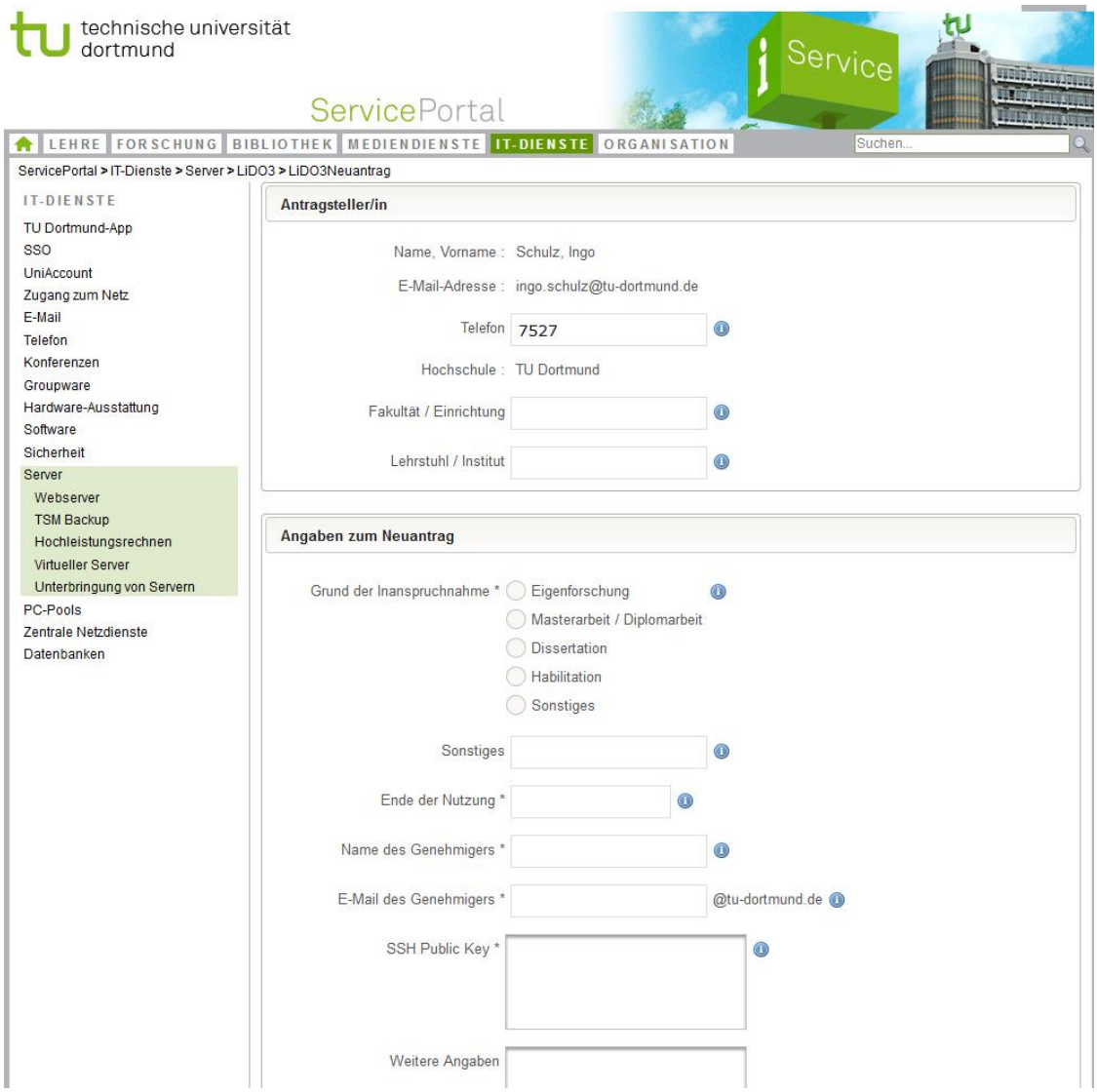


**Abbildung 1.3:** Formular für einen Neuantrag von LiDO3. [fIICdTD18]

### 1.3.2 Login

Der Zugriff auf LiDO3 erfolgt per SSH oder SCP über einen der beiden Gatewayserver:

- gw01.lido.tu-dortmund.de
- gw02.lido.tu-dortmund.de

Die Anmeldung erfolgt zertifikatsbasiert. Dazu benötigen Sie ihren privaten Key der zu Ihrem bereits hinterlegtem Public Key passt. Der Loginname ist Ihr UniAccount in der Form m... oder sm...

Unter Linux kann daher ein ssh Befehl (z.B. `ssh -i privkey smnutzer@gw...`) genutzt werden. Für das Betriebsystem Windows wird ein ssh-Client wie z.B. „Putty“ empfohlen.

Für die Einrichtung eines Profil in Putty gibt [fIICdTD18, Folie 10] eine kurze Anleitung.

Die Gatewayserver, sowie alle anderen Rechenknoten sind mit dem zentralen Dateisystem BeeGFS verbunden, so dass von jedem Knoten die gleiche Dateistruktur (`/work`) einsehbar ist. Erinnerung (siehe Abschnitt 1.2): die Homeordner sind auf den Gatewayservern schreib- und lesbar gemountet und auf Rechnerknoten ausschließlich **lesbar**!

## 1.4 Das Modulsystem

Auf LiDO3 steht eine umfangreiche Auswahl an Software zur Verfügung. Die Nutzung der Software wird mit Hilfe des Modulsystems ermöglicht. So besteht nicht nur die Möglichkeit eine bestimmte Software zu laden sondern es kann darüber hinaus auch die Version dieser Software ausgesucht werden. Dies hat den Vorteil, dass Ergebnisse reproduzierbar bleiben, selbst wenn eine neue Version installiert wird. Das Modulsystem erlaubt eine saubere Trennung der installierten Versionen einer Software. Diese Trennung wird dadurch erreicht, dass für eine bestimmte Softwareversion ein bestimmtes Modul geladen werden muss, bevor diese Software einsatzbereit wird.

Eine Übersicht mit allen installierten Modulen kann mit dem Kommando `module avail` abgefragt werden. Es erscheint eine Liste mit allen zur Verfügung stehenden Modulen. Die Liste lässt sich einschränken, indem nach `module avail` ein oder mehrere Anfangsbuchstaben der gesuchten Software als Argument übergeben werden.

→ Beispiel: `module avail mat`

Wurde eine passendes Modul gefunden, so kann es mit einem der beiden Kommandos:

→ `module add <modulename>` oder `module load <modulename>`

geladen werden. Das Kommando `module list` gibt eine Übersicht über alle geladenen Module.

Um zu sehen, welche Umgebungsvariablen durch ein Modul in welcher Weise verändert werden, kann das Kommando `module display <modulename>` verwendet werden.

Soll ein geladenes Modul wieder entladen werden, weil z.B. eine andere Version der Software genutzt werden soll (es ist nicht möglich Module einer Software mit unterschiedlichen Versionen gleichzeitig zu laden), so kann das Kommando `module rm <modulename>` genutzt werden.

Um alle geladenen Module gleichzeitig zu entladen, lautet die Anweisung `module purge`.

### 1.4.1 Kommandos im Überblick

Die Tabelle 1.3 fasst die wichtigsten Kommandos zur Verwaltung des Modulsystems von LiDO3 zusammen.

| | |
|---|---|
| `module avail` | Liste verfügbarer Module. |
| `module avail mat` | Liste verfügbarer Module mit dem Anfangsbuchstaben „mat“. |
| `module list` | Aktuell geladene Module anzeigen. |
| `module add <modulename>` | Lädt das Modul <modulename>. |
| `module load <modulename>` | Lädt das Modul <modulename>. |
| `module display <modulename>` | Zeigt an, welche Umgebungsvariablen durch ein Module verändert werden. |
| `module rm <modulename>` | Entlädt das Modul <modulename>. |
| `module purge` | Entlädt alle geladenen Module. |

**Tabelle 1.3:** Die wichtigsten Kommandos zur Verwaltung des Modulsystems von LiDO3.

**Hinweis 1:** Zu Beginn sind keine Module geladen. Es müssen also zunächst alle benötigten Module geladen werden. Um stets benötigte Module nicht ständig neu laden zu müssen, können diese in der `.bash_profile`-Datei im Homeordner festgelegt werden.

**Hinweis 2:** Es ist nicht möglich unterschiedliche Versionen einer Software gleichzeitig zu laden.

## 1.5 Scheduling mit Slurm

Neben den folgenden Informationen wird zusätzlich auf das Slurm-Cheatsheet verwiesen: `literature/slurm_summary.pdf`

### 1.5.1 Resourcenmanagement

Um die bereitgestellten Ressourcen von LiDO3 zu managen, wird das Schedulingsystem Slurm (Link zur Produktseite: [Slu18]) verwendet. Mit dieser Software können so genannte **Jobs** priorisiert werden.

#### 1.5.1.1 Partitionen

Die Grundlage des Schedulingsystems bilden Warteschlangen, die in Slurm die Bezeichnung **Partitionen** tragen.

Die Partitionen von LiDO3:

- **öffentliche Partitionen:** Sind von der maximalen Walltime (siehe Tabelle 1.4) abhängig, d.h. der maximal möglichen zur Verfügung stehenden Rechenzeit, wenn der Job startet.
- **private Partitionen:** Nur für bestimmte Forschergruppen.

