INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Bachelorarbeit**

# Skalierungsverhalten eines Raspberry Pi-Clusters unter der Workload ausgewählter HPC-Benchmarks

Judith Greif

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN


**Bachelorarbeit**

# Skalierungsverhalten eines Raspberry Pi-Clusters unter der Workload ausgewählter HPC-Benchmarks

Judith Greif

| | |
|---|---|
| Aufgabensteller: | Prof. Dr. Dieter Kranzlmüller |
| Betreuer: | Dr. Nils gentschen Felde<br>Christian Straube |
| Abgabetermin: | 5. August 2014 |

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 31. Mai 2014

. . . . . .
*(Unterschrift der Kandidatin)*

**Abstract**

Seit dem Beginn seiner Entwicklung punktet der Mini-Computer Raspberry Pi durch Flexibilität, Preis-Leistungs-Verhältnis niedrigschwelligen Zugang und geringen Stromverbrauch. Das macht ihn zum idealen Kandidaten für einen Beowulf-Cluster. Er kann z.B. an Universitäten zur Forschungszwecken eingesetzt werden oder in eingeschränktem Rahmen einen Supercomputer simulieren.

Tritt der Raspberry Pi in die Welt der Supercomputer ein, muss er sich auch mit ihren Spielregeln messen lassen. Die vorliegende Arbeit untersucht das Skalierungsverhalten eines Raspberry Pi-Clusters unter der Workload von Linpack, Whetstone und STREAM. Sie zeigt auf, ob und in welcher Form sich die ausgewählten HPC-Benchmarks auf dem Cluster auführen lassen und evaluiert die Ergebnisse. Ein Schwerpunkt liegt dabei auf dem Energieverbrauch des Clusters bei unterschiedlichen Versuchsaufbauten.

# Inhaltsverzeichnis

# 1 Einleitung

Seit Beginn seiner Entwicklung im Jahr 2009 boomt der Minicomputer Raspberry Pi (im Folgenden als *RPi* bezeichnet). Er erhielt z.B. den Designpreis INDEX: award 2013 (vgl. `http://designtoimprovelife.dk/category/s15-award2013`), wurde als Innovation des Jahres bei den T3 Gadget Awards 2012 ausgezeichnet (vgl. `http://www.t3.com/news/t3-gadget-awards-2012-award-winners`) und zum Gadget of the Year 2012 des Linux Journal gewählt (vgl. `http://www.linuxjournal.com/slideshow/readers-choice-2012`). Im Februar dieses Jahres war das Modell B über 2,5 Millionen Mal verkauft worden.

## 1.1 Hintergrund

Die Leistung von Rechnern, seien es Großrechner, Desktop-Rechner oder Minicomputer, wird häufig durch Benchmarks ermittelt. Das ermöglicht die Vergleichbarkeit der Testergebnisse unterschiedlicher Systeme. Bekannte und erprobte Benchmark-Suites sind z.B. Linpack und Whetstone, mit denen seit den 70er Jahren die Performance von Supercomputern ermittelt wird. Für Einzelrechner mit Linux-Systemen gibt es z.B. die Phoronix Test Suite oder UnixBench, um die Performance der einzlnen Komponenten wie CPU, GPU und RAM zu evaluieren.

## 1.2 Fragestellung

Im wissenschaftlichen Umfeld werden immer häufiger mehrere RPis zu einem Beowulf-Cluster verschaltet, um parallele Berechnungen auszuführen. Vor diesem Hintergrund stellt sich die Frage: Wie verhält sich ein RPi bei der Ausführung von HPC-Benchmarks? Noch interessanter ist das Verhalten eines RPi-Clusters: Welche CPU-Performance lässt sich damit erzielen und wie verhält sich der Cluster bei Hinzunahme von Ressourcen, d.h. RPi-Rechenkernen? Im Zentrum dieser Arbeit stehen daher CPU-Performance und Skalierungsverhalten eines RPi-Clusters unter den Testbedingungen ausgewählter HPC-Benchmarks. Dazu wird zunächst die die Performance eines RPi-Einzelrechners ermittelt. Anschließend wird die Performance eines RPi-Clusters bei Ausführung der Benchmarks evaluiert. Im Fokus steht dabei der Energievebrauch, der mit Hilfe eines Strommessgeräts für die unterschiedlichen Versuchsaufbauten ermittelt wird.

## 1.3 Vorgehensweise

Grundsätzlich stellt sich die Frage, welche Benchmarks sich für das zu untersuchende System (RPi-Einzelrechner bzw. RPi-Cluster) und die zu untersuchende Komponente (CPU) eignen. Anschließend müssen geeignete Implementierungen der Benchmarks gefunden bzw. ausgewählt werden.

Für den physischen Versuchsaufbau muss sicher gestellt sein, dass alle erforderlichen Komponenten (RPi-Cluster, RPi-Einzelrechner und Strommessgerät) möglichst zuverlässig arbeiten. Dies muss regelmäßig überprüft werden, z.B. durch Kontrolle des Netzwerkstatus der Komponenten.

Die Ausführung der Benchmarks mit unterschiedlichen Anzahlen aktiver und angeschalteter RPi-Nodes erfolgt sinnvollerweise automatisiert durch entsprechende Shellskripte. Die Resultate müssen in geeigneter Form in eine Datenbank geloggt werden, was ebenfalls durch Shellskripte realisiert wird.

Der Stromverbrauch des RPi-Clusters mit unterschiedlichen Anzahlen aktiver und angeschalteter RPi-Nodes wird durch ein Strommessgerät ermittelt, das an die Stromversorgung des RPi-Clusters angeschlossen wird. Seine Messwerte müssen ebenfalls in geeigneter Form in eine Datenbank geloggt werden.

Die Messwerte der Benchmarks müssen mit denen des Strommessgeräts abgeglichen werden, um Aussagen über den Stromverbrauch treffen zu können. Für die Interpretation der ermittelten Versuchsergebnisse ist eine grafische Aufbereitung erforderlich.

