HTWK Leipzig
Fachbereich IMN
Wintersemester 2012/2013

Ausarbeitung zum Fach Message-Passing-Programmierung –VORABVERSION–

Beleg im Fach Message-Passing-Programmierung

Kurt Junghanns
Philipp-Rosenthal-Straße 32
04103 Leipzig
kurt.junghanns@stud.htwk-leipzig.de

Marcel Kirbst
Sieglitz 39
06618 Molau
marcel.kirbst@stud.htwk-leipzig.de

30. Januar 2013

Inhaltsverzeichnis

1 Einleitung

Diese Ausarbeitung ist das Resultat der Veranstaltung Message-Passing-Programmierung im Wintersemester 2012/2013 und präsentiert die eingereichten Programme als Grundlage der mündlichen Prüfung der Prüflinge Kurt Junghanns und Marcel Kirbst.
Die Aufgabenstellung erfordert die Bearbeitung von zwei Aufgaben, die auf unterschiedlichen Hardware-Plattformen zu implementieren waren.

2 Message-Passing-Interface (MPI)

2.1 Aufgabenstellung / Problembeschreibung

Die empfohlene Aufgabenstellung für die MPI-Teilaufgabe ist die Implementierung eines so genannten Merge-Splitting-Sort-Algorithmus, der eine vorzugebende Anzahl natürlicher Zahlen in zufälliger Reihenfolge auf einer vorzugebenden Anzahl an Prozessoren sortiert. Dabei soll die benötigte Laufzeit ermittelt werden um im Anschluß Aussagen über das Laufzeitverhalten der Implementierung in Abhängigkeit zur verwendeten Element- und Prozessorzahl treffen zu können.
Dieser Algorithmus wurde in einem C-Programm unter Zuhilfenahme der MPI Bibliothek umgesetzt. Nachfolgend werden Aussagen zum Laufzeitverhalten getroffen. Dabei wurden den Laufzeitmessungen die Anzahl zu sortierender Elemente wie in der Aufgabenstellung empfohlen mit 20.000, 40.000 sowie 80.000 Elementen zu Grunde gelegt.

2.2 Programmbeschreibung

```
1  #include "mpi.h"
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <time.h>
5
6        //ungerade Prozessornummer
7
8        //erhalte Array
9        MPI_Recv(&local[nLocal], nLocal, MPI_INT, rank_world+1, 1, MPI_COMM_WORLD, ↩
             status);
10
11       //sortiere Array
12       wtimesinnersort[j*2+0] = MPI_Wtime();
13       quicksort(local, 0, nLocal*2-1);
14       wtimesinnersort[j*2+1] = MPI_Wtime();
15
16       //obere Teil des Arrays wird an Prozessor rank_world+1 gesendet
```

```
17        MPI_Send(&local[nLocal], nLocal, MPI_INT, rank_world+1, 1, MPI_COMM_WORLD);
18    }
```

Listing 1: MPI C-Programm: cluster.c

2.3 Laufzeitumgebung

Um die Entwicklung und die Tests der Implementierung so effektiv wie möglich zu gestalten, wurden mehrere BASH-Skripte erstellt. Das BASH-Skript `run.sh`, dass im Anhang vollständig aufgeführt ist, erfüllt dabei die folgenden Funktionen:

- Ermitteln der Prozessoranzahl **p**, Anzahl der zu sortierenden Elemente **n**, Name der zu kompilierenden C-Datei, Name der kompilierten Binärdatei als Startparameter
- prüfen, welcher der Rechner im Pool per SSH erreichbar sind
- ermitteln der durchschnittlichen Auslastung aller errichbaren Rechner im Pool
- sortieren der erreichbaren Poolrechner aufsteigend nach der durchschnittlichen Auslastung der letzten Minute, der letzten 5 Minuten, der letzten 15 Minuten
- Kompilieren der angegebenen C-Datei
- Ausführung der resultierenden Binärdatei auf den **p** Rechnern mit der geringsten durchschnittlichen Auslastung um das Risiko einer Verfälschung der Messergebnisse durch Fremdeinwirkung zu minimieren

Ein weiteres weiteres BASH-Skript `bench.sh` ruft die Binärdatei mit den empfohlenen Elementanzahlen 20.000, 40.000 und 80.000 sequentiell für 2, 4, 8, 10, 16 und 20 Prozessoren auf und gibt die jeweils gemessenen Zeitintervalle übersichtlich aus um eine grafische Auswertung mit gängigen Tabellenkalkulationsprogrammen zu gestatten.