Die öffentlichen Partitionen von LiDO3:

| **Partition** | **Max Walltime** (hhh:mm:ss) | **Bemerkung** |
|---|---|---|
| short | 02:00:00 | – |
| med | 08:00:00 | – |
| long | 48:00:00 | – |
| ultralong | 672:00:00 | Keine GPU und „non-blocking“ Knoten nutzbar. |

**Tabelle 1.4:** Die öffentlichen Partitionen von LiDO3.

#### 1.5.1.2 Knotentypen

Neben der Partition kann bei der Joberstellung der gewünschte Knotentyp (in Slurm mit **Feature** bezeichnet) mit angegeben werden. Die folgende Übersicht zeigt die wichtigsten Features:

| **Feature** | **CPU Typ** | **GPU Typ** | **Max Cores** | **Max RAM/Knoten** |
|---|---|---|---|---|
| cstd01 | E5-2640v4 | – | 20 | 64 GB |
| cstd02 oder ib_1to1 | E5-2640v4 | – | 20 | 64 GB |
| cquad01 | E5-4640v4 | – | 48 | 256 GB |
| cquad02 | E5-4640v4 | – | 48 | 1024 GB |
| tesla_k40 | E5-2640v4 | 2x Nvidia Tesla K40 | 20 | 64 GB |

**Tabelle 1.5:** Die Knotentypen von LiDO3. Alle CPU Typen sind Intel Xeon.

**Hinweis 1:** Wenn bei der Joberzeugung keine Angabe zur RAM-Größe gemacht wird, stehen standardmäßig 512 MB RAM pro Core zur Verfügung.

**Hinweis 2:** Wenn keine Spezialanforderung wie z.B. „tesla_k40“ benötigt wird, kann auf eine Featureangabe verzichtet werden.

**Hinweis 3:** Die Auswahl eines GPU-Knotens reicht allein nicht aus um mit GPUs zu arbeiten! Bei der Joberstellung muss zusätzlich der Parameter `--gres=gpu:x` angegeben werden, um pro Knoten `x` Grafikkarten (mit: $x \leq 2$) anzufordern.

### 1.5.2 Jobmanagement

Das Jobmanagement erfolgt auf einem der beiden Gatewayserver via Slurm. Dabei besitzt jeder Job (schon in der Wartephase) eine Job ID. Mit dem Befehl `squeue -u <username>` können auf dem Gatewayservern die ID sowie zusätzliche Informationen ermittelt werden. Um einen Job aus der Warteschlange zu löschen bzw. vorzeitig zu Beenden, kann `scancel <ID>` durchgeführt werden.

Außerdem gibt es eine graphische Übersicht über alle Jobs. Diese kann mit dem Befehl `sview` gestartet werden. Dazu ist es erforderlich, dass die ssh Sitzung mit X-Umleitung gestartet wurde z.B. durch `ssh -X`. Weiterhin muss auf dem eigenen Rechner ein X-Server aktiv sein.

#### 1.5.2.1 Interaktiv

Die interaktive Nutzung eines LiDO3-Knoten erfolgt in zwei Schritten:

1. Knoten für Job mit Hilfe von `salloc` (Syntax siehe unten) reservieren.
2. Konsole auf einem der reservierten Knoten starten: `srun --pty bash`

Die allgemeine Syntax von `salloc`:

```
salloc -N x -C F -c c --mem=y -t w -p part
```

Mit diesem Kommando werden `x` Knoten mit Feature `F` angefordert. Es werden höchstens `c` Cores, `y` MB pro Knoten an RAM und `w=hh:mm:ss` an Walltime benötigt. Die Anforderung wird der Partition `part` zugeordnet.

Ein konkretes `salloc`-Beispiel:

```
salloc -N 1 -C cquad01 -c 4 --mem=250G -t 01:00:00 -p short
```

In diesem Beispiel wird ein Knoten `-N 1` mit dem Feature cquad01 `-C cquad01` angefordert. Weiterhin werden pro Knoten 4 Cores `-c 4` und 250 GB RAM `--mem=250G` gefordert. Die Walltime beträgt eine Stunde `-t 01:00:00` (alternativ funktioniert auch die Angabe in Minuten, in diesem Beispiel `-t 60`) und der Job wird in der Partition short `-p short` priorisiert.

Weitere Parameter werden im Abschnitt 1.5.2.3 aufgelistet. Außerdem kann mit dem Kommando `man salloc` eine detaillierte Aufstellung angezeigt werden.

**„Hello World“-Programm als interaktiver Job**

Im Folgenden wird ein einfaches „Hello World“-Programm, dass in C geschrieben wurde, interaktiv auf einem der Rechnerknoten von LiDO3 ausgeführt. Dazu sind folgende Schritte notwendig:

0. „helloWorld.c“ befindet sich initial im Ordner: `/home/<username>`
1. Einen beliebigen Rechenknoten für 1 Stunde in der Partition `short` reservieren.

   → `salloc -N 1 -c 1 -t 01:00:00 -p short`
2. Konsole erhalten.

   → `srun --pty bash`
3. „helloWorld.c“ von `/home` nach `/scratch` kopieren.

   → `mkdir /scratch/<username>`

   → `cp /home/<username>/helloWorld.c /scratch/<username>/`
4. Das gcc-Modul laden. Im Beispiel wird die gcc-Version 8.2.0 geladen.

   → `module load gcc/8.2.0`
5. „helloWorld.c“ kompilieren.

   → `cd /scratch/<username>`

   → `gcc helloWorld.c -o helloWorld`
6. „./helloWorld“ ausführen und Ergebnis in die Datei „result.txt“ schreiben.

   → `./helloWorld > result.txt`
7. Das Ergebnis „result.txt“ von `/scratch` nach `/work` kopieren

   → `cp /scratch/<username>/result.txt /work/<username>/`
8. Rechnerknoten verlassen.

   → `exit`
9. Zur Sicherung des Ergebnis: „result.txt“ von `/work` nach `/home` kopieren.

   → `cp /work/<username>/result.txt /home/<username>/`

#### 1.5.2.2 Per Slurm Script

Slurm Scripts ermöglichen das bequeme automatische Starten eines Jobs. Ein Slurm Script

```
#!/bin/bash -l
# The following line asks for usage of 1 compute node with 2 GPUs
# for 10 minutes using, 20 cores per node for a non-threaded code.
#SBATCH --time=10
#SBATCH --nodes=1 --cpus-per-task=20 --constraint=cgpu01
   --gres=gpu:2
# or: #SBATCH -N 1 -c 1 -C cgpu01 --gres=gpu:2
#SBATCH --partition=long
# Possible 'constraint' values (Features):
#  all, public
#
#  cgpu01 OR gpu OR tesla_k40
#
#  cquad01 OR xeon_e54640v4
#  cquad02 OR xeon_e54640v4
#
#  cstd01 OR xeon_e52640v4 OR ib_1to3
#  cstd02 OR xeon_e52640v4 OR ib_1to1 OR nonblocking_comm
#
# Maximum 'mem' values depending on constraint (values in MB):
# cstd01/xeon_e52640v4/ib_1to3/cgpu01 AND
# cstd02/xeon_e52640v4/ib_1to1/nonblocking_comm: 62264
# cquad01: 255800
# cquad02: 1029944
#SBATCH --mem=60000
#SBATCH --mem_bind=verbose,local --hint=memory_bound

#SBATCH --mail-user=test.user@tu-dortmund.de
# Possible 'mail-type' values: NONE, BEGIN, END, FAIL, ALL
   (=BEGIN,END,FAIL)
#SBATCH --mail-type=ALL
#SBATCH --signal=15@30

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID
   $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
srun ./myapp
```

**Code 1.1:** Ein Slurm Script Beispiel.

startet im Normalfall mit der folgenden Zeile:

```
#!/bin/bash -l
```

Anschließend folgen verschiedene Zeilen, die die Eigenschaften des Jobs festlegen, u.a. Walltime, Partition, Speicher, Prozesse pro Knoten, Knoteneigenschaften, E-Mail Benachrichtigung, spezielle Ausgabedatei und Programmaufruf. Diese Zeilen beginnen immer mit `#SBATCH` gefolgt von Parametern. Beginnt eine Zeile mit `#` aber nicht mit `#SBATCH`, so ist diese Zeile als Kommentarzeile aufzufassen. Beispielkommentare:

```
#Dies ist ein Kommentar
##SBATCH ...
```

Der Code 1.1 zeigt ein Slurm Script Beispiel. Die einzelnen Parameter werden im Abschnitt 1.5.2.3 aufgelistet. Außerdem kann mit dem Kommando `man sbatch` eine detaillierte Aufstellung angezeigt werden.

### „Hello World“-Programm per Slurm Script

Im Folgenden wird ein einfaches „Hello World“-Programm, dass in C geschrieben wurde, per Slurm Script auf einem der Rechnerknoten von LiDO3 ausgeführt. Das Programm „helloWorld.c“ befindet sich initial im LiDO3-Ordner: `/home/<username>`.

Zur Ausführung von „helloWorld.c“ kann beispielsweise das Slurm Script in Code 1.3 verwendet werden. Dazu einfach das Kommando `sbatch <Scriptname>` nutzen.