## 1.4 Struktur

Um den Bezugsrahmen zu verdeutlichen, werden in Kapitel 2 zunächst grundlegende Definitionen geklärt. Insbesondere werden die ausgewählten Benchmarks vorgestellt (vgl. Kap. 2.4) und die Spezifikationen von RPi (vgl. Kap. 2.5) und RPi-Cluster erläutert (vgl. Kap. 2.6). Versuchsaufbau und -ablauf werden mit Blick auf Auswahl und Anpassung der RPi-spezifischen Parameter im folgenden Kapitel dargestellt (vgl. Kap. 3). Schließlich werden die Messergebnisse dargestellt (vgl. Kap. 3.3) und interpretiert (vgl. Kap. 4). Den Abschluss bilden eine Zusammenfassung der Untersuchungsergebnisse und ein Ausblick (vgl. Kap. 5).

# 2 Grundlagen und Begriffsbildung

Es gibt zahlreiche Benchmarks, die bereits an den RPi angepasst und auf diesem ausgeführt wurden. Dabei steht oft die Performance verschiedener Betriebssysteme (z.B. Fedora vs. Debian) oder einzelner Hardware-Komponenten im Vordergrund. Zu diesem Zweck werden hauptsächlich Linux-spezifische Benchmarks verwendet wie Sysbench CPU Benchmark (CPU), PyBench (Python-Implementierung), Apache Benchmark (Webserver), OpenSSL (CPU) oder ioquake3 (GPU).

Bei näherer Betrachtung erscheint es schwierig, sich einen Überblick über die existierenden Benchmarks zu verschaffen. Vieles, was von den Anwendern als „Benchmark“ bezeichnet wird, stellt sich als selbst geschriebene Routine heraus, mit der z.B. die Performance der Grafikkarte getestet werden soll. Auf solche Routinen darf sich die Untersuchung nicht stützen. Im Folgenden wird daher auf grundlegende Begriffe der Untersuchung eingegangen. Anschließend werden die verwendeten Benchmarks (vgl. 2.4) und ihre Ausführung auf dem RPi erläutert (vgl. 3.2).

## 2.1 Definition: Benchmarking

Unter *Benchmarking* oder „Maßstäbe vergleichen“ versteht man im Allgemeinen die vergleichende Analyse von Ergebnissen oder Prozessen mit einem festgelegten Bezugswert oder Vergleichsprozess. *Computer-Benchmarks*, die hier von Bedeutung sind, dienen dem Vergleich der Rechenleistung von Computer-Systemen, wozu in der Regel Software verwendet wird:

> A simple method of measuring performance is by means of a benchmark program. [...] The intention is that by running it upon a new type of machine one may learn something of the performance the machine would have if it ran the original programs [CW76].

Das *Computer Lexikon 2012* kennt folgende Definition:

> Mit einem Benchmark-Programm werden Hardwarekomponenten meist auf Geschwindigkeit getestet, wie z.B. die CPU, das Mainboard, die Festplatte (Schreib-Lese-Geschwindigkeit), die Grafikkarte (Frames/s) usw. Verschiedene Benchmark-Programme liefern oft unterschiedliche Ergebnisse, so dass ein direkter Vergleich zwischen den erreichten Werten kaum aussagekräftig ist [Pre11].

Hieran wird deutlich, dass die Aussagekraft von Benchmarks eng mit der jeweiligen Testumgebung und der Zielsetzung des Benchmarks zusammenhängt. Zwei Benchmarks, die die CPU-Performance evaluieren, liefern möglicherweise unterschiedliche Ergebnisse, weil unterschiedliche Parameter oder sogar Messgrößen zu Grunde liegen. Das ist insbesondere bei der

Auswahl der Implementierungen und Gestaltung der Testumgebung zu berücksichtigen (vgl. Kap. 2.5, 2.6 und 3.2).

In dieser Arbeit soll die Leistung von Hardware-Komponenten eines oder mehrerer parallel arbeitender Rechenkerne mit standardisierten Verfahren ermittelt werden. Rechenberg bezeichnet „*Analyse*, *Auswahl* und *Konfiguration* von Gesamtsystemen aus Hardware und Software [Rec06]“ als eine Hauptaufgabe der Leistungsbewertung:

> [F]ür diese Aufgaben, die die etwa im Zuge einer Rechnerbeschaffung anfallen, wurde in Form von standardisierten Meßprogrammen und -methoden (*benchmarking*) eine solide Basis geschaffen [Rec06].

Daher wird hier hier mit *Benchmarking* das ***Standardisieren von Arbeit*** bezeichnet.

## 2.2 Definition: Performance

Die physikalische Größe *Leistung* ist als *Energie pro Zeit* definiert. In der Informatik wird der Begriff meist abweichend als Rechenleistung verwendet, d.h. das Zeitverhalten von Programmen und Geräten oder die Leistungsfähigkeit eines Computersystems. Eine Abgrenzung erscheint jedoch schwierig:

> The performance of a computer is a complicated issue, a function of many interrelated quantities. These quantities include the application, the algorithm, the size of a problem, the high-level language, the implementation, the human level of effort used to optimize the program, the compiler's ability to optimize, the age of the compiler, the operating system, the architecture of the computer and the hardware characteristics [DLP03].

Das *Informatik-Handbuch* liefert folgende Aspekte:

> Quantitative Leistungsanalysen (*performance analyses*) ermitteln Leistungskenngrößen von Rechenanlagen. [...] Leistungsbewertung kann sich auf Teilschaltungen, Komponenten (wie Prozessor, Speichersystem oder periphere Geräte), gesamte Rechnerverbünde beziehen [Rec06].

In dieser Arbeit soll die Performance eines RPis und eines RPi-Clusters bei der Ausführung von Benchmark-Programmen ermittelt werden. Curnow erläutert in Bezug auf Whetstone:

> We are not claiming [to reflect] the overall performance of a given system. On the contrary, we believe that no single number ever can. It does, however, reflect the performance of a dedicated machine for solving a dense system of linear equations [CW76].

Mit *Performance* wird daher hier die ***Rechenleistung*** in diesem abgegrenzten Sinn bezeichnet, d.h. die Datenverarbeitungsgeschwindigkeit eines spezifischen Systems, nicht dessen Leistung im physikalischen Sinn.