2.4 Ergebnisse und Auswertung

2.4.1 Initiale Phase

In der Initialphase des Merge-Splitting-Sort-Algorithmus wird das lokale Array eines jeden Prozessors initial sortiert. Da die Initialphase im Gegensatz zu den nachfolgenden Phasen unabhängig von der Anzahl der genutzen Prozessoren immer nur einmalig durchlaufen wird, sinkt der Anteil der Initialphase an der Gesamtlaufzeit des Algorithmus mit steigender Prozessoranzahl. Während sich bei der Nutzung von nur zwei Prozessoren noch ein Laufzeitanteil der Phase 1 von 50 Prozent ergibt, sinkt

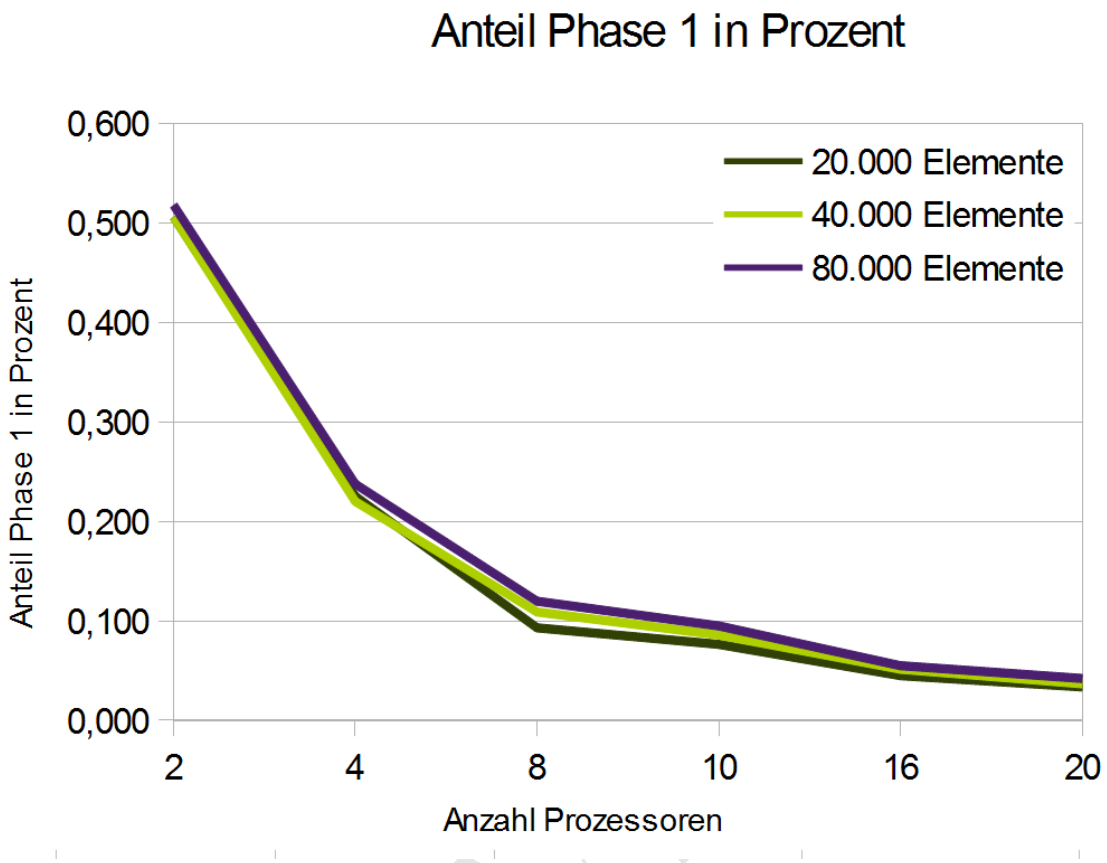

Abbildung 1: Phase-1-Diagramm für die MPI-Implementierung

dieser Wert bei Nutzung von 10 Prozessoren schon auf unter 10 Prozent, Tendenz weiter fallend.