Während der Ausführung des Slurm Scripts in Code 1.3 wird die Datei `slurm-<JobId>.out` generiert. Der Inhalt der Datei ist in Code 1.2 zu sehen.

```
1 Do 23. Aug 14:30:56 CEST 2018
2 sbatch: START SLURM_JOB_ID 1469084 (SLURM_TASK_PID 172179) on
     cstd01-196
3 sbatch: SLURM_JOB_NODELIST cstd01-196
4 sbatch: SLURM_JOB_ACCOUNT fak04
```

**Code 1.2:** Der Inhalt der generierten Datei „slurm-<JobId>.out“.

Um von der Sicherung des Homeverzeichnis zu profitieren, sollte nach der Ausführung des Slurm Script (Code 1.3) das Ergebnis „result.txt“ von `/work` nach `/home` kopiert werden.

```
#!/bin/bash -l

# Einen beliebigen Rechenknoten fuer 1 Stunde in der Partition short reservieren.
#SBATCH -N 1 -c 1 -t 01:00:00 -p short

# E-Mail Benachrichtigung einrichten.
#SBATCH --mail-user=test.user@tu-dortmund.de
#SBATCH --mail-type=ALL
#SBATCH --signal=15@30

# Zusaetzliche Informationen ausgeben.
date
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID) on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"

# helloWorld.c von /home nach /scratch kopieren
mkdir -p /scratch/<username>
cp /home/<username>/helloWorld.c /scratch/<username>/

# Das gcc-Modul laden. Im Beispiel wird die gcc-Version 8.2.0 geladen.
module purge
module load gcc/8.2.0

# helloWorld.c kompilieren.
cd /scratch/<username>
gcc helloWorld.c -o helloWorld

# helloWorld ausfuehren und Ergebnis in die Datei result.txt schreiben.
srun -c 1 ./helloWorld > result.txt # -c ist optional

# Das Ergebnis result.txt von /scratch nach /work kopieren.
cp /scratch/<username>/result.txt /work/<username>/

# Erstellten Ordner loeschen.
rm -r /scratch/<username>
```

**Code 1.3:** Slurm Script zur Ausführung von „helloWorld.c“.

#### 1.5.2.3 salloc und #SBATCH Parameter

Die folgende Liste zeigt einige wichtige Parameter, die im Befehl `salloc` und `#SBATCH` genutzt werden können. Weitere Parameter können mit dem jeweiligen `man`-Befehl (`man salloc` bzw. `man sbatch`) abgefragt werden. Außerdem gibt es eine Übersicht mit allen Parameter für `salloc` in [Sal18] bzw. für `#SBATCH` in [Sba18].

- `-N x-y` oder `--nodes=x-y`

→ Anzahl der Knoten wird auf mindestens `x` und höchstens `y` festgelegt. Die Angabe von `-y` ist optional.

- `--mem=x`

→ Pro Knoten werden `x` MB RAM reserviert. Folgt dem `x` ein T oder G, werden nicht `x` MB sondern `x` Terabyte bzw. `x` Gigabyte RAM reserviert.

- `-p <partition_name>` oder `--partition=<partition_name>`

→ Angabe der Partition.

- `-t <walltime>` oder `--time=<walltime>`

→ Angabe der Walltime in Minuten. Zu den zulässigen Zeitformaten gehören “Minuten“, “Minuten:Sekunden“, “Stunden:Minuten:Sekunden“, “Tage-Stunden“, “Tage-Stunden:Minuten“ und “Tage-Stunden:Minuten:Sekunden“.

- `-J <jobname>` oder `--job-name=<jobname>`

→ Festlegung des Jobnamens.

- `-o mypath/filename` oder `--output=mypath/filename` (nur in `#SBATCH` möglich)

→ Pfad und Dateiname für die Ausgabe (für mögliche Formatoptionen mit Slurmvariablen bitte manpage von `sbatch` aufrufen)

- `-C <constraint>` oder `--constraint=<constraint>`

→ Nur Knoten mit dem Feature `constraint` werden angefordert.

- `--gres=gpu:x`

→ Pro Knoten werden `x` Grafikkarten angefordert. (Hinweis: `-C gpu01` oder `-C tesla_k40` reicht allein nicht aus, um mit GPUs zu arbeiten!)

- `-c x` oder `--cpus-per-task=x`

→ `x` Cores pro Task werden angefordert.

- `-n y` oder --ntasks=y

→ `y` Tasks werden maximal pro Jobstep ausgeführt.

- `--ntasks-per-node=z`

→ `z` Tasks pro Knoten sollen ausgeführt werden. Bei Angabe von `-n y` (siehe Aufzählungspunkt davor) wird `z` nur als Höchstzahl angesehen.

- `--export=var1,...,varn`

→ Umgebungsvariablen `var1` bis `varn` stehen dem Job zur Verfügung, Alternativ: bei `=ALL` werden alle und bei `=NONE` keine Umgebungsvariable übernommen.

- `--signal=<sig_num>[@<sig_time>]`

→ Wenn ein Job sich `sig_time` Sekunden von seiner Endzeit entfernt befindet, wird das Signal `sig_num` gesendet. `sig_time` muss ein ganzzahliger Wert zwischen 0 und 65535 sein. Wenn ein `sig_num` ohne `sig_time` angegeben wird, beträgt die Standardzeit 60 Sekunden.

- `--mem-bind=<type>`

→ Bindet Tasks an den Speicher. Unterstützte Optionen sind u.a.:

  - local: lokalen Speicher des verwendeten Prozessors verwenden
  - no[ne]: Aufgaben nicht an den Speicher binden (default)
  - q[uiet] leise binden, bevor der Task läuft (default)
  - v[erbose] ausführliche Berichterstellung vor der Ausführung der Aufgabe

- `--hint=<type>`

→ Binden von Aufgaben gemäß Anwendungshinweisen. Unterstützte Optionen sind:

  - compute_bound: Die Einstellungen für die Berechnung gebundener Anwendungen wählen: alle Kerne in jedem Sockel verwenden, ein Thread pro Kern.
  - memory_bound: Die Einstellungen für speichergebundene Anwendungen wählen: Verwenden Sie nur einen Kern in jedem Sockel, einen Thread pro Kern.
  - (no)multithread: (keine) Verwendung zusätzlicher Threads mit In-Core-Multi-Threading, was kommunikations-intensiven Anwendungen zugute kommen kann. Wird nur mit dem Task/Affinity-Plugin unterstützt.
  - help: Hilfemeldung anzeigen.

Für die Einrichtung einer E-Mail Benachrichtigung können in `salloc` und `#SBATCH` die folgenden Parameter genutzt werden:

- `--mail-user=user.name@tu-dortmund.de`

→ Legt die Email Benachrichtigungsadresse fest.

- `--mail-type=BEGIN`

→ Mailbenachrichtigung bei Jobstart

- `--mail-type=END`

→ Mailbenachrichtigung bei Jobende.

- `--mail-type=FAIL`

→ Mailbenachrichtigung bei Jobabbruch.

- `--mail-type=ALL`

→ Mailbenachrichtigung in allen Fällen.

# Kapitel 2

# Ein Vergleich mit den Workstation ls4gpu1 und ls4gpu2

## 2.1 Approximation der Kreiszahl Pi

Die Tabellen 2.1 und 2.2 zeigen die gemessenen Ausführungszeiten von unterschiedlichen Implementierungen der Approximation der Kreiszahl Pi[1] unter Verwendung eines *tesla_k40*-Knotens von LiDO3 und der Workstation ls4gpu2. Wie bereits zu vermuten war, ist die Intel® Xeon® E5-2699 v4 CPU der Workstation ls4gpu2 leistungsstärker als die Intel® Xeon® E5-2640 v4 CPU eines *tesla_k40*-Knotens (siehe CPU-Spezifikationen in Tabelle A.1). Demzufolge ist die sequentielle und die OpenMP-Implementierung auf der Workstation ls4gpu2 schneller. Außerdem ist bei beiden Systemen und Compilern erkennbar, dass die OpenACC-Variante für CPUs zu keiner Performanceverbesserung gegenüber der sequentielle Implementierung führt und deutlich langsamer als die OpenMP-Implementierung ist. Die Ausführungszeiten der OpenACC-Implementierung für GPUs zeigen, dass der PGI-Compiler und die bessere Performance der NVIDIA® Tesla® K40m bei doppelter Genauigkeit (siehe Abschnitt 2.2.3) zu einer kürzeren Ausführungszeit auf einem *tesla_k40*-Knoten führen. Die NVIDIA® Quadro® P6000 der Workstation ls4gpu2 ist hier fast doppelt so langsam.

[1] Weitere Details zum Algorithmus und den einzelnen Implementierungen können in der Dokumentation „GPGPU-Programmierung mit OpenACC“ nachgelesen werden. Außerdem befindet sich der verwendete Code im Ordern: tests/approximationOfPi/LiDO3/ bzw. tests/approximationOfPi/ls4gpu2/.

| N | Sequentiell<br>CPU: Intel® Xeon®<br>E5-2640 v4<br>Threads: 1<br>GCC-Compiler 8.2.0 | OpenMP<br>CPU: Intel® Xeon®<br>E5-2640 v4<br>Threads: 20<br>GCC-Compiler 8.2.0 | OpenACC<br>CPU: Intel® Xeon®<br>E5-2640 v4<br>ng=2048,nw=32,vl=32<br>PGI-Compiler 18.5 | OpenACC<br>GPU: NVIDIA®<br>Tesla® K40m<br>ng=2048,nw=32,vl=32<br>PGI-Compiler 18.5 |
|---|---|---|---|---|
| $10^3$ | 0,0035 | 0,0877 | 0,0047 | 0,3497 |
| $10^4$ | 0,0352 | 0,0905 | 0,0475 | 0,3498 |
| $10^5$ | 0,3571 | 0,1524 | 0,4731 | 0,3525 |
| $10^6$ | 3,5776 | 0,6005 | 4,7261 | 0,3903 |
| $10^7$ | 35,6675 | 5,3641 | 47,2502 | 0,8264 |
| $10^8$ | 356,7856 | 40,1020 | 472,4540 | 5,1852 |
| $10^9$ | 3.566,7187 | 386,5697 | 4.724,3615 | 48,5446 |
| $10^{10}$ | 35.645,6206 | 3.637,0772 | 47.246,6524 | 482,3448 |