## 2.3 Definition: Energieverbrauch

Obwohl der Begriff *Energieverbrauch* (engl. *energy consumption* oder *power consumption*) weit verbreitet ist (vgl. z.B. [FH12] und [BL08]), stimmt er nicht mit der physikalischen Definition überein: Nach dem Energieerhaltungssatz bleibt die Gesamtenergie in einem geschlossenen System konstant. Dennoch wird der Begriff auch in der Informatik häufig verwendet, um die Leistungsfähigkeit eines Rechners oder Rechensystems in Abhängigkeit von der Energieaufnahme zu charakterisieren:

> Vom physikalischen Standpunkt aus betrachtet, ist der Ausdruck „Energieverbrauch“ falsch. Energie kann nicht verbraucht oder erzeugt, sondern nur umgewandelt werden. Trotzdem hat sich dieser Bergriff eingebürgert [...]. Bei elektronischen Systemen wird elektrische Energie verwendet, um Rechenleistung in einer bestimmten Zeitspanne zu generieren, physikalisch gesehen wird die elektrische Energie jedoch zu nahezu hundert Prozent in Wärmeenergie umgewandelt [LB12].

In dieser Arbeit liegt ein Schwerpunkt der Leistungsbewertung auf der Energieaufnahme eines RPi-Clusters bei der Ausführung von Benchmark-Programmen. Mit *Energieverbrauch* ist hier der Exergieverbrauch bzw. die Entropieerzeugung in Watt im physikalischen Sinn gemeint.

## 2.4 Benchmarks

Aus der Fülle an existierenden Benchmarks eine Auswahl zu treffen erweist sich als trickreich. Mögliche Kriterien für die Auswahl nennt Weicker:

> The [...] best benchmark (1) is written in a high-level language, making it portable across different machines, (2) is representative for some kind of programming style (for example, systems programming, numerical programming, or commercial programming), (3) can be measured easily, and (4) has wide distribution [Wei90].

Der Benchmark muss zudem überhaupt auf das gewählte System anwendbar sein, d.h. das System muss dessen Spezifikationen erfüllen bzw. es muss eine geeignete Implementierung vorliegen. Aus diesem Grund musste z.B. SHOC ausscheiden. SHOC ist nicht lauffähig auf dem RPi, da Open CL nicht unterstützt wird.

Für die Untersuchung wurden drei etablierte HPC-Benchmarks vorgegeben, die sowohl auf Cluster-Architekturen als auch Einzelrechnern zur Anwendung kommen: Linpack, Whetstone und STREAM. Sie werden im Folgenden vorgestellt.

### 2.4.1 Linpack

Grundsätzlich muss zwischen der Linpack-Bibliothek und dem Linpack-Benchmark unterschieden werden. Erstere ist eine numerische Programmbibliothek zum Lösen von linearen Gleichungssystemen und galt darin lange Zeit als Standard. Der Linpack-Benchmark basiert auf zwei ihrer Routinen:

> Linpack was designed out of a real, purposeful program that is now used as a benchmark [Wei90].

Er wurde hauptsächlich von Jack Dongarra entwickelt und wird seit den 1970er Jahren zur Klassifizierung von Rechnern verwendet:

> Over the years additional performance data was added [...] and today the collection includes over 1300 different computer systems [DLP03].

Seit 1993 werden die leistungsfähigsten Supercomputer der Welt durch die Top500-Rankings ermittelt. Hierzu dient die Variante HPLinpack, auch $N \times N$ Linpack oder High Parallel Computing genannt:

> Over recent years, the LINPACK Benchmark has evolved from a simple listing for one matrix problem to an expanded benchmark describing the performance at three levels of problem size on several hundred computers. The benchmark today is used by scientists worldwide to evaluate computer performance, particularly for innovative advanced-architecture machines [DLP03].

Dort findet sich auch eine vertiefte Darstellung des Linpack-Benchmarks. Der Quellcode findet sich unter `http://www.netlib.org/benchmark/1000d`.

#### Funktionsweise

Linpack dient der Ermittlung der CPU-Leistung. Dazu werden Fließpunkt-Operationen auf einer Matrix durchgeführt, die intern in eine lineare Darstellung umgewandelt wird. Das Ergebnis wird als *Performance* in FLOPs („Floating Point Operations Per second“) ermittelt. Der Anteil der Nicht-Fließpunkt-Operationen wie Berechnungen auf Integer-Werten werden bei der Auswertung entweder vernachlässigt oder in die Fließpunkt-Operationen integriert (vgl. [Wei90]). Die Linpack-Varianten Linpack 100, Linpack 1000 und HPLinpack verwenden Matrizen der Größe $100 \times 100$, $1000 \times 1000$ bzw. $n \times n$, d.h. eine Matrix variabler Größe, zur Berechnung. HPLinpack liefert Messergebnisse für $R_{max}$ („Performance for the largest problem“), $N_{max}$ (“Size of the largest problem run“), $N_{1/2}$ (“Size where half the $R_{max}$ execution rate is achieved“) und $R_{peak}$ („Theoretical peak performance“) in *FLOPs*. Entscheidend für die Positionierung in der Top500-Liste ist dabei der $R_{max}$-Wert. Dabei erreicht der derzeit leistungsfähigste Supercomputer, *Tianhe-2 (MilkyWay-2)*, 33862.7 TFLOPs. *SuperMUC*, der zuletzt auf Platz 10 der Top500 gerankt wurde, erzielte 2897.0 TFLOPs.

Beim Vergleich der Ergebnisse von Linpack ist die Matrixgröße von Bedeutung, da ihre Abänderung bei unterschiedlichen Rechnerarchitekturen zu geringerer Datenlokalität und damit zu starken Abweichungen der Ergebnisse führen kann (vgl. [Wei90]). In der Regel werden daher die oben genannten Matrixgrößen verwendet, obwohl der Quellcode eine Änderung zulässt.

#### Linpack auf dem Raspberry Pi

Bereits kurz nach Verkaufsstart des RPi wurden Implementierungen von Linpack für den RPi bereit gestellt und Ergebnisse von Testläufen im Internet veröffentlicht. Somit liegen bereits Vergleichswerte für einen RPi-Einzelrechner vor. Auch Implementierungen von Linpack für

verteilte Systeme unabhängig von den Top500-Rankings sind im Umlauf (vgl. `http://www.netlib.org/benchmark/hpl`).

