Belegarbeit Computational Science and Engineering: Verteilte GPGPU-Berechnungen mit Spark

Maximilian Knespel

Betreuer: Dipl.-Inf. Nico Hoffmann

GPU-Beschleuniger zum Hochleistungsrechnen

NVIDIA

Rainer Knäpper, Free Art License (artlibre.org/licence/lal/en/)

Aktueller (Juni 2016) Platz 3 Titan XK7 in der Top 500:

- 18 688 AMD Opteron 6274 (16 Kerne) (140 TDPFLOPS)
- 18 688 Nvidia Tesla K20X (1.3 TDPFLOPS)

General Purpose Graphical Processing Units(GPGPU) im Vergleich Prozessoren:

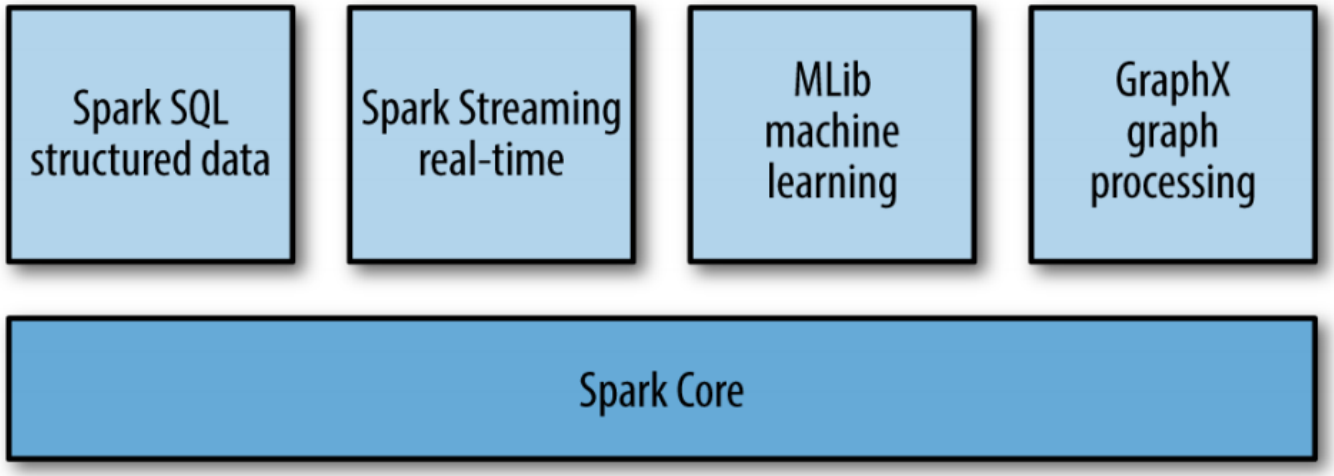
+ Kosteneffektiv in Anschaffung
+ sehr gute Leistungsaufnahme pro GFlops
– nur für bestimmte Anwendungen geeignet
+ hohe Speicherbandbreiten
– niedrige Speicherlatenzen
+ Massiv parallel (3840 CUDA cores gegen 24 cores Xeon E7-8890-v4 (AVX2)
– **zusätzliche Komplexität beim Programmieren**

Spark Übersicht

Nachfolger zu Hadoop MapReduce. Geschrieben in Scala. Vorteile gegenüber von Hadoop MapReduce:

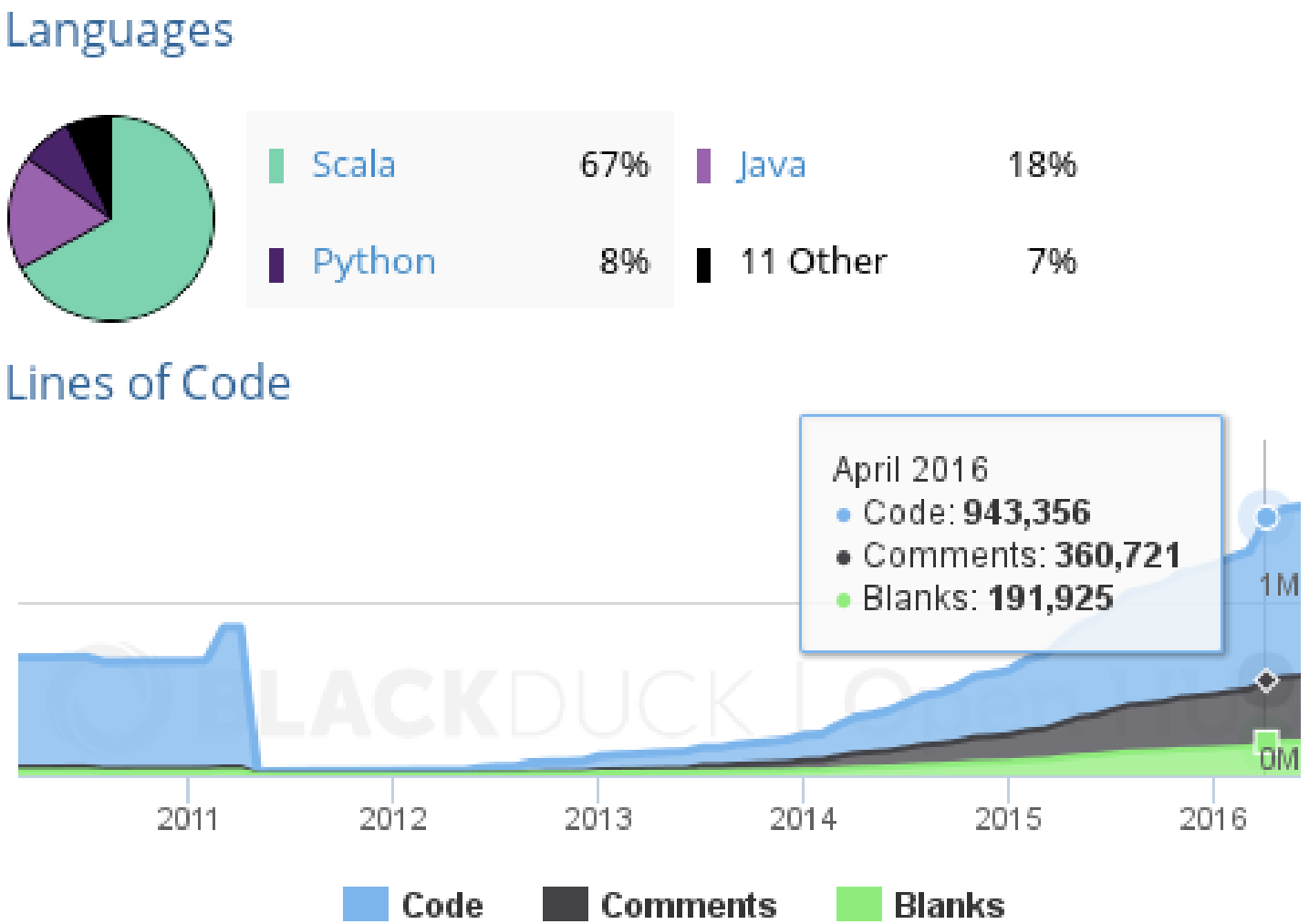
+ Kann im Arbeitsspeicher arbeiten
+ Stellt viele Komplexbefehle zur Verfügung
+ iterative Algorithmen schneller als Hadoop wegen cache/persist-Funktionen
+ interaktive Konsole
+ Unterstützung für: Scala, Java, Python, ...
+ Im Gegensatz zu zu Hadoop MapReduce auch lokales Dateisystem nutzbar

Spark Anwendungen



Spark auf Github

`github.com/apache/spark.gitmaster`



`https://www.openhub.net/p/apache-spark`

MapReduce Programmiermodell

- 2008: Paper "MapReduce: Simplified Data Processing on Large Clusters" von Jeffrey Dean und Sanjay Ghemawat (Google Inc.)

Resilient Distributed Datasets (RDDs)

- zu bearbeitende Daten liegen in RDD-Objekten
- Methoden zur Verarbeitung der Daten

```
def distinct(numPartitions: Int): RDD[T]
```

Return a new RDD containing the distinct elements in this RDD.

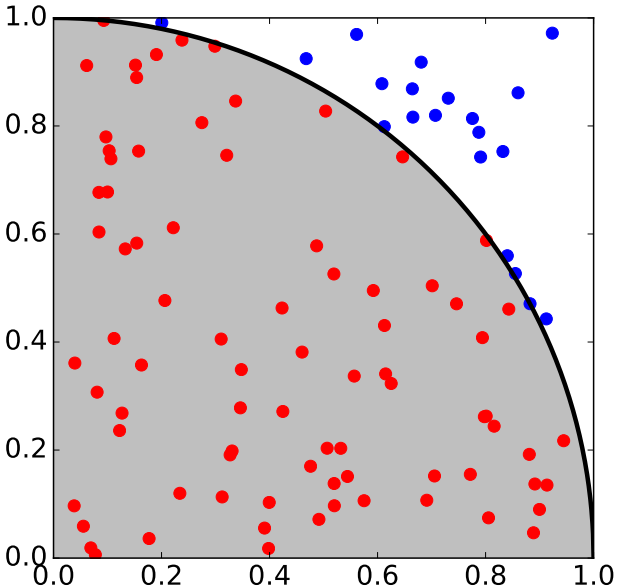
```
def filter(f: (T) ⇒ Boolean): RDD[T]
```

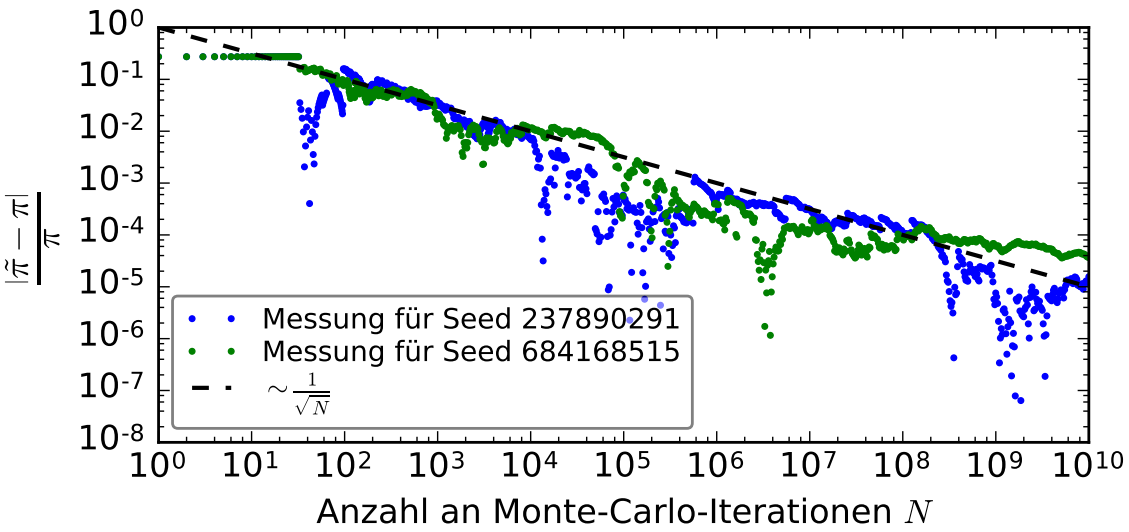
Return a new RDD containing only the elements that satisfy a predicate.

https://spark.apache.org/docs/0.8.1/api/core/org/apache/spark/rdd/RDD.html

- distributed:
- resilient: opake Neuberechnung der Teildaten bei Absturz eines Knotens

Monte-Carlo-Integration





Spark starten

```
"$SPARK_ROOT"/bin/spark-shell --master local[*]
```

```
val seed0 = 875414591
val nPartitions = 100
val nIterationsPerPartition = 1e6.toLong
val quarterPis = sc.
  parallelize( 0 until nPartitions ).
  map( iRank => ( {
    val seed  = ( seed0 + ( iRank.toDouble /
    nPartitions * Integer.MAX_VALUE ).toLong ) %
    Integer.MAX_VALUE
    calcQuarterPi( nIterationsPerPartition, seed ) ).
  cache
quarterPis.take(4).foreach( x => print( x + " " ) )
println( "pi = " + 4 * quarterPis.reduce(_+_) /
    nPartitions )
```

Output:

```
0.7852178 0.7852558 0.7855507 0.78554
pi = 3.1415955324
```

Scala GPU: - ScalaCL ("ScalaCL is not production-ready!") - BIDMach - Firepile - Rootbeer

Rootbeer Funktionsweise

...

```
1       jobid=$(sbatch "$@" --output="$SPARK_LOGS/%j.
    out" --error="$SPARK_LOGS/%j.err" $HOME/scaromare
    /start_spark_slurm.sh)
2       jobid=${jobid##Submitted batch job }
3
4
5       function startSpark() {
6           export SPARK_LOGS=$HOME/spark/logs
7           mkdir -p "$SPARK_LOGS"
8           if [ ! -d "$SPARK_LOGS" ]; then return 1;
     fi
9           jobid=$(sbatch "$@" --output="$SPARK_LOGS
    /%j.out" --error="$SPARK_LOGS/%j.err" $HOME/
    scaromare/start_spark_slurm.sh)
10          jobid=${jobid##Submitted batch job }
11          echo "Job ID : $jobid"
12          # looks like: 16/05/13 20:44:59 INFO
    MasterWebUI: Started MasterWebUI at http
    ://172.24.36.19:8080
13          echo -n "Waiting for Job to run and Spark
     to start.."
14          MASTER_WEBUI=''
15          while [ -z "$MASTER_WEBUI" ]; do
```

Rootbeer-Erfahrungen

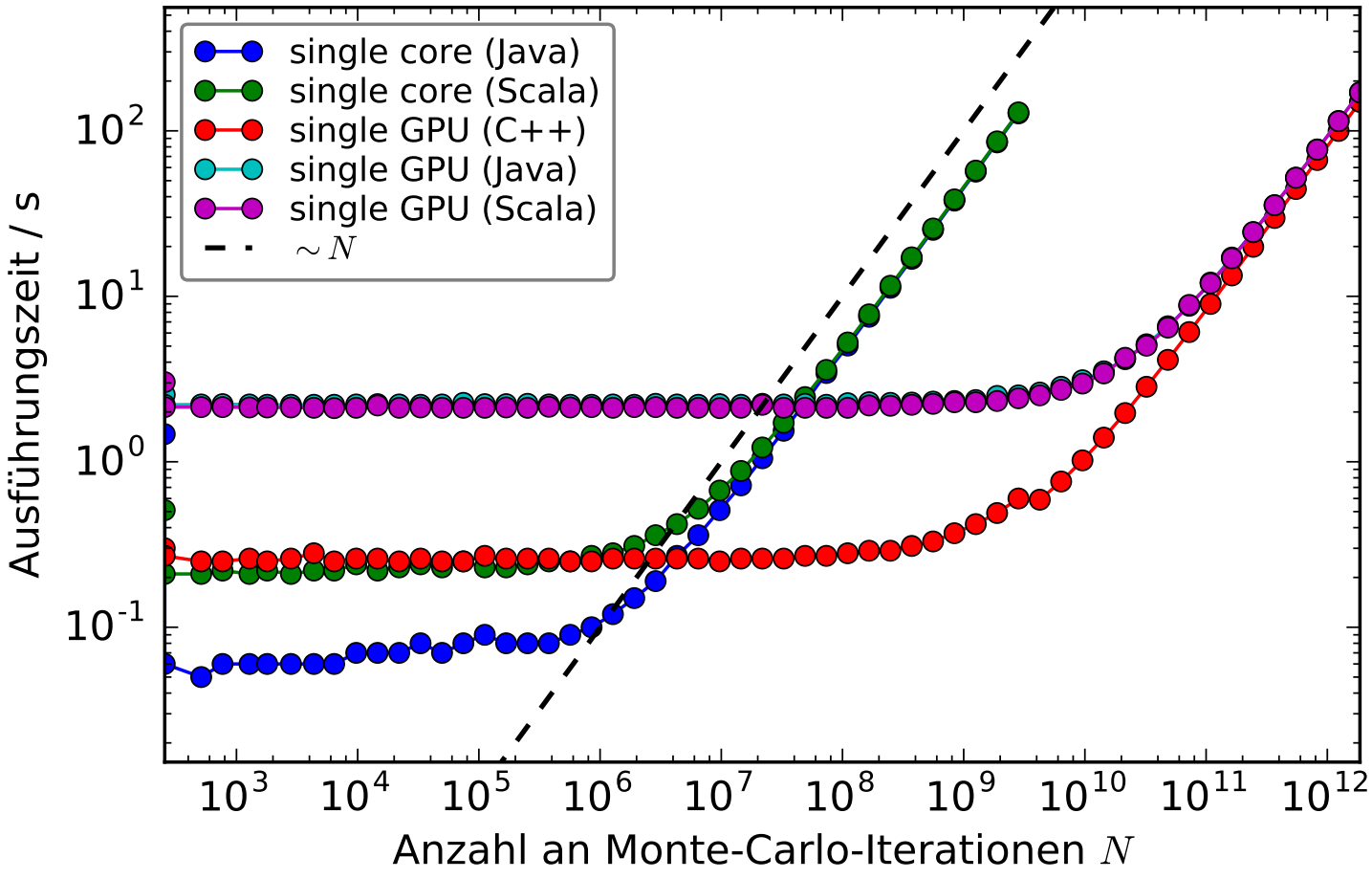
- Bei dem Versuch Rootbeer mit Java 8 zu benutzen, kommt es zu

```
1                    java.lang.NullPointerException
2                        at soot.rbclassload.
    RootbeerClassLoader.loadHierarchySootClasses(
    RootbeerClassLoader.java:963)
3
```