**Tabelle 2.1:** Die Ausführungszeiten von unterschiedlichen Implementierungen zur Approximation der Kreiszahl Pi unter Verwendung eines ***tesla_k40*-Knotens von LiDO3**. Die Tabelle zeigt die Messwerte für $N$ zwischen $10^3$ und $10^{10}$ in Millisekunden (ms). Die Abkürzungen $ng$, $nw$ und $vl$ stehen für die Anzahl an num_gangs (ng), num_worker und vector_length (vl). Die einzelnen Implementierungen befinden sich im Ordner: tests/approximationOfPi/LiDO3/

| N | Sequentiell<br>CPU: Intel® Xeon®<br>E5-2699 v4<br>Threads: 1<br>GCC-Compiler 7.1.0 | OpenMP<br>CPU: Intel® Xeon®<br>E5-2699 v4<br>Threads: 20<br>GCC-Compiler 7.1.0 | OpenACC<br>CPU: Intel® Xeon®<br>E5-2699 v4<br>ng=2048,nw=32,vl=32<br>GCC-Compiler 7.1.0 | OpenACC<br>GPU: NVIDIA®<br>Quadro® P6000<br>ng=2048,nw=32,vl=32<br>GCC-Compiler 7.1.0 |
|---|---|---|---|---|
| $10^3$ | 0,0077 | 0,3885 | 0,0056 | 2,5667 |
| $10^4$ | 0,0750 | 0,8725 | 0,0418 | 2,5877 |
| $10^5$ | 0,3332 | 0,4500 | 0,3279 | 2,5242 |
| $10^6$ | 5,7357 | 1,8829 | 2,8381 | 2,6817 |
| $10^7$ | 40,5872 | 3,5937 | 25,5672 | 2,5462 |
| $10^8$ | 276,2175 | 27,0945 | 254,3576 | 9,8536 |
| $10^9$ | 2.603,5820 | 174,4920 | 2.544,1908 | 82,1389 |
| $10^{10}$ | 25.659,3237 | 1.704,1143 | 25.429,9191 | 807,2331 |

**Tabelle 2.2:** Die Ausführungszeiten von unterschiedlichen Implementierungen zur Approximation der Kreiszahl Pi unter Verwendung der **Workstation ls4gpu2**. Die Tabelle zeigt die Messwerte für $N$ zwischen $10^3$ und $10^{10}$ in Millisekunden (ms). Die Abkürzungen $ng$, $nw$ und $vl$ stehen für die Anzahl an num_gangs (ng), num_worker und vector_length (vl). Die einzelnen Implementierungen befinden sich im Ordner: tests/approximationOfPi/ls4gpu2/

## 2.2 GPU-Performance bestimmen

GPUs können verwendet werden, um bestimmte Arten von Berechnungen zu beschleunigen. Da jedoch die GPU-Leistung stark zwischen den verschiedenen GPU-Geräten variiert, werden im folgenden drei Tests[2] vorgestellt. Sie können genutzt werden, um eine GPU in puncto Leistung zu quantifizieren.

Alle Tests können in der Dokumentation „GPU-Nutzung in MATLAB®“ nachgelesen werden. Außerdem befinden sich alle MATLAB®-Skripte im Ordner: tests/Matlab/code

Verwendete MATLAB®-Versionen:

- *tesla_k40*-Knoten von LiDO3: MATLAB® R2018a (Version 9.4)
- Workstation ls4gpu2: MATLAB® R2017b (Version 9.3)

### 2.2.1 GPU-Bandbreite testen

Das Resultat für einen *tesla_k40*-Knoten von LiDO3:

```
Using a Tesla K40m GPU.
Achieved peak send speed of 10.8591 GB/s
Achieved peak gather speed of 4.64616 GB/s
```

Das Resultat für die ls4gpu2-Workstation:

```
Using a Quadro P6000 GPU.
Achieved peak send speed of 10.674 GB/s
Achieved peak gather speed of 3.86652 GB/s
```

Die dazugehörigen Diagramme sind in Abbildung 2.1 und 2.2 zu sehen.

### 2.2.2 Testen von speicherintensiven Operationen

Das Resultat für einen *tesla_k40*-Knoten von LiDO3:

```
Using a Tesla K40m GPU.
Achieved peak read+write speed on the GPU:  211.784 GB/s
Achieved peak read+write speed on the host: 147.669 GB/s
```

Das Resultat für die ls4gpu2-Workstation:

```
Using a Quadro P6000 GPU.
Achieved peak read+write speed on the GPU:  371.035 GB/s
Achieved peak read+write speed on the host: 196.625 GB/s
```

Die dazugehörigen Diagramme sind in Abbildung 2.3 und 2.4 zu sehen.

[2] Quelle: https://de.mathworks.com/help/distcomp/examples/measuring-gpu-performance.html