### 2.4.2 Whetstone

Auch Whetstone ist als Benchmark im HPC-Bereich und zur Klassifizierung von Einzelrechnern seit vielen Jahren etabliert:

> [T]he most common 'stone age' benchmarks (CPU/memory/compiler benchmarks only) [are] in particular the Whetstone, Dhrystone, and Linpack benchmarks. These are the benchmarks whose results are most often cited in manufacturers' publications and in the trade press [Wei90].

Er wurde 1976 von Roy Longbottom u.A. entwickelt und gilt als erstes Programm, das jemals explizit für das Benchmarking industrieller Standards designt wurde (vgl. [Wei90]). Wie Linpack misst er die CPU-Leistung.

#### Funktionsweise

Die Funktionsweise von Whetstone ähnelt Linpack. Allerdings werden nicht nur Fließkomma-Berechnungen ausgeführt. Auch mathematische Funktionen, Integer-Arithmetik, bedingte Anweisungen etc. kommen zur Anwendung, da die ursprüngliche Programmversion nicht komplex genug war, um ein durchschnittliches FORTRAN-Programm zu simulieren. Gegenüber der ursprünglichen Implementierung in ALGOL 60 war ein damals gängiger FORTRAN-Compiler in der Lage, Zwischenergebnisse in schnellen Registern abzuspeichern und darauf zurückzugreifen. So fiel die gemessene CPU-Leistung deutlich höher aus als erwartet (vgl. [CW76]).

Die einzelnen Teile werden als Module bezeichnet und sind jeweils in eine For-Schleife eingebettet, die viele Male hintereinander ausgeführt wird. Ein Rahmenprogramm steuert Aufruf der Module und Ausgabe der Ergebnisse. Auch die Ausgabe ist Teil des Benchmarks, allerdings bemerkt Curnow hierzu:

> This output is only required to ensure that the calculations are logically necessary; it is not intended to represent the output from a typical program [CW76].

Die Ergebnisse wurden zunächst in *Whetstone Instructions per Second* gemessen, heutige Implementierungen liefern Ergebnisse in MFLOPs. Eine detaillierte Darstellung der Entwicklung und Implementierung von Whetstone findet sich bei [CW76], ebenso der ursprüngliche Quellcode in ALGOL 60.

#### Whetstone auf dem Raspberry Pi

Whetstone erschien wie für den RPi gemacht, da er sich als Standard für Mini-Computer etabliert hat: Der Benchmark war in den 70er Jahren für Maschinen mit deutlich geringerer Rechenleistung als heute entwickelt worden. Bereits damals gingen die Entwickler von notwendigen Anpassungen für neue Speicherhierarchien aus:

> When more is known about the characteristics of programs running on these multi-level store machines it may be possible to produce a typical program for particular types of machine. [...] Despite these limitations the program described should be of some value, particularly in relation to smaller machines [CW76].

Er wurde vom Entwickler selbst für den RPi angepasst und und auf diesem getestet (vgl. `http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm.`. Auch hier liegen also Vergleichswerte für einen RPi-Einzelrechner vor.

Wie für die meisten etablierten HPC-Benchmarks existieren Implementierungen in höheren Programmiersprachen wie C, C++, Fortran und Java (vgl. `http://freespace.virgin.net/roy.longbottom`). Ob eine lauffähige Implementierung für die Cluster-Architektur existiert, wird sich zeigen müssen.

### 2.4.3 STREAM

Mit zunehmender Rechenleistung der Prozessoren und der Zunahme an Prozessorkernen innerhalb eines Rechnersystems zeigte sich, dass es nicht ausreicht, die Leistungsfähigkeit lediglich an Hand der CPU-Leistung zu bewerten. Zudem lässt sich eine Tendenz beobachten, dass Rechnersysteme gezielt für die Ausführung bestimmter Benchmarks wie HPLinpack optimiert werden, um in Ranglisten bessere Plätze zu erreichen.

STREAM wurde mit dem Ziel entwickelt, das Verhältnis von CPU-Leistung zu Speichergeschwindigkeit in die Bewertung einzubeziehen. Mittlerweile ist STREAM auch Bestandteil der HPCChallenge Benchmark-Suite. Sie wurde mit der Zielsetzung entwickelt, Anforderungen aus der Anwendungspraxis in die Bewertung von HPC-Rechnersystemen einfließen zu lassen (vgl. [LDK+05]).

#### Funktionsweise

STREAM ist ein synthetischer Benchmark und basiert auf einem Programm aus der wissenschaftlichen Praxis von John McCalpin. Es ermittelt die dauerhafte Speicher-Bandbreite (engl. „sustainable memory bandwidth“ [McC95]) für angrenzende Speicherzugriffe langer Vektoren (engl. „sustainable memory bandwidth for contiguous, long-vector memory accesses“[McC95]). Die Vektoren müssen lang genug sein, dass sie nicht im Prozessor-Cache vorgehalten, sondern aus dem nicht flüchtigen Speicher geladen werden. STREAM ist somit Teil eines neuen Modells, das als Performance-Maßzahl die Gesamtzeit $T_{total} = T_{cpu}$ (CPU-Performance) + $T_{memory}$ (Speicher-Performance, ermittelt durch STREAM) festlegt (vgl. [McC95] und [McC05]).

STREAM durchläuft bei seiner Ausführung vier Module (Copy, Scale, Add und Triad) und liefert für jedes Modul die durchschnittliche, maximale und minimale Ausführungszeit sowie die Ausführungsrate. Jedes Modul wird mehrfach ausgeführt, um verlässliche Mittelwerte zu erhalten. Zwischen dem Programmcode der Module bestehen Abhängigkeiten, um starke Optimierungen durch den Compiler zu verhindern (vgl. [McC05]). Quellcode und Ergebnisse für STREAM auf verschiedenen Systemen finden sich unter `http://www.cs.virginia.edu/stream`.

#### STREAM auf dem Raspberry Pi

Auch für STREAM wurden bereits Ergebnisse auf einem RPi-Einzelrechner bereits veröffentlicht (vgl. `http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html`). Die auf dem RPi-Cluster verwendete Implementierung findet sich unter `http://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.f`.