2.4.2 Speedup

Der Speedup ist der Quotient aus der Laufzeit des Algorithmus bei der Nutzung eines Prozessors und der Laufzeit bei Nutzung mehrerer Prozessoren. Ein Speedup-Wert von 1 sagt aus, dass der Algorithmus auf einem Prozessor der Laufzeit des Algorithmus auf mehreren Prozessoren entspricht. Im Idealfall steigt der Speedup proportional mit der Anzahl der Prozessoren.

Nach Ahmdal setzt sich die Gesamtlaufzeit des parallelisierten Algorithmus zusammen aus einem Anteil mit nichtparallelisierbaren Code (sog.: sequenzieller Anteil) und einem Anteil an parallelisierbaren Code, dessen Laufzeit sich umgekehrt proportional zur Anzahl der benutzten Prozessoren verhält. Ahmdals Gesetz berücksichtigt

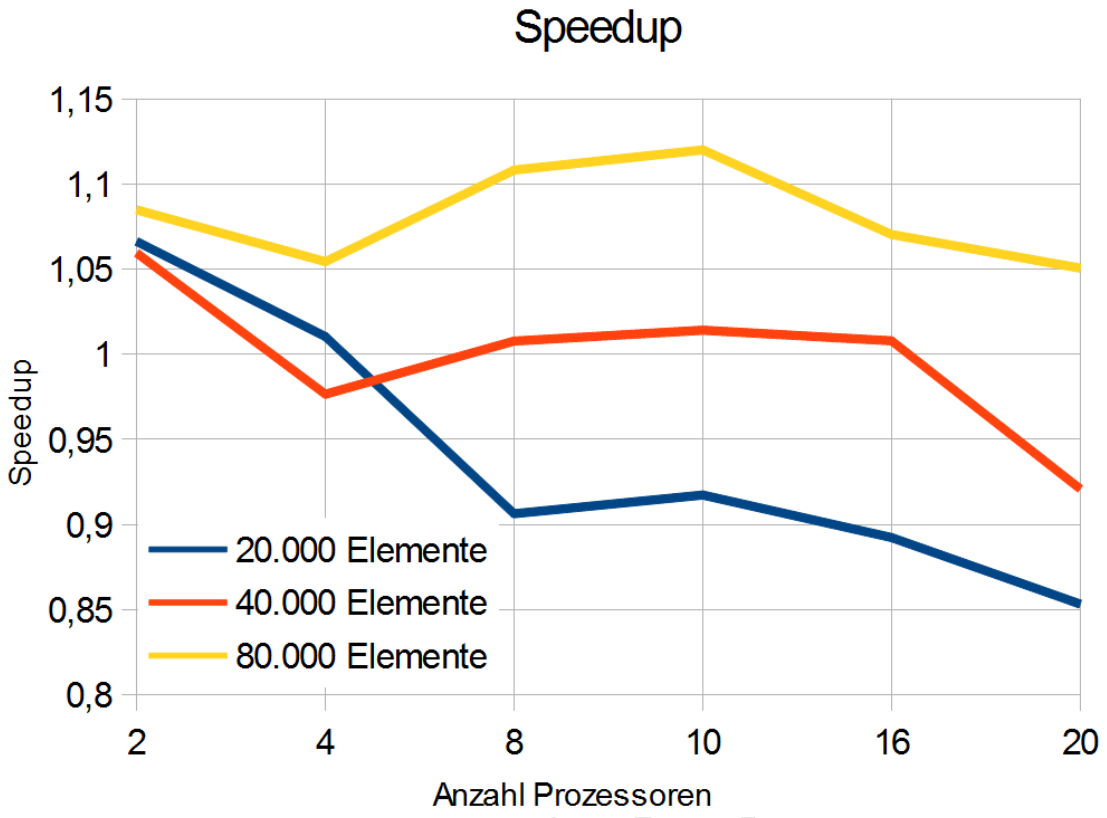

Abbildung 2: Speedup-Diagramm für die MPI-Implementierung

hierbei jedoch nicht den mit steigender Prozessoranzahl ebenfalls wachsenden Kommunikationsaufwand.