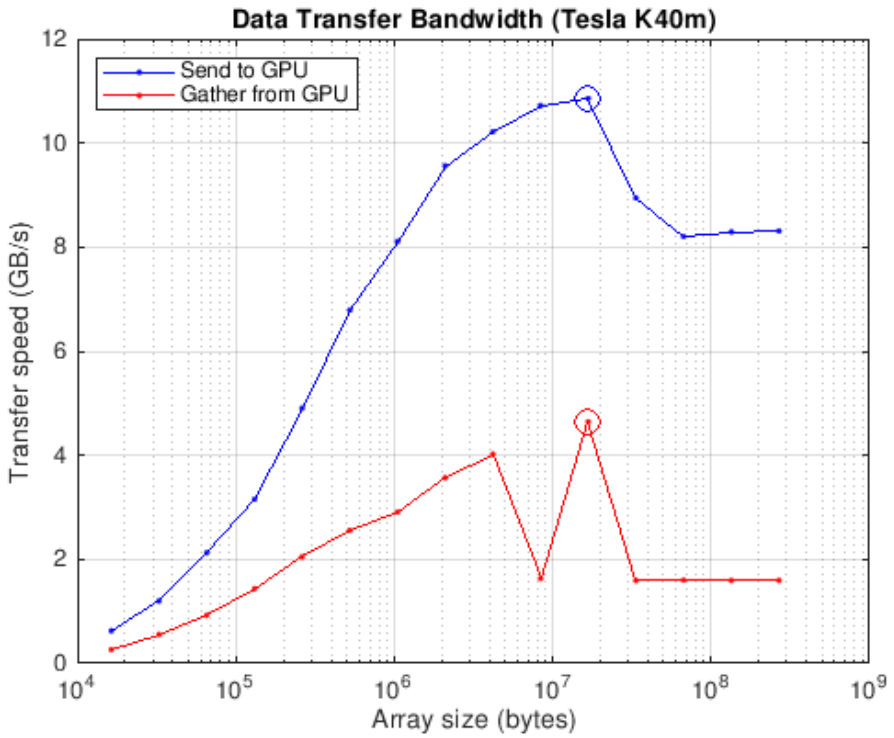


**Abbildung 2.1:** Send- und Gather-Performance der GPU eines *tesla_k40*-Knotens von LiDO3.

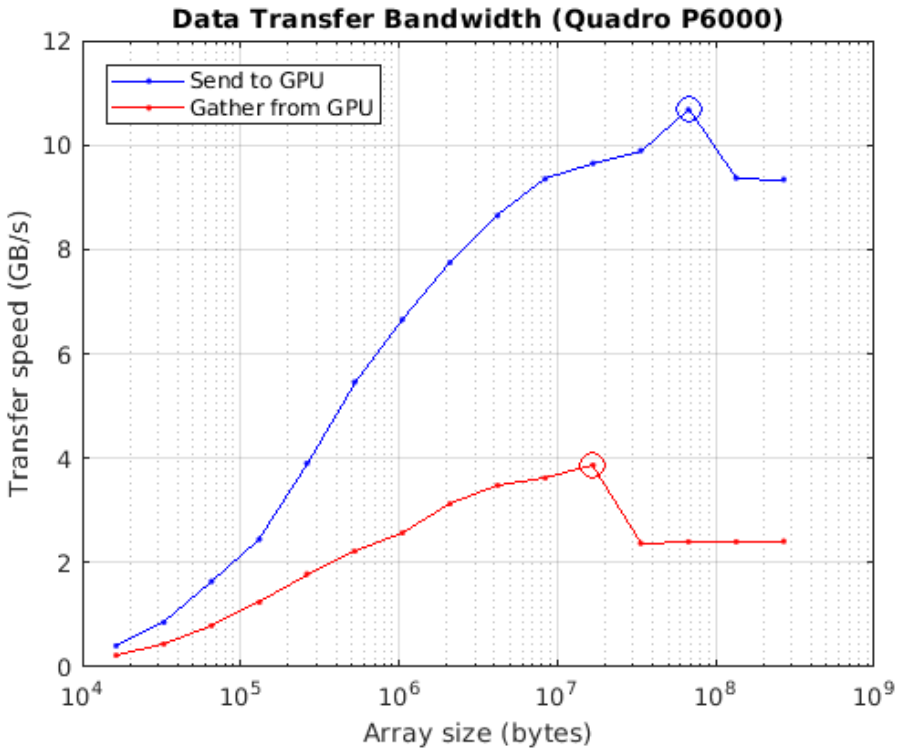


**Abbildung 2.2:** Send- und Gather-Performance der GPU von der ls4gpu2-Workstation.

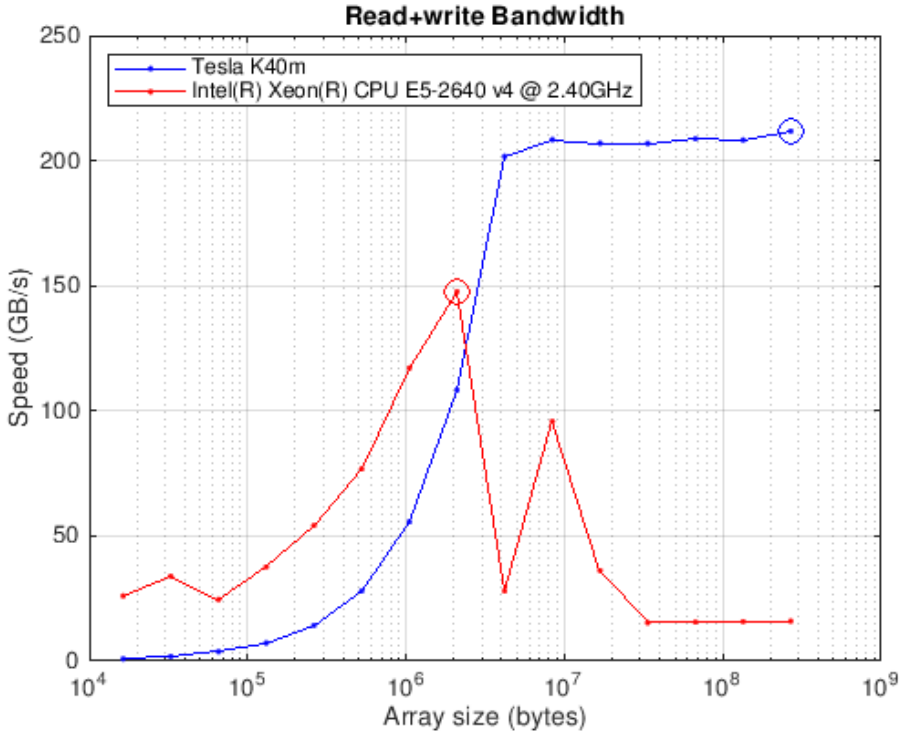


**Abbildung 2.3:** Schreib-/Lesegeschwindigkeit eines *tesla_k40*-Knotens von LiDO3.

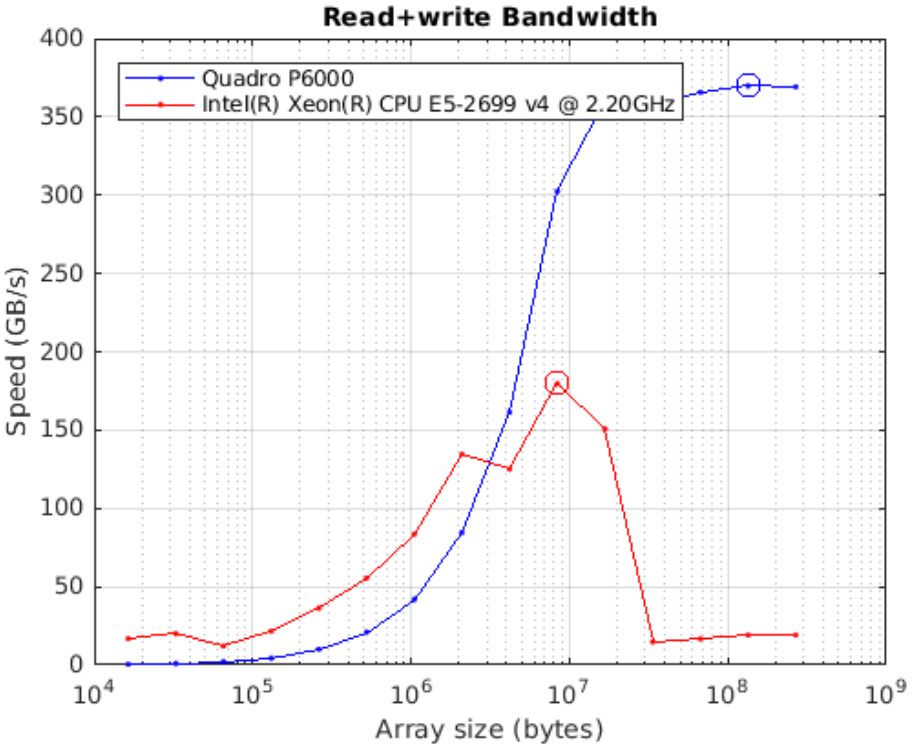


**Abbildung 2.4:** Schreib-/Lesegeschwindigkeit der ls4gpu2-Workstation.

### 2.2.3 Rechenleistung bestimmen (FLOPS)

#### 2.2.3.1 Einfache Genauigkeit (Single)

Das Resultat für einen *tesla_k40*-Knoten von LiDO3:

```
Using a Tesla K40m GPU.
Achieved peak calculation rates for single precision of:
      741.9 GFLOPS (Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz)
     3371.6 GFLOPS (Tesla K40m)
```

Das Resultat für die ls4gpu2-Workstation:

```
Using a Quadro P6000 GPU.
Achieved peak calculation rates for single precision of:
      1509.9 GFLOPS (Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz)
     10029.3 GFLOPS (Quadro P6000)
```

Die dazugehörigen Diagramme sind in Abbildung 2.5 und 2.6 zu sehen.

#### 2.2.3.2 Doppelte Genauigkeit (Double)

Das Resultat für einen *tesla_k40*-Knoten von LiDO3:

```
Using a Tesla K40m GPU.
Achieved peak calculation rates for double precision of:
      368.8 GFLOPS (Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz)
     1329.8 GFLOPS (Tesla K40m)
```

Das Resultat für die ls4gpu2-Workstation:

```
Using a Quadro P6000 GPU.
Achieved peak calculation rates for double precision of:
     703.7 GFLOPS (Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz)
     375.4 GFLOPS (Quadro P6000)
```

Die dazugehörigen Diagramme sind in Abbildung 2.7 und 2.8 zu sehen.

### 2.2.4 Zusammenfassung

Besonders die gemessenen GFLOPS-Werte zeigen einen großen Unterschied zwischen einfacher und doppelter Genauigkeit. Während die NVIDIA® Quadro® P6000 der ls4gpu2-Workstation bei einfacher Genauigkeit knapp dreimal so schnell ist wie die NVIDIA® Tesla® K40m von LiDO3, erreicht die Tesla® K40m bei doppelter Genauigkeit ein sehr gutes Ergebnis. Dort dreht sich das Ganze um und die Tesla® K40m ist ca. 3,5x so schnell wie die Quadro® P6000.