## 2.5 Spezifikation des RPi Modell B

Was dem Nutzer zuerst auffällt, ist sicherlich die Größe des RPi. Er wird manchmal als „Scheckkarten-Computer“ bezeichnet und überschreitet diese Maße mit 85.60 mm × 53.98 mm × 17 mm tatsächlich nur geringfügig. Auf dieser Platine sind alle Komponenten verbaut, die den RPi zu einem voll funktionsfähigen Rechner machen.

### 2.5.1 Physischer Aufbau

Der RPi ist mit einem Broadcom BCM2835 als System on a Chip ausgestattet. Es enthält CPU (ein ARM1176JZFS-Prozessor) und GPU (ein Broadcom VideoCore IV-Koprozessor, vgl. [Lan12]). Auffällig ist, dass die CPU des RPi (ein ARM-Chip, wie er auch in Mobiltelefonen zur Anwendung kommt), relativ schwach ist im Vergleich zur GPU. Diese unterstützt Full HD-Auflösung und das Hardware-beschleunigte Rendern verschiedener Videoformate. Das Modell B verfügt über 512 MB SDRAM. Er kann nicht erweitert werden, doch die Aufteilung zwischen CPU und GPU ist variabel. Da der RPi kein BIOS hat, muss dazu die Datei `/boot/start.elf` manipuliert werden (vgl. [Pow12]).

Der RPi verfügt über einen HDMI-Ausgang, einen Cinch-Ausgang („RCA Jack“) und einen analogen Tonausgang. Er hat zwei USB 2.0-Schnittstellen und eine RJ45-Buchse zum Anschluss einer 10/100 Base-T Ethernet-Schnittstelle. Ein Steckplatz ist für eine SD-Karte vorgesehen, auf der der nicht flüchtige Speicher liegt. Die Stromversorgung erfolgt über einen Mini-USB-Eingang. Zum Betrieb werden 700 mA/5 V benötigt (vgl. [Pow12]). Der Vollständigkeit halber sei das Vorhandensein einer Allzweckeingabe/-ausgabe mit sechs Anschlüssen und jeweils 26 Pins erwähnt. Darüber können, ähnlich wie bei einem Arduino-Board, z.B. LEDs, Sensoren und Displays angesteuert werden.

### 2.5.2 Betriebssystem

Für den RPi existieren mehrere Varianten bzw. Derivate verbreiteter Betriebssysteme. Am verbreitetsten sind Linux/Unix-basierte Systeme wie FreeBSD und NetBSD (BSD-Varianten), Raspbian (Debian-Variante), Pidora (Fedora-Variante) oder eine Variante von Arch Linux (vgl. [Pow12]). Daneben gibt es Implementierungen von RISC OS und Plan 9. Als Betriebssytem für die RPi-Einzelrechner war Raspbian gewählt worden, die offizielle Distribution der Raspberry Pi Foundation (vgl. `http://www.raspberrypi.org/faqs`). Das Abbild findet sich unter `http://downloads.raspberrypi.org/raspbian_latest`.

## 2.6 Raspberry Pi-Cluster als Testumfeld

Seit einiger Zeit lässt sich die Tendenz beobachten, eine größere Anzahl von Raspberry Pis zu einem Cluster zu koppeln (weitere Projekte werden in [CCB+13], [Kie13], [Bal12] und

[Ou13] dargestellt). Die hier verwendete Architektur wird im Folgenden kurz vorgestellt. Eine detaillierte Darstellung liefert [Kli13].

### 2.6.1 Physischer Aufbau

Der RPi-Cluster besteht aus 20 RPi Modell B-Einzelrechnern ((1), DNS-Namen `pi01` – `pi20`). Sie sind jeweils über ein Ethernet-Kabel (2) mit einem 24 Port Gigabit-Switch (3) verbunden, der an einen zentralen x86-Server (DNS-Name `careme`) angeschlossen ist.

Er besteht aus einem Mini-ITX-Mainboard (4) und Festplatten mit einem Software-RAID Level 5-System für den Netzwerkzugriff und einem RAID Level 1-System für das Betriebssystem (5).

Die Stromversorgung erfolgt über ein zentrales Netzteil (6). Daran sind zwei Verteiler (7) angeschlossen, an die jeweils 10 RPi-Einzelrechner über Mini-USB-Kabel (8) verbunden sind.

Alle Komponenten befinden sich in einem Metallgehäuse, in dessen Mitte drei Kühlgebläse angebracht sind. Ein solches System relativ kostengünstiger Rechner mit einem BSD- oder

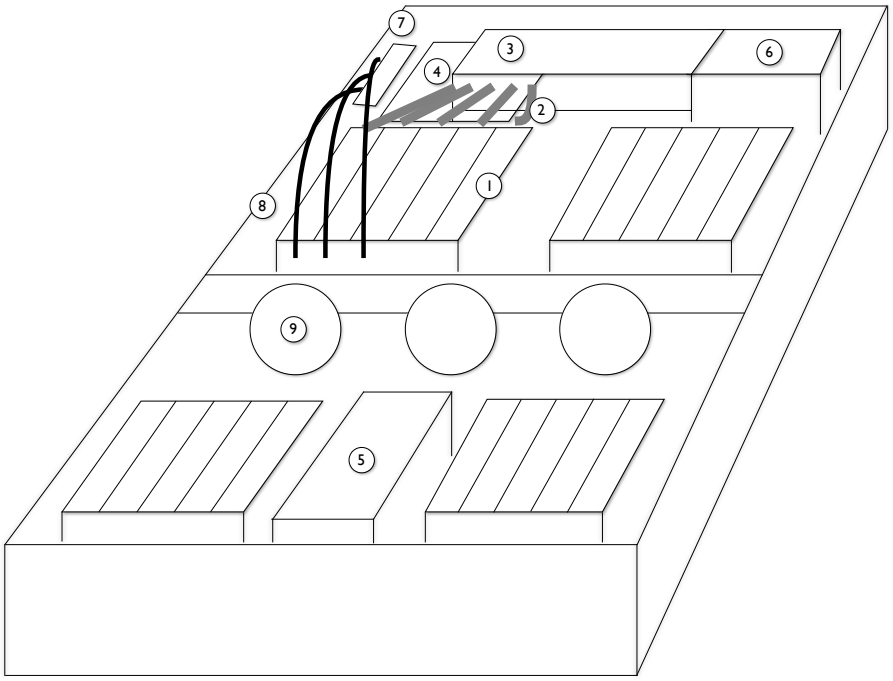


Abbildung 2.1: Schematischer Aufbau des hier verwendeten Bramble.