Das zu erwartende Laufzeitverhalten für reale Implementierungen legt daher nahe, das der Speedup nicht linear mit der Anzahl der eingesetzten Prozessoren ansteigt, sondern auf Grund des ebenfalls ansteigenden Kommunikationsaufwandes ab einer bestimmten Prozessoranzahl wieder abnimmt.

Die durchgeführten Laufzeitmessungen mit der Implementierung des Algorithmus zeigen jedoch, dass bereits bei Nutzung von mehr als 10 Prozessoren der Speedup mit steigender Prozessoranzahl abnimmt. Der im Test beste erreichte Speedup stellte sich bei Nutzung von 10 Prozessoren und hinreichend vieler Elemente ein (>= 80.000). Bereits beim Einsatz von von 16 Prozessoren war die Laufzeitverringerung gegenüber der vollständig sequenziellen Implementierung nur noch marginal, Tendenz abnehmend. Dieses von den theoretisch erwarteten Messwerten abweichende

Laufzeitverhalten ist das Ergebnis weiterer Einflussfaktoren wie beispielsweise:

- Eingesetzte Hardware (Netzwerkstruktur, nicht exklusiv genutzte Hardware)
- Eingesetzte Software (Betriebssystem, genutze Implementierung des Message-Passing-Interface)
- Implementierung des Algorithmus (eingesetzter Sortieralgorithmus, Kommunikationsablauf)

Im Laufe der Implementation wurde ein direkter Einfluss des verwendeten Sortieralgorithmus auf die Gesamtlaufzeit deutlich. Es wurden verschiedene Quicksort-Implementationen getestet, wobei durch die in der Standardbibliothek von C enthaltene Funktion `qsort()` die besten Ergebnisse liefert.