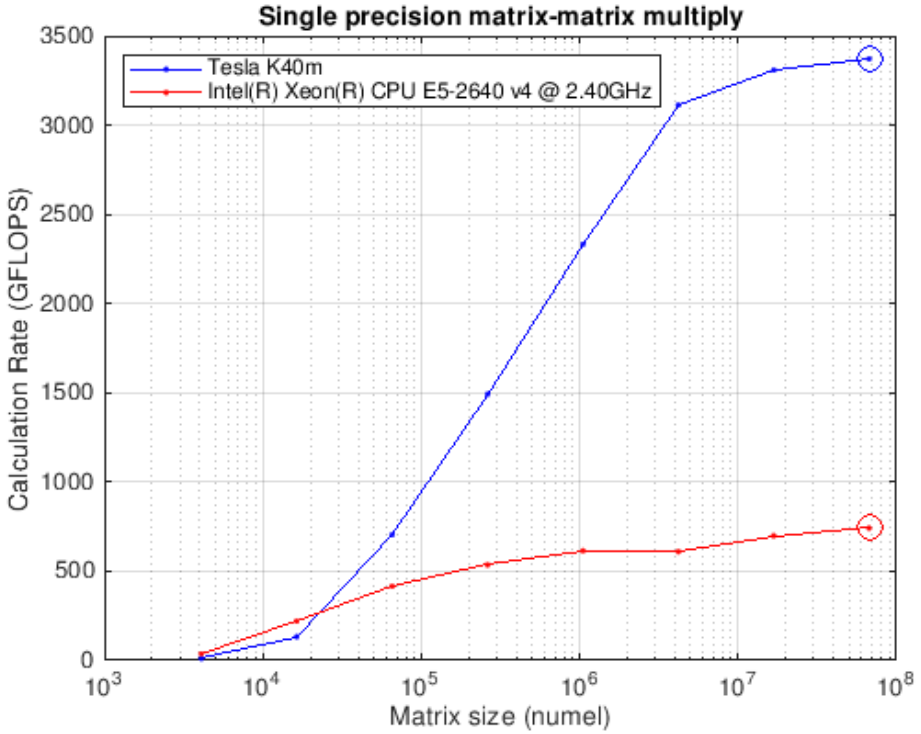


**Abbildung 2.5:** GFLOPS eines *tesla_k40*-Knotens von LiDO3 für einfache Genauigkeit.

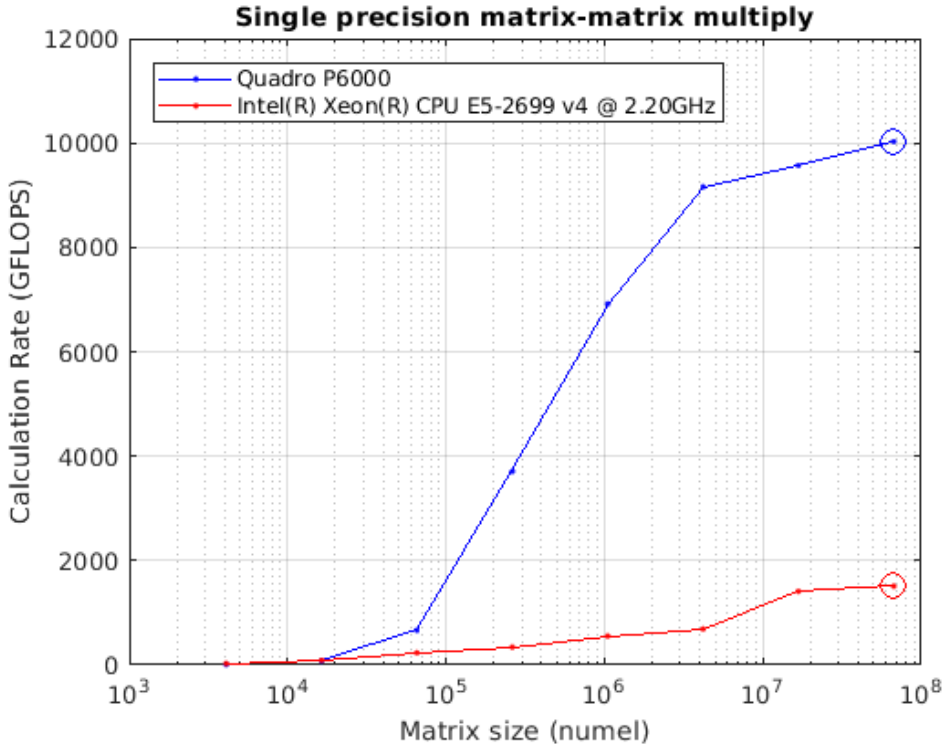


**Abbildung 2.6:** GFLOPS der ls4gpu2-Workstation für einfache Genauigkeit.

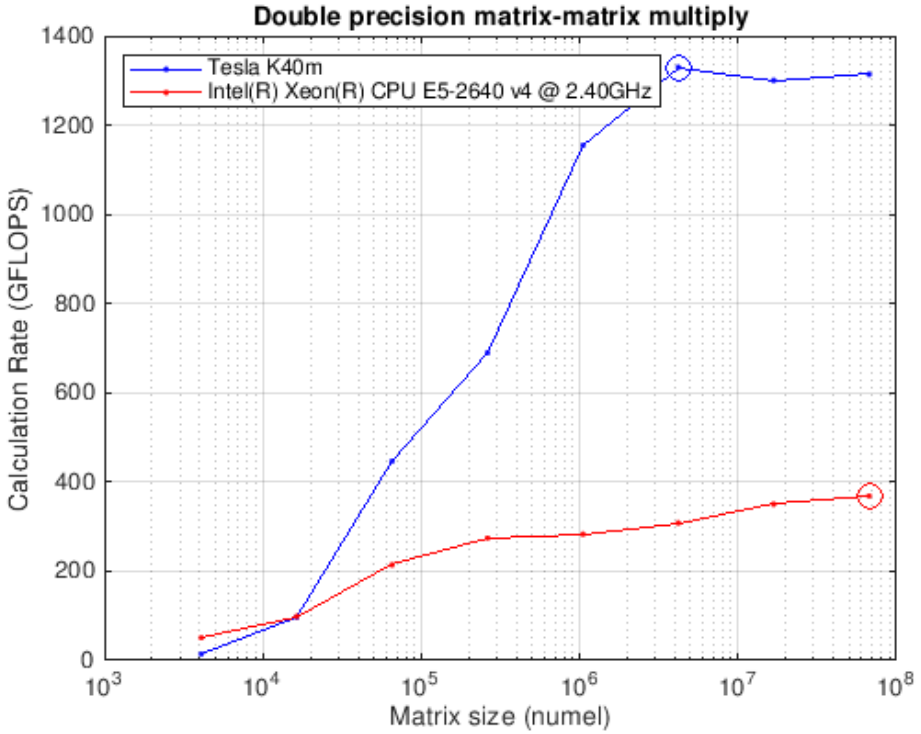


**Abbildung 2.7:** GFLOPS eines *tesla_k40*-Knotens von LiDO3 für doppelte Genauigkeit.

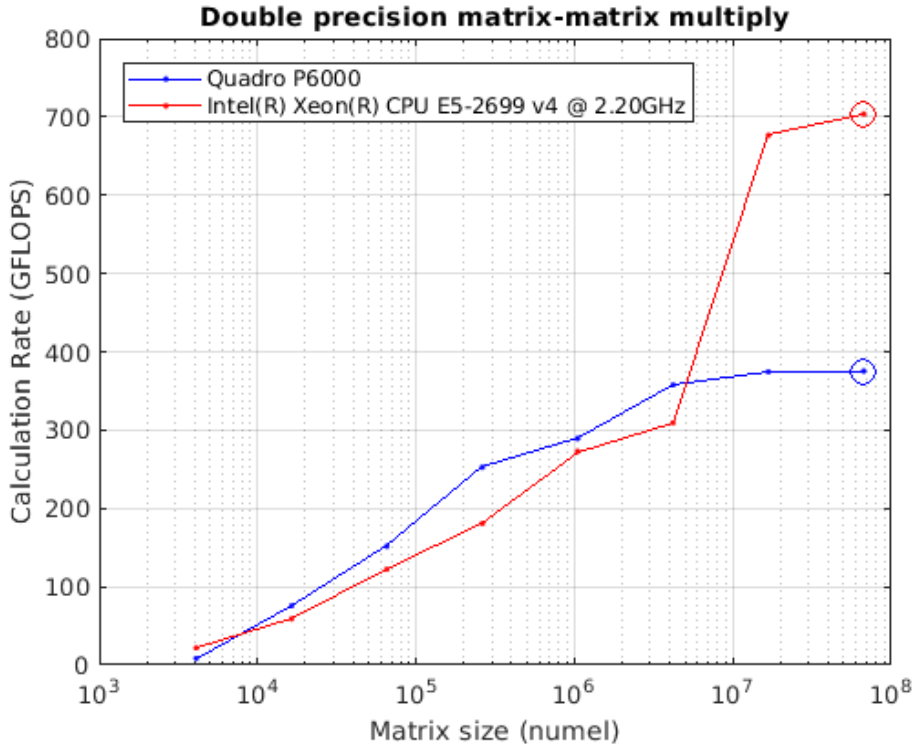


**Abbildung 2.8:** GFLOPS der ls4gpu2-Workstation für doppelte Genauigkeit.

## 2.3 Parallele Sortierung von Daten

Um zu überprüfen, ob die gemessenen Ausführungszeiten in Abschnitt 2.1 auf den Einsatz von unterschiedlichen Compilern (PGI und GCC) zurückzuführen sind, wurde zusätzlich ein mit CUDA [CUD18b] und der Thrust-Bibliothek [Thr18] implementierter paralleler Mergesort-Algorithmus ausgeführt und die Ausführungszeit gemessen.

Die Basis bildet der in [Sor18] vorgestellte Code, der sowohl eine parallele Version des Mergesort-Algorithmus (Idee des Algorithmus: siehe Abbildung 2.9) für CPUs als auch GPUs beinhaltet. Die parallele CPU-Version, die mit C++ implementiert wurde, verwendet die Sortierfunktion `std::sort` und die Mergefunktion `std::merge` aus der STL-Bibliothek. Die GPU-Version wurde mit CUDA und der Thrust-Bibliothek implementiert. Während die Thrust-Bibliothek für das erstellen des Arrays auf der GPU (`thrust::device_vector<>`) und das Sortieren (`thrust::stable_sort(...)`) genutzt wird, kann mit CUDA die Ausführungszeit des GPU-Programms gemessen werden.

Der Quellcode, die verwendeten Skripte und die Messergebnisse befinden sich im Ordner: tests/CUDA/sortDataParallel/