Linux-Betriebssystem, die über IP kommunizieren, wird im Allgemeinen als *Beowulf* bezeichnet (vgl. [Kie13]). Für einen Beowulf-Cluster aus RPis ist der Begriff *Bramble* gebräuchlich, der im Folgenden verwendet wird (vgl. `www.raspberrypi.org/archives/tag/bramble`).

### 2.6.2 Betriebssystem und Dateisystem

Auf den RPi-Knoten ist das oben genannte Betriebssytem Raspbian installiert (vgl. 2.5.2), auf `careme` eine Standard-Debian-Version. Ein Charakteristikum eines Beowulf-Clusters ist,

dass es kein Shared Memory-Interface und keine Cache-Kohärenz gibt. Somit stellt sich die Frage nach der Zeit- und Datensynchronisation der RPi-Knoten und des Servers sowie den Möglichkeiten des gemeinsamen Speicherzugriffs.

Die Datei-Synchronisation des Servers und der RPi-Knoten erfolgt über ein Netz-Dateisystem (hier: `nfs`). Dessen Verwaltung erfolgt auf dem Server über ein Union-Dateisystem (hier: `aufs`), das mehrere Schichten des Dateisystems anlegt: Die unteren Schichten haben nur Leseberechtigungen, die oberste Lese-Schreib-Berechtigungen. Werden Daten in einem geteilten Verzeichnis verändert, so bleiben die unteren Schichten unverändert. Die Änderungen werden nur auf der obersten Schicht geschrieben. Für gemeinsam genutzte Daten des Servers und der Clients steht auf dem Server das Verzeichnis `/srv/nfs-share` bzw. auf den Clients `/srv` zur Verfügung. Eine genaue Darstellung findet sich bei [Kli13].

### 2.6.3 Implikationen für den Versuchsaufbau

Welche Zugriffsmöglichkeiten auf die Komponenten des Bramble gibt es, und wie können verteilte Berechnungen darauf ausgeführt werden?

#### Netzzugriff

Der Zugriff auf den Bramble erfolgt aus dem internen Netz über IP und SSH. Auf die einzelnen RPi-Knoten wird von `careme` aus ebenfalls über IP und SSH zugegriffen (private Adressen `10.0.0.2` bis `10.0.0.21`).

Es gibt auf jedem RPi-Knoten den Raspbian-Standard-User `pi`, der nicht verändert wurde. Aus Sicherheitsgründen wurden für `careme` keine root-Rechte erteilt, nur für die einzelnen RPi-Knoten. Stattdessen wurde der Benutzer `rpi-user` eingerichtet. Das bedeutet, dass alle Änderungen im geteilten Verzeichnis `/srv` bzw. `/nfs-share/srv` von einem der RPi-Nodes aus vorgenommen werden müssen, da `rpi-user` hierauf keine Schreibberechtigungen hat. Somit wurde vorab RPi-Node `pi03` als Berechnungsknoten definiert, von dem aus alle Skripte ausgeführt, Programme installiert werden etc.. Die Datenbank zur Speicherung von Konfigurationen und Messwerten war hingegen auf dem Server angelegt worden. Dadurch wird häufiges Navigieren zwischen dem Server und dem Berechnungsknoten mit unterschiedlichen Benutzern notwendig.

Die Skripte zur Ausführung der Benchmarks sehen häufig das Herunterfahren einzelner RPi-Knoten vor, um den Energieverbrauch beim Abschalten nicht aktiver RPi-Knoten zu ermitteln. Alle Anwendungen wurden daher auf maximal n=19 RPi-Nodes ausgeführt und von `pi03` aus gesteuert.

#### Ausführung verteilter Anwendungen

Der Versuchsaufbau sieht vor, verteilte Anwendungen auf unterschiedlichen Anzahlen von RPi-Knoten auszuführen. Zur verteilten Berechnung von Anwendungen auf mehreren CPUs steht auf dem Bramble die MPI-Implementierung MPICH in der Version 3.0.4 zur Verfügung. Um die CPUs bzw. RPi-Knoten und die Anzahl der darauf auszuführenden Prozesse zu spezifizieren, muss ein entsprechendes Machinefile erstellt und im gesharten Verzeichnis abgelegt werden. Dieses wird der Ausführung von MPICH mit `mpiexec` als Parameter übergeben (vgl. Kap. 3.2.2).

# 3 Versuchsaufbau und -ablauf

Das folgende Kapitel beschreibt Versuchsaufbau und -durchführung sowie Ziele der Messung, von der Erkenntnisse über das Skalierungsverhalten eines RPi-Clusters unter der Arbeitslast der ausgewählten HPC-Benchmarks erwartet werden.

## 3.1 Zielsetzung

Für das Skalierungsverhalten des Bramble unter der Workload von Linpack und STREAM werden wie in Kap. **??** beschrieben zwei Maßzahlen betrachtet: Performance und Energieverbrauch.

Die augewählten Benchmarks werden auf n–1 RPi-Knoten des Bramble mit n=19 RPi-Nodes ausgeführt (vgl. Kap. 2.6.3). Alle Messungen werden zweimal durchgeführt: Mit Stromanschluss der nicht beteiligten RPi-Nodes und ohne. Für beide Benchmarks werden zwei Ergebnisparameter betrachtet: Ausführungsrate in GFLOPs und Ausführungszeit in s für HPLinpack, Ausführungsrate in MB/s und durchschnittliche Ausführungszeit in s für STREAM. Die Ergebnisse der Messreihen werden anschließend gegenübergestellt (vgl. Kap. 3.3).

## 3.2 Aufbau und Art der Messung

Hier stellen sich zwei grundsätzliche Fragen: Welcher Art ist die Messung und welche Voraussetzungen sind dafür erforderlich?