2.4.3 Effizienz

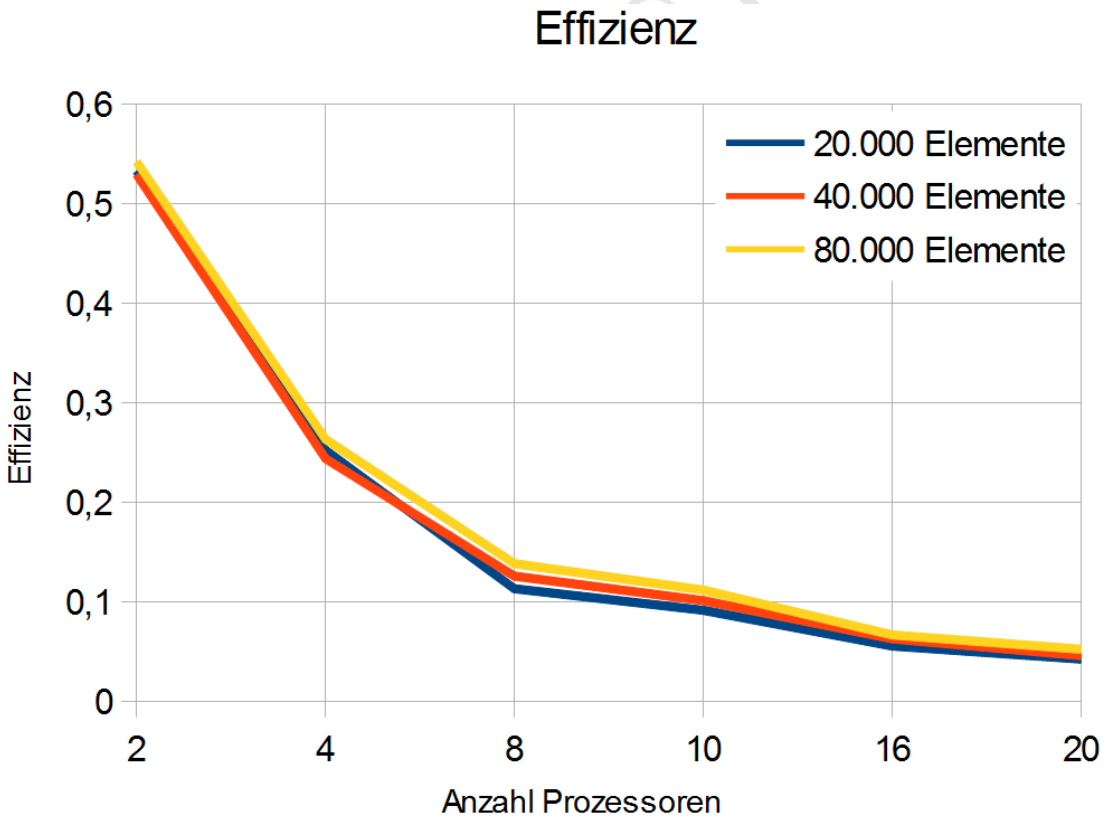

Abbildung 3: Effizienz-Diagramm für die MPI-Implementierung

Eine weiterer aussagekräftiger Wert ist die Effizienz. Die Effizienz gibt die relative Verbesserung in der Verarbeitungsleistung an und ergibt sich aus dem Quotient von

Speedup und Prozessoranzahl. Wie aus dem betreffenden Diagramm ersichtlich wird, nimmt die Effizienz umgekehrt proportional zur Anzahl der eingesetzten Prozessoren ab. Dabei hat die Anzahl der zu sortierenden Elemente nur marginalen Einfluss auf die jeweiligen Effizienzwerte.