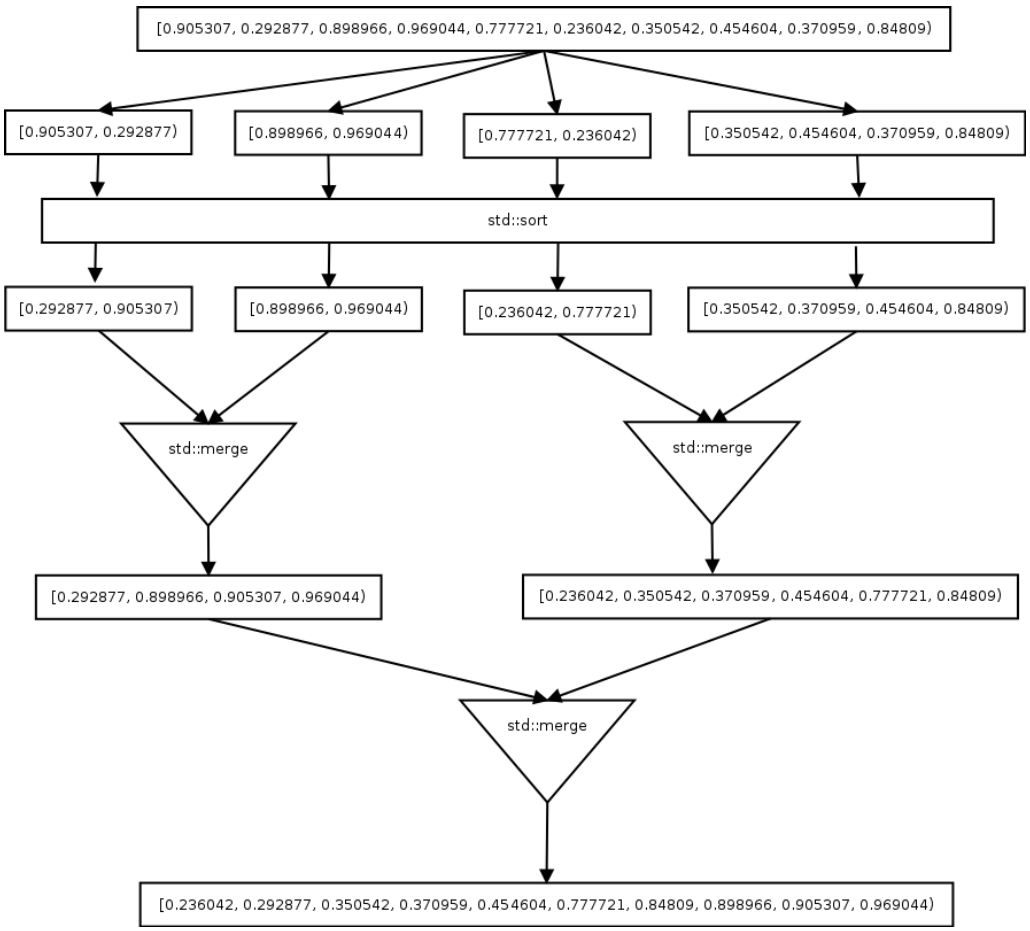


**Abbildung 2.9:** Der Ablauf des parallelen Merge-Sort-Algorithmus für CPUs. [Sor18]

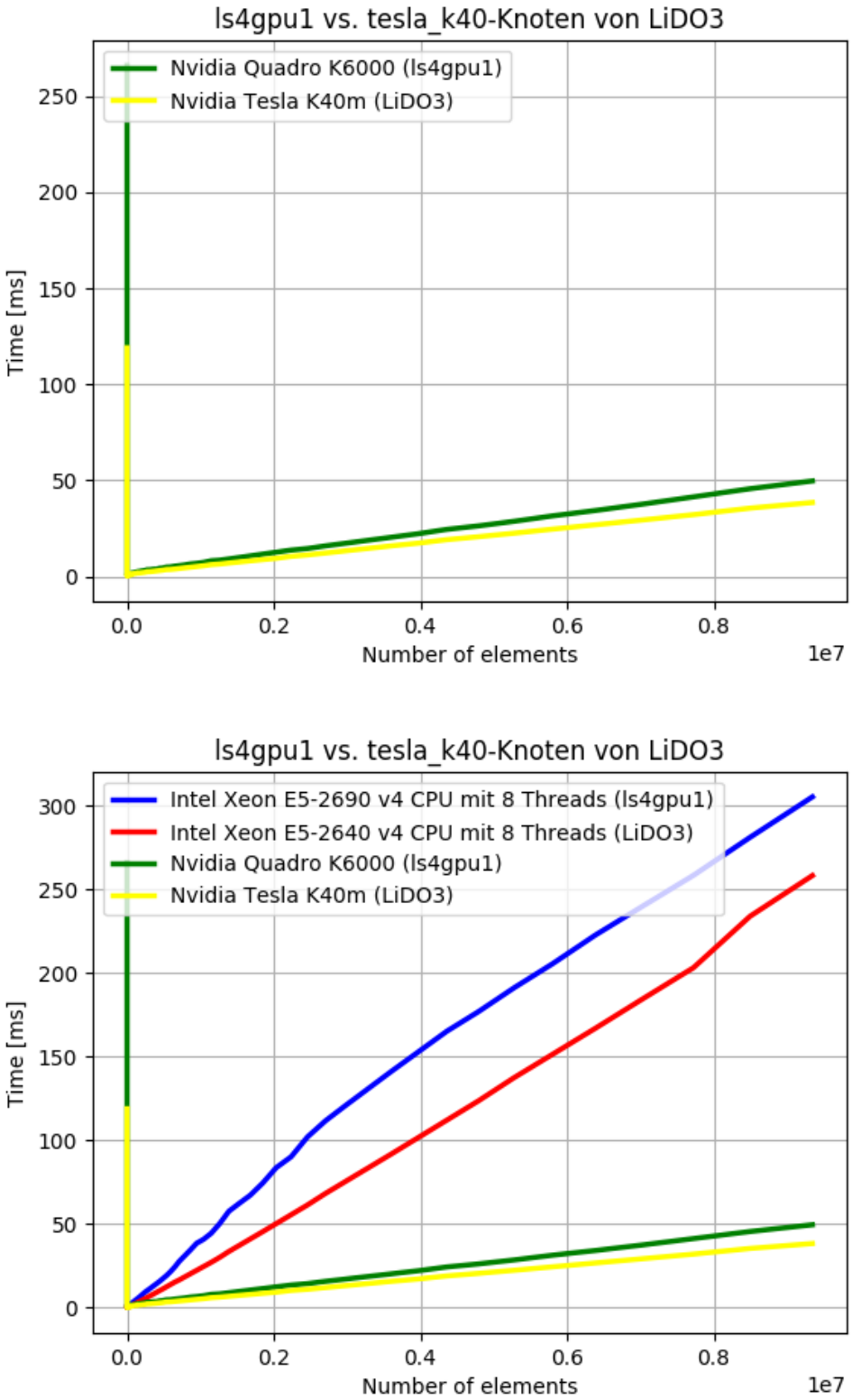


**Abbildung 2.10:** Das Ergebnis der Workstation ls4gpu1 und eines *tesla_k40*-Knotens von LiDO3 für 0 bis $9,4 * 10^6$ Elemente vom Typ *double*.

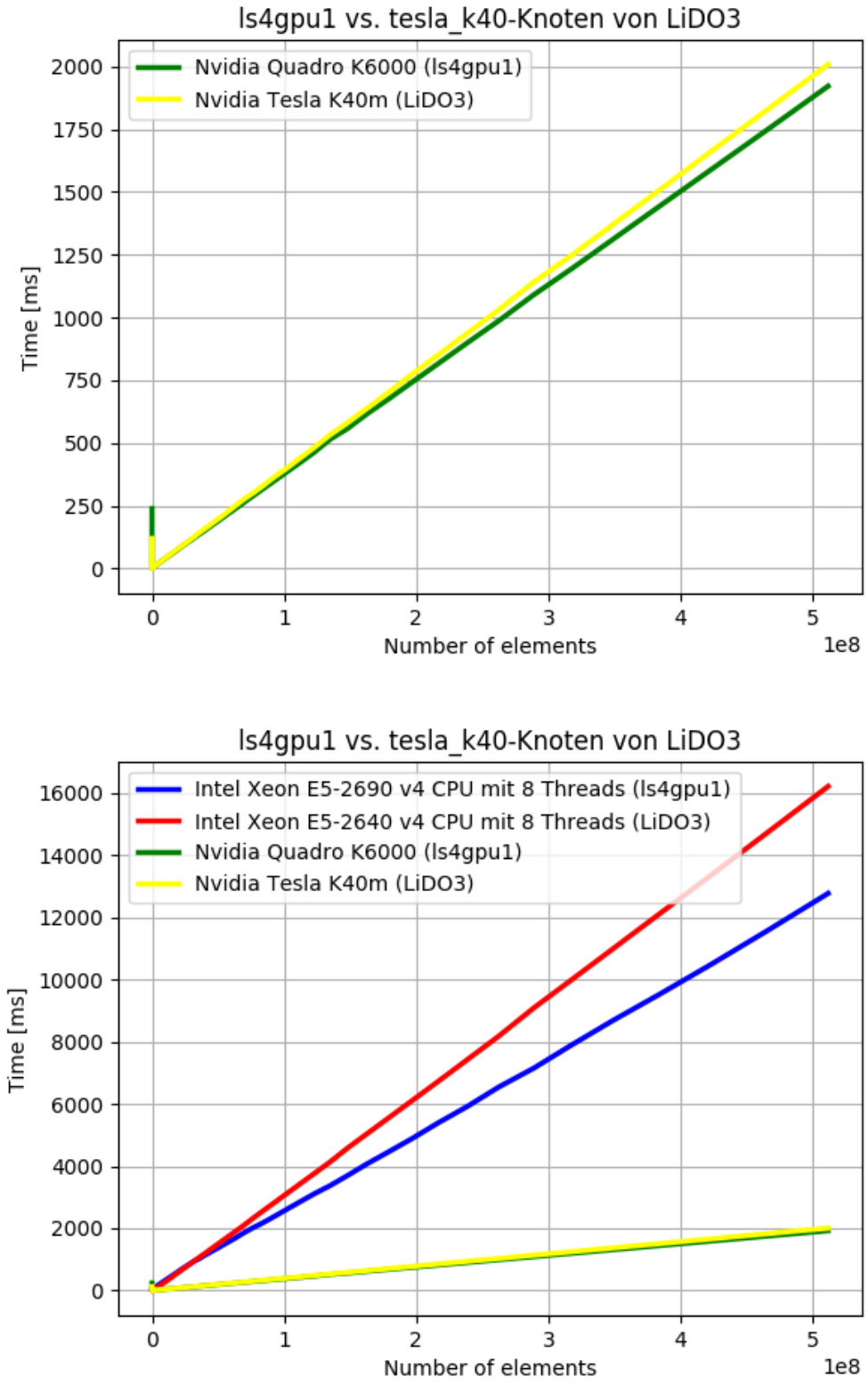


**Abbildung 2.11:** Das Ergebnis der Workstation ls4gpu1 und eines *tesla_ k40*-Knotens von LiDO3 für 0 bis $5,2 * 10^8$ Elemente vom Typ *double*.

### 2.3.1 Fazit

Die Ergebnisse in Abbildung 2.10 und 2.11 zeigen deutlich, dass der Einsatz von GPUs die Ausführungszeit von Algorithmen sehr stark beschleunigen kann. Die parallele Implementierung des Mergesort-Algorithmus für CPUs ist je nach CPU bis zu 8x langsamer als die GPU-Implementierung, insbesondere wenn der Rechenaufwand sehr hoch ist.

Beim Vergleich zwischen NVIDIA® Quadro® K6000 (ls4gpu1) und Tesla® K40m (*tesla_k40*-Knotens von LiDO3) ist zu sehen, dass die K40m erst bei einem (sehr) hohen Rechenaufwand ihr volles Potential ausnutzen kann und etwas schneller ist als die K6000. Ist die zu leistende Arbeit jedoch relativ klein so besitzt die K6000 ein kleinen Vorteil.

**Anmerkung:** Die langsame Ausführungszeit der beiden GPUs bei der ersten Zeitmessung (hoher Balken im Diagramm), ist auf die Initialisierungszeit der GPUs zurückzuführen, die nicht separat vor der Zeitmessung durchgeführt wird.