  Nur Java 6 ist offiziell unterstützt, aber Java 7 scheint auch zu funktionieren
- GCC 4.9 nur unterstützt (wegen CUDA-Compiler)
- Kaum neue Commits auf Github seit Juni 2015
- Die option `-computecapability=sm_30` ist Pflicht, da seit CUDA 7.0 die Standardarchitektur `compute_12` nicht mehr unterstützt wird (korrekt nun `sm_12`)

```
1                    nvcc fatal   : Unsupported gpu
    architecture 'compute_12'
```

Leistungsanalyse auf einem CPU-Kern



Profiling von Rootbeer ohne Spark mit dem NVIDIA Visual Profiler

```
1 srun -p gpu-interactive --nodes=1 --ntasks-per-node=1
      --cpus-per-task=1 --gres=gpu:2 --time=1:30:00 --
     mem-per-cpu=6000 --x11=first nvvp
2 Executable: /sw/global/tools/java/jdk1.7.0_25/bin/
     java
3 Working Directory: ...
4 Arguments: -jar ./MontePi.jar 2684354560
```

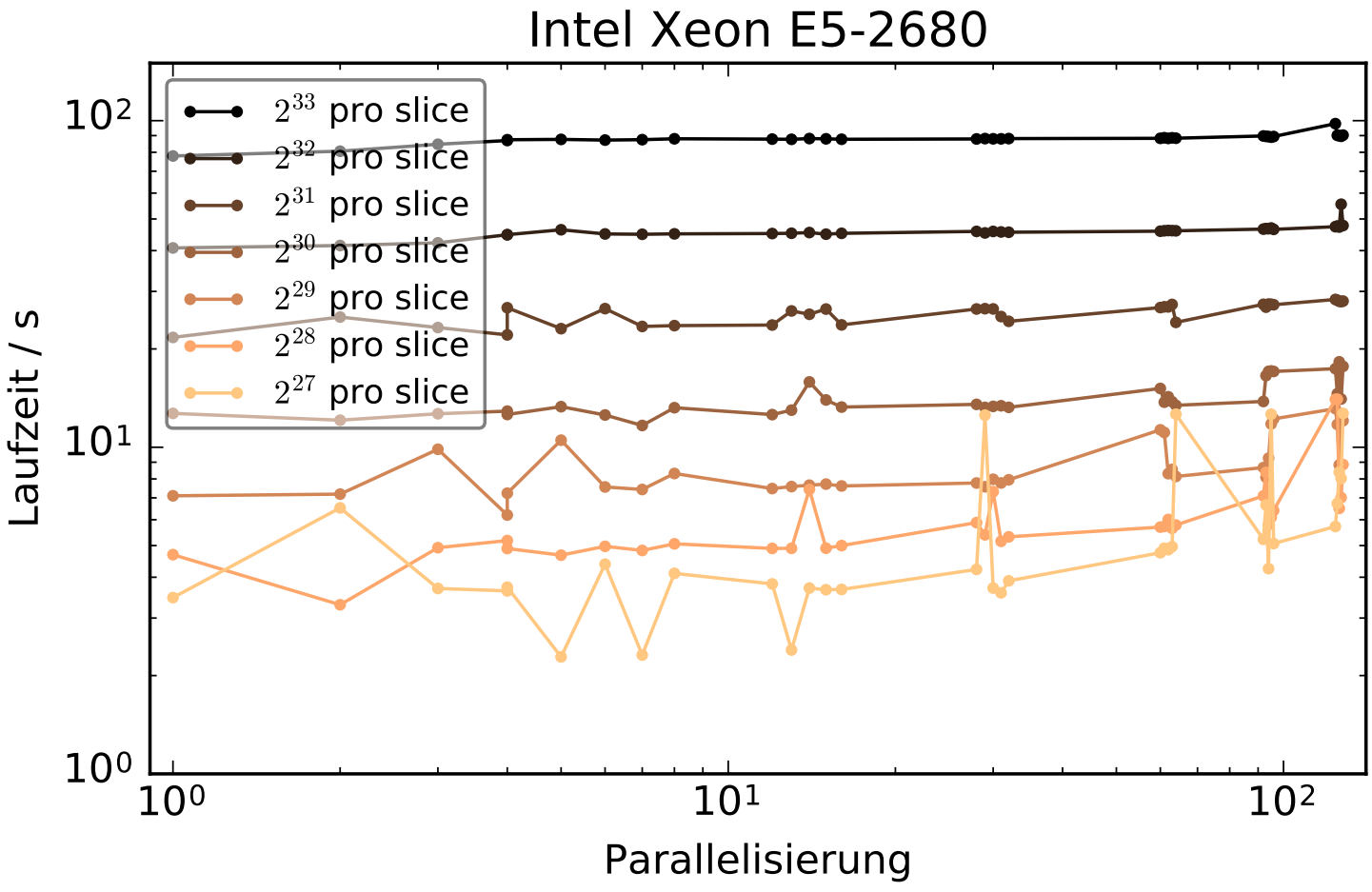
-> BILDER!

Spark auf Taurus über Slurm starten

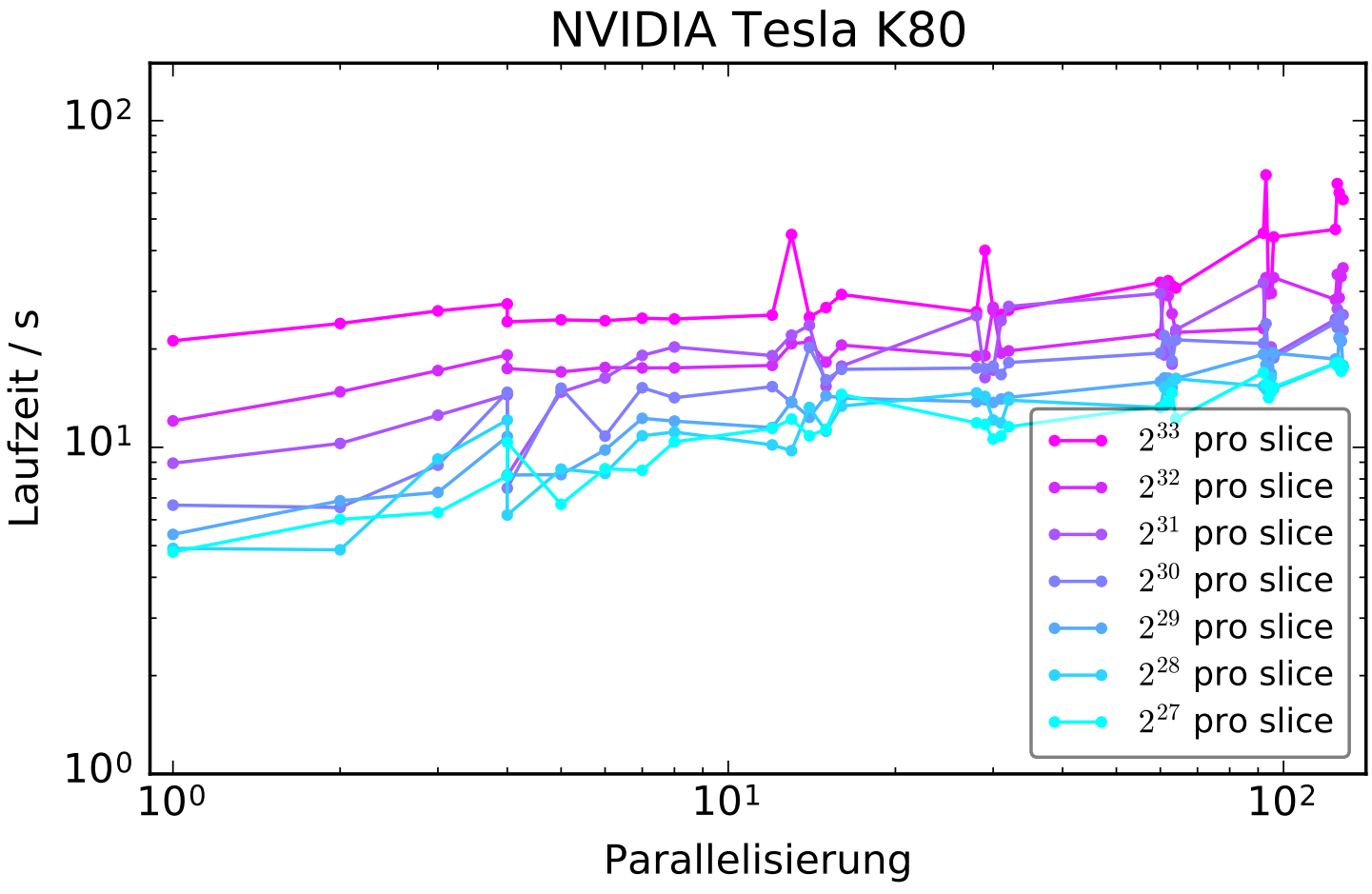
```
1        module load java/jdk1.7.0_25 scala/2.10.4
    cuda/7.0.28
2        # manuelle Installation von maven, ant, spark
    , zipmerge
3        % üfr spark wurden precompiled binaries
    genommen, weil es bei der Kompilation auch
    Probleme gab, die nicht behoben werden konnten
4        export PATH="$PATH:$HOME/spark-1.5.2-bin-
    hadoop2.6/bin/:$HOME/spark-1.5.2-bin-hadoop2.6/
    sbin/:$HOME/programs/bin"
5
```

...

Benchmark Spark auf CPU



Benchmark Spark mit Rootbeer (alte Ergebnisse)



Zusammenfassung

- Kombination aus GPGPU mittels Rootbeer und Spark ausgetestet
- Mehrere Bugfixes für Rootbeer geschrieben

Ausblick:

- Codereview von Rootbeer oder andere GPU-API ist nötig
- heterogene Berechnungen auf CPU + GPU
- Erweiterung von Rootbeer um neue Features wie NVIDIA NVLink
- Implementation direkt in Spark würde z.B. cache/persist auf GPUs erlauben, um Host-GPU-Transfers zu sparen