2.4.4 Kommunikationsanteil

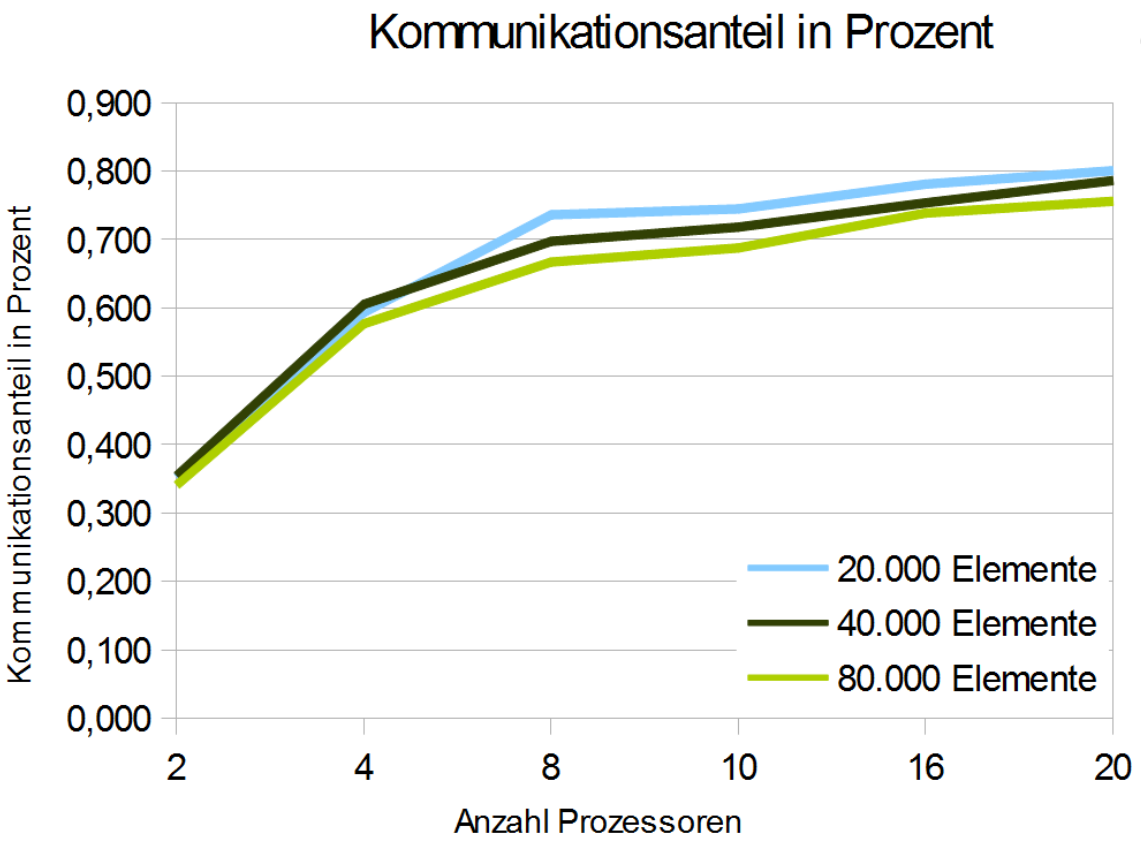

Abbildung 4: Diagramm des Kommunikationsanteils für die MPI-Implementierung

Die Auswertung der Messwerte zeigt, dass mit steigender Anzahl zu sortierender Elemente der Kommunikationsanteil an der Gesamtlaufzeit exponentiell ansteigt. Bereits bei Verwendung von 8 Prozessoren beträgt der Kommunikationsanteil an der Gesamtlaufzeit etwa 70 Prozent und steigt weiter an, sodass bereits für 20 Prozessoren der Kommunikationsanteil über 80 Prozent mit weiterhin steigender Tendenz beträgt.

3 Parallelrechnersystem MC3

3.1 Aufgabenstellung / Problembeschreibung

3.2 Programmbeschreibung

3.3 Laufzeitumgebung

3.4 Ergebnisse

3.5 Auswertung

4 Anhang

```
#!/bin/bash

##0##  process script args:
CPUCOUNT=0
VRANGE=0
INPUTFILENAME="cluster.c"
OUTPUTFILENAME="cluster"


function usage {
  echo "Usage: $0 -c CPUCOUNT -v VALUERANGE -i INPUTFILE -o OUTPUTFILE"
  exit 1;
}

##8 params required
if [ $# -ne 8 ] ; then ## erzwinge die Angabe aller Startparameter
  usage;
fi

##process args
while getopts c:hi:o:v: opt
do
  case "$opt" in
    c) CPUCOUNT=$OPTARG;;
    h) usage;;
    i) INPUTFILENAME=$OPTARG;;
    o) OUTPUTFILENAME=$OPTARG;;
    v) VRANGE=$OPTARG;;
    \?) usage;;
  esac
done
echo "CPUCOUNT: $CPUCOUNT"
echo "VRANGE:   $VRANGE"
echo "INPUT:    $INPUTFILENAME"
echo "OUT:      $OUTPUTFILENAME"

```

```
37
38 ##1##  compile
39 echo "STAGE 1 - compiling $INPUTFILENAME ..."
40 mpicc -Wall -o $OUTPUTFILENAME $INPUTFILENAME
41
42
43 ##2## create hostlist dynamically
44 echo "STAGE 2 - creating host list ..."
45 HOSTLISTFILENAME="load.txt"
46
47 ##remove already existing file without warning
48 touch $HOSTLISTFILENAME         ## create file if not already there
49 rm $HOSTLISTFILENAME            ## remove file
50
51 ##ssh trough simson clients for every pingable simson
52 for i in 01 02 03 04 05 06 07 08 09 {10..24}
53 do
54   ping -c 1 simson$i  > /dev/null
55   if [ $? = 0 ]
56   then
57  ## check per ssh cat /proc/loadavg and check with regex
58     echo "`ping -c 1 simson${i} | grep "64 bytes" | awk ' BEGIN {FS="("} {print $2}' | awk ' BEGIN {FS=")"} {print $1}'` `ssh simson${i} cat
59 /proc/loadavg`" | grep -v -E '141.57.9.[0-9]{2} $' >> $HOSTLISTFILENAME
60   fi
61 done
62 HOSTNR=`wc -l $HOSTLISTFILENAME | awk '{print $1}'`  ## zaehle Anzahl erreichbarer Hosts
63
64 ##remove already existing file without warning
65 touch $HOSTLISTFILENAME.sorted         ## create file if not already there
66 rm $HOSTLISTFILENAME.sorted            ## remove file
67
68 echo "Sortiere ${HOSTNR} Hosts nach Auslastung ..."
69 sort -k 2 $HOSTLISTFILENAME >> $HOSTLISTFILENAME.sorted
70 awk '{print $1}' $HOSTLISTFILENAME.sorted > $HOSTLISTFILENAME
71
72 echo "Zur Ausfuehrung werde folgenden ${CPUCOUNT} Hosts benutzt, da diese derzeit die geringste Auslastung haben:"
73 head -n ${CPUCOUNT} ${HOSTLISTFILENAME}.sorted > head.list
74 awk '{print "Node: " $1 " - Load on this Node:   " $2 " (avg last min)   " $3 " (avg last 5 min)   " $4 " (avg last 15 min) "}' head.list
75
76 sleep 1
77
78 ##3## run program on this hosts
79 echo "STAGE 3 - run $OUTPUTFILENAME on $CPUCOUNT cpus "
80 mpirun -np $CPUCOUNT -hostfile $HOSTLISTFILENAME $OUTPUTFILENAME $VRANGE
81
82 ##cleanup - remove temporary used files
83 #rm $HOSTLISTFILENAME
84 #rm $HOSTLISTFILENAME.sorted
```