# Anhang A

# Hardwaredetails

Die Workstation `ls4gpu1` ist mit dem Befehl `ssh ls4gpu1.cs.tu-dortmund.de` erreichbar. Sie enthält eine Intel® Xeon® E5-2690 v4 CPU und eine NVIDIA® Quadro® K6000 Grafikkarte. Der Arbeitsspeicher (RAM) beträgt 32 GB.

Die Workstation `ls4gpu2` ist mit einer Intel® Xeon® E5-2699 v4 CPU und zwei NVIDIA® Quadro® P6000 Grafikkarten ausgestattet. Außerdem sind 64 GB RAM enthalten. Erreichbar ist die Workstation mit dem Befehl `ssh ls4gpu2.cs.tu-dortmund.de`.

Ein *tesla_k40*-Knoten von LiDO3 ist mit zwei Intel® Xeon® E5-2640 v4 CPU und zwei NVIDIA® Tesla® K40m Grafikkarten ausgestattet. Der RAM kann je nach Bedarf per `salloc`- und `#SBATCH`-Befehl spezifiziert werden.

## A.1 CPU-Informationen

In der Tabelle A.1 sind die wichtigsten Informationen zum Prozessor Intel® Xeon® E5-2690 v4 (ls4gpu1), E5-2699 v4 (ls4gpu2) und E5-2640 v4 (*tesla_k40*-Knoten von LiDO3) zu finden.

## A.2 GPU-Informationen

In der Tabelle A.2 sind die wichtigsten Informationen zu den Grafikkarten NVIDIA® Quadro® K6000 (ls4gpu1) und P6000 (ls4gpu2) sowie Tesla® K40m (*tesla_k40*-Knoten von LiDO3) zu finden.

| | **Intel® Xeon® E5-2690 v4** (ls4gpu1) | **Intel® Xeon® E5-2699 v4** (ls4gpu2) | **Intel® Xeon® E5-2640 v4** (*tesla_k40*-Knoten von LiDO3) |
|---|---|---|---|
| Number of Cores | 14 | 22 | 10 |
| Number of Threads | 28 | 44 | 20 |
| Processor Base Frequency | 2,60 GHz | 2,20 GHz | 2,40 GHz |
| Max Turbo Frequency | 3,50 GHz | 3,60 Ghz | 3,40 GHz |
| Cache | 35 MB SmartCache | 55 MB SmartCache | 25 MB SmartCache |
| Bus Speed | 9,6 GT/s QPI | 9,6 GT/s QPI | 8 GT/s QPI |
| Number of QPI Links | 2 | 2 | 2 |
| TDP | 135 W | 145 W | 90 W |
| **Memory Specifications** | | | |
| Max Memory Size | 1,54 TB | 1,54 TB | 1.54 TB |
| Memory Types | DDR4 1600/1866/2133/2400 | DDR4 1600/1866/2133/2400 | DDR4 1600/1866/2133 |
| Max Number of Memory Channels | 4 | 4 | 4 |
| Max Memory Bandwidth | 76,8 GB/s | 76,8 GB/s | 68.3 GB/s |
| Physical Address Extensions | 46-bit | 46-bit | 46-bit |
| ECC Memory Supported | Yes | Yes | Yes |
| **Expansion Options** | | | |
| Scalability | 2S | 2S | 2S |
| PCI Express Revision | 3.0 | 3.0 | 3.0 |
| PCI Express Configurations | x4, x8, x16 | x4, x8, x16 | x4, x8, x16 |
| Max Number of PCI Express Lanes | 40 | 40 | 40 |

**Tabelle A.1:** Intel® Xeon® E5-2690 v4 (ls4gpu1), E5-2699 v4 (ls4gpu2) und E5-2640 v4 (*tesla_k40*-Knoten von LiDO3) Spezifikation. [int16b, Int16c, Int16a]

| | **Quadro® K6000** (ls4gpu1) | **Quadro® P6000** (ls4gpu2) | **Tesla® K40m** (*tesla_k40*-Knoten von LiDO3) |
|---|---|---|---|
| CUDA Driver Version / Runtime Version | 6.5 / 6.0 | 8.0 / 8.0 | 9.2 / 8.0 |
| CUDA Capability Major/Minor version number | 3.5 | 6.1 | 3.5 |
| Total amount of global memory: | 12280 MBytes | 24443 MBytes | 11441 MBytes |
| CUDA Cores | 2880 | 3840 | 2880 |
| Multiprocessors | 15 | 30 | 15 |
| CUDA Cores per Multiprocessor | 192 | 128 | 192 |
| GPU Max Clock rate | 902 MHz | 1645 MHz | 876 MHz |
| Memory Clock rate | 3004 Mhz | 4513 Mhz | 3004 Mhz |
| Memory Bus Width | 384-bit | 384-bit | 384-bit |
| L2 Cache Size | 1,5 MiB | 3 MiB | 12 MiB |
| Constant memory: | 64 KiB | 64 KiB | 64 KiB |
| Shared memory per block | 48 KiB | 48 KiB | 48 KiB |
| Registers per block | 65536 | 65536 | 65536 |
| Warp size | 32 | 32 | 32 |
| Maximum number of threads per multiprocessor | 2048 | 2048 | 2048 |
| Maximum number of threads per block | 1024 | 1024 | 1024 |
| Max dimension size of a thread block (x,y,z) | (1024, 1024, 64) | (1024, 1024, 64) | (1024, 1024, 64) |
| Max dimension size of a grid size (x,y,z) | (2147483647, 65535, 65535) | (2147483647, 65535, 65535) | (2147483647, 65535, 65535) |
| Maximum memory pitch | 2 GiB | 2 GiB | 2 GiB |
| Concurrent copy and kernel execution | Yes with 2 copy engine(s) | Yes, with 2 copy engines | Yes, with 2 copy engines |
| Run time limit on kernels | Yes | No | No |
| Integrated GPU sharing Host Memory | No | No | No |
| Support host page-locked memory mapping | Yes | Yes | Yes |
| Device has ECC support | Disabled | Disabled | Enabled |
| Device supports Unified Addressing (UVA) | Yes | Yes | Yes |

**Tabelle A.2:** Nvidia® Quadro® K6000 (ls4gpu1) und P6000 (ls4gpu2) sowie Tesla® K40m (*tesla_k40*-Knoten von LiDO3) Spezifikation. (Quelle: *deviceQuery* aus den CUDA Samples [CUD18a, CUD18c] siehe: tests/CUDA/cudaSamples/)

# Literaturverzeichnis


[Bee18] Homepage von BeeGFS - The Leading Parallel Cluster File System. https://www.beegfs.io, 2018. [Online; zuletzt abgerufen: 6. August 2018].

[CUD18a] CUDA Samples :: CUDA Toolkit Documentation (v9.2.148). https://docs.nvidia.com/cuda/cuda-samples/index.html, 2018. [Online; zuletzt abgerufen: 1. September 2018].

[CUD18b] CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/, 2018. [Online; zuletzt abgerufen: 26. September 2018].

[CUD18c] GitHub - NVIDIA/cuda-samples: Samples for CUDA Developers which demonstrates features in CUDA Toolkit. https://github.com/NVIDIA/cuda-samples, 2018. [Online; zuletzt abgerufen: 1. September 2018].

[dTUD17] IT & Medien Centrum (ITMC) der Technische Universität Dortmund. Dienstleistungskatalog des ITMC - Version 2017/10. https://www.itmc.tu-dortmund.de/cms/Medienpool/pdfs/171006_Dienstleistungskatalog_2017_ITMC_web.pdf, 2017. [Online; zuletzt abgerufen: 6. August 2018].

[fIICdTD18] Ingo Schulz (Fakultät für Informatik / ITMC CC:HPC der TU Dortmund). Einführung in LiDO3. https://www.lido.tu-dortmund.de/cms/de/LiDO3/lido3kurz.pdf, 2018. [Online; zuletzt abgerufen: 6. August 2018].

[Int16a] Intel® Xeon® Processor E5-2640 v4 Specification. https://ark.intel.com/products/92984/Intel-Xeon-Processor-E5-2640-v4-25M-Cache-2_40-GHz, 2016. [Online; zuletzt abgerufen: 30. August 2018].

[int16b] Intel® Xeon® Processor E5-2690 v4 Specification. https://ark.intel.com/products/91770/Intel-Xeon-Processor-E5-2690-v4-35M-Cache-2_60-GHz, 2016. [Online; zuletzt abgerufen: 12. Juni 2018].

[Int16c] Intel® Xeon® Processor E5-2690 v4 Specification. https://ark.intel.com/en/products/91317/Intel-Xeon-Processor-E5-2699-v4-55M-Cache-2_20-GHz, 2016. [Online; zuletzt abgerufen: 30. August 2018].

[Sal18] salloc Manpage. https://slurm.schedmd.com/salloc.html, 2018. [Online; zuletzt abgerufen: 14. August 2018].

[Sba18] sbatch Manpage. https://slurm.schedmd.com/sbatch.html, 2018. [Online; zuletzt abgerufen: 14. August 2018].

[SCP13] scp() - Linux man page. https://linux.die.net/man/1/scp, 2013. [Online; zuletzt abgerufen: 6. August 2018].

[Slu18] Homepage vom Slurm Workload Manager. https://slurm.schedmd.com/, 2018. [Online; zuletzt abgerufen: 14. August 2018].

[Sor18] Sorting data in parallel CPU vs GPU. https://solarianprogrammer.com/2013/02/04/sorting-data-in-parallel-cpu-gpu/, 2018. [Online; zuletzt abgerufen: 26. September 2018].

[Thr18] Thrust - Parallel Algorithms Library. https://thrust.github.io/, 2018. [Online; zuletzt abgerufen: 26. September 2018].