Die Performance wird pro Benchmark durch zwei Ergebnisparameter ermittelt. Zwei Messreihen werden durchgeführt: Bramble mit n–1 RPi-Knoten aktiv/19 RPi-Knoten angeschaltet, Bramble mit n–1 RPi-Knoten aktiv/n RPi-Knoten angeschaltet. Für den Versuchsaufbau sind folgende Aspekte von Bedeutung: Mögliche Modifikationen der RPi-Knoten (Hardware, Betriebssystem-Konfiguration), Zeitsynchronisation der RPi-Knoten, Skalierung der Messung auf n–1 RPi-Knoten, automatisierte Durchführung der Messung auf 19–n RPi-Knoten sowie Einlesen der Messwerte in eine geeignete Datenstruktur.

### 3.2.1 Versuchsaufbau Ri-Einzelrechner

Viele Nutzer stellen sich nach Inbetriebnahme eines RPi-Einzelrechners die Frage nach Swap Space und Übertakten (vgl. [Pow12]). Beides liegt nahe, da das Modell B des RPi nur über 512 MB Arbeitsspeicher verfügt. Die CPU-Leistung ist mit 700 MHz ebenfalls eher niedrig.

Im Praxisbetrieb wurde gezeigt, dass ein Übertakten der CPU auf bis zu 1 GHz gefahrlos möglich ist. Bei den hier verwendeten RPi-Knoten wurde darauf verzichtet, da es wenig zielführend erscheint, die Komponente zu manipulieren, deren Performance man durch

Benchmarking evaluieren möchte[1] Allerdings gibt es bereits Ergebnisse für Linpack 100 und Whetstone bei Übertakten der CPU auf 1 MHz (vgl. `http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm`).

Zur Allokierung von Swap Space auf dem RPi gibt es grundsätzlich drei Möglichkeiten: Swap-Datei, Swap-Partition oder zRAM.

Das Betriebssystem Raspbian verwendet standardmäßig eine Swap-Datei `/var/swap` auf der SD-Karte. Hierbei zeigen sich Probleme: Erstens können ständige Schreibzugriffe auf Dauer die SD-Karte beschädigen. Zweitens sind Schreibzugriffe darauf sehr langsam, was die Performance des Systems bei hoher Arbeitsspeicherlast beeinträchtigen kann (vgl. [Pow12]).

Das gilt auch für die Allokierung einer auf Unix-Systemen üblicherweise genutzten Swap-Partition, weswegen diese Möglichkeit in der Praxis keine Rolle spielt.

Eine weitere Möglichkeit des Swapping ist die Verwendung von zRAM. Hierbei wird ein Teil des Arbeitsspeichers komprimiert und als Swap Space genutzt. Hierbei werden keine Zugriffe auf die SD-Karte notwendig (vgl. [Pow12]).

Die Swap-Datei kann mit dem Befehl `sudo update-rc.d dphys-swapfile remove` deaktiviert werden.

### 3.2.2 Versuchsaufbau Bramble

Das Komponentendiagramm 3.1 zeigt den Versuchsaufbau pro Messreihe auf dem Bramble, der als *ExperimentSuite* bezeichnet wird. Es enthält die systemkritischen Elemente des Versuchsaufbaus: Stromversorgung und Netzanschluss des Bramble als physische Komponenten sowie die MySQL-Datenbank `rpiWerte` als logische Komponente.

#### Modifikationen der RPi-Nodes

Betriebssystem und Filesystem des Bramble wurden so übernommen wie bei Beginn der Untersuchung zur Verfügung gestellt: Die CPUs der RPi-Nodes sind nicht übertaktet und der Bramble-Server `careme` nutzt keinen Swap Space. Bei den RPi-Nodes war allerdings ein großer Teil der SD-Karten als Swap Space allokiert worden[2]. Im Praxisbetrieb zeigten sich jedoch rasch Schwierigkeiten, die Anpassungen in OS-Konfiguration und Hardware erforderlich machten. Folgende Fehlerfälle waren am häufigsten:

1. **Defekte Hardware (Mini-USB-Kabel)**
   Einige Mini-USB-Kabel waren bereits zu Beginn der Untersuchung defekt oder hatten einen Wackelkontakt. Sie mussten durch funktionsfähige Kabel ersetzt werden.

2. **RPi-Node nicht erreichbar (`ping`)**
   Häufig reagierte ein RPi-Node nicht auf ein `ping` von einem RPi-Node oder von `careme` aus (Fehlermeldung `Destination Host Unreachable`), obwohl die Status-LEDs aktiv waren. Als einzige Lösung erwies sich Ziehen und Wiedereinstecken des Mini-USB-Kabels; war das nicht erfolgreich, musste das Vorgehen mit dem Netzwerkkabel wiederholt werden (das Netzwerkkabel allein reicht nicht aus). Danach war der Host i.d.R. wieder mit `ping` erreichbar.

[1] Für eine zukünftige Untersuchung wäre es interessant zu ermitteln, ob man den relativ hohen Stromverbrauch des Bramble bei Niedriglast (vgl. [Kli13]) durch Untertakten der einzelnen CPUs senken kann (vgl. 5).

[2] Vgl. [Kli13].

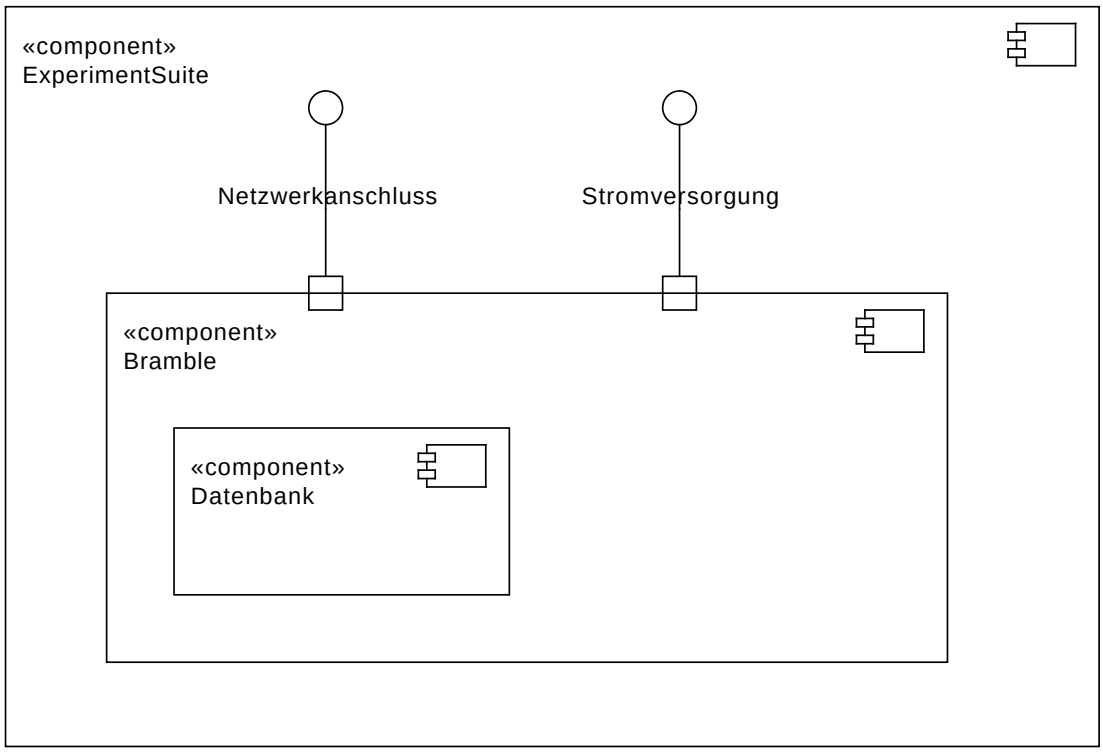


Abbildung 3.1: Komponentendiagramm des Versuchsaufbaus.

3. **RPi-Node nicht erreichbar (SSH)**
   Hier traten drei Fehlerfälle auf: Am häufigsten war die Fehlermeldung `No route to host` beim Versuch, von `careme` oder einem anderen RPi-Node aus eine SSH-Verbindung zum Zielhost herzustellen. Die einzige Lösung ist Ziehen und wieder Einstecken des Netzwerkkabels. Nach einigen Minuten ist der Host i.d.R. wieder erreichbar.

   Ein weiterer Fehlerfall war ein überraschender Passwortprompt für `root` beim Versuch, eine SSH-Verbindung zu einem RPi-Node zu einem anderen RPi-Zielnode aufzubauen. Dieses Problem ließsich durch Eintragen des RSA Public Key von `root` in die Datei `~/.ssh/authorized_keys` auf dem Zielhost lösen.

   Seltener trat die Fehlermeldung `WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!` auf. Dies konnte durch Korrektur des RSA Public Key des Anfragehosts in der Datei `~/.ssh/known_hosts` auf dem Zielhost gelöst werden.

4. **Geshartes Verzeichnis nicht gemountet**
   Beim Neustart eines RPi-Nodes oder des gesamten Clusters wurde häufig das gesharte Verzeichnis nicht gemountet, was zu Fehlermeldungen wie `-bash: /srv/libraries/etc/.sharedprofile: No such file or directory` führte und sich durch Mounten des Verzeichnisses mit `mount /srv` beheben ließ.

5. **bash-Befehle werden nicht erkannt**
   Aus unklaren Gründen wurden manchmal von einzelnen RPi-Nodes häufig verwendete

bash-Befehle nicht mehr erkannt, was sich an Fehlermeldungen wie `mpiexec: command not found` zeigt. Latenzen beim Zugriff auf den gemeinsamen Speicherbereich könnten der Grund hierfür sein, da sich ein Logout und erneuter Login auf dem betreffenden RPi-Node als einzige Lösung erwies.

### Zeitsynchronisation der RPi-Nodes

Der RPi-Einzelrechner besitzt aus Kostengründen keine Systemuhr (vgl. [Sch13]), sondern synchronisiert sich beim Booten gegen einen NTP-Server im Internet. Für die parallele Ausführung eines Programms auf mehreren Rechnerkernen ist die Zeitsynchronisation der RPi-Knoten und des Servers essentiell. Auf dem Bramble-Server gibt es daher einen OpenNTP-Server, gegen den sich die RPi-Knoten synchronisieren (vgl. [Kli13]).

### Skalierung der Messung auf 19–n RPi-Nodes

Das folgende Aktivitätsdiagramm zeigt, welche Schritte aus Benutzersicht für die Durchführung einer ExperimentSuite, d.h. der Ausführung eines Benchmarks auf einer gewählten Anzahl von aktiver und powered RPi-Nodes erforderlich sind. Es wird mit n=19 RPi-Nodes begonnen und einmal über alle Nodes von 20–1 (ohne den Ausführungsknoten `pi03`) iteriert. Danach wird die zweite Messung durchgeführt, wobei nach jedem Iterationsschritt der nicht mehr benötigte RPi-Node abgeschaltet wird. Jede Iteration der Benchmark-Ausführung läuft prinzipell gleich ab, während am Anfang und am Ende spezielle Vorkehrungen zu treffen sind.

### Automatisierte Durchführung der Messung auf 19–n RPi-Nodes

Die automatisierte Durchführung der Messung erfolgt durch Shellskripte. Sie werden im gesharten Verzeichnis abgelegt und können von `pi03` oder einem anderen Ausführungsknoten aus gestartet werden. Folgende Schritte werden durch die Skripte realisiert:

1. Erstellen eines Machinefile zur Verteilung der Workload auf n RPi-Nodes, ggf. Löschen des alten.
2. Mounten des gesharten Verzeichnisses auf allen RPis.
3. Navigation ins Arbeitsverzeichnis der Experimentsuite.
4. Iteration über n ausgewählte Benchmarks:
   a) Erstellen der Logfiles.
   b) Iteration über n RPi-Nodes:
      i. Einrichten der Datenbank.
      ii. Verteilte Ausführung des Benchmarks auf n RPi-Nodes.
      iii. Loggen der Ergebnisdaten.
   c) Iteration über n RPi-Nodes:
      i. Einrichten der Datenbank.
      ii. Verteilte Ausführung des Benchmarks auf n RPi-Nodes.