Listing 2: MPI BASH-Script: run.sh

```
#!/bin/bash

## Initial run.sh aufrufen um Auslastung der Pool-Rechner zum jetzigen Zeitpunkt zu ermitteln
./run.sh -c 20 -v 20 -i cluster.c -o cluster

for val in 20000 40000 80000 # Anzahl der zu messenden n Elemente
do
  for cpu in 2 4 8 10 16 20  # fuer p Prozessoren
  do       # jeweils 5 Messungen
   mpirun -np $cpu -hostfile load.txt cluster $val
   mpirun -np $cpu -hostfile load.txt cluster $val
   mpirun -np $cpu -hostfile load.txt cluster $val
   mpirun -np $cpu -hostfile load.txt cluster $val
   mpirun -np $cpu -hostfile load.txt cluster $val
  done
done
```

Listing 3: MPI BASH-Script: bench.sh

5 Glossar

DHCP-Server DHCP steht als Abkürzung für "Dynamic Host Configuration Protokollünd beschreibt Techniken um Hosts in Netzwerken dynamisch Netzwerkparameter wie IP-Adressen zuzuweißen[1]

Router Ein Rechnersystem mit mindestens zwei Netzwerkschnittstellen, das Netzwerkschverkehr zwischen diesen Netzwerkschnittstellen nach einem Regelwerk vermittelt und weiterleitet.

Routerdistribution Eine spezielle Art von Betriebssystem, deren Hauptaugenmerk bei der Konzeption und Entwicklung darauf liegt Router-Funktionen sicher und stabil auszuführen

VLAN Die Abkürzung VLAN steht für Virtual Local Area Network und fasst Techniken zusammen um physikale Netzwerkstrukturen logisch zu Segmentieren, beispielsweise zur Erhöhung der Sicherheit oder um Broadcast-Domänen zu verkleinern.

[1] [6]

6 Literaturverzeichnis

Musterfrau, Renate: Muster. Frankfurt 2003.
Mustermann, Helmut: Noch ein Muster. Mit einer Einleitung hrsg. von Frank Muster. Frankfurt 2003.

Literatur

[1] http://www.ipcop.org/1.4.0/en/install/html/ , abrufbar am 16.12.2012

[2] http://www.ipcopwiki.de/index.php/Samba_Server , abrufbar am 20.12.2012
Anm.: Der Artikel zu diesem Addon ist zwar noch verfügbar, jedoch nicht die eigentlichen Dateien, die für das Addon erforderlich sind.

[3] http://www.ipcop.org/2.0.0/en/admin/html/whatsnew.html , abrufbar am 11.01.2013

[4] http://wiki.ipfire.org/de/addons/start , abrufbar am 10.01.2013

[5] http://www.freebsd.org/doc/de/books/faq/hardware.html#which-hardware-to-get , abrufbar am 12.01.2013

[6] http://www.isc.org/software/dhcp , abrufbar am 11.01.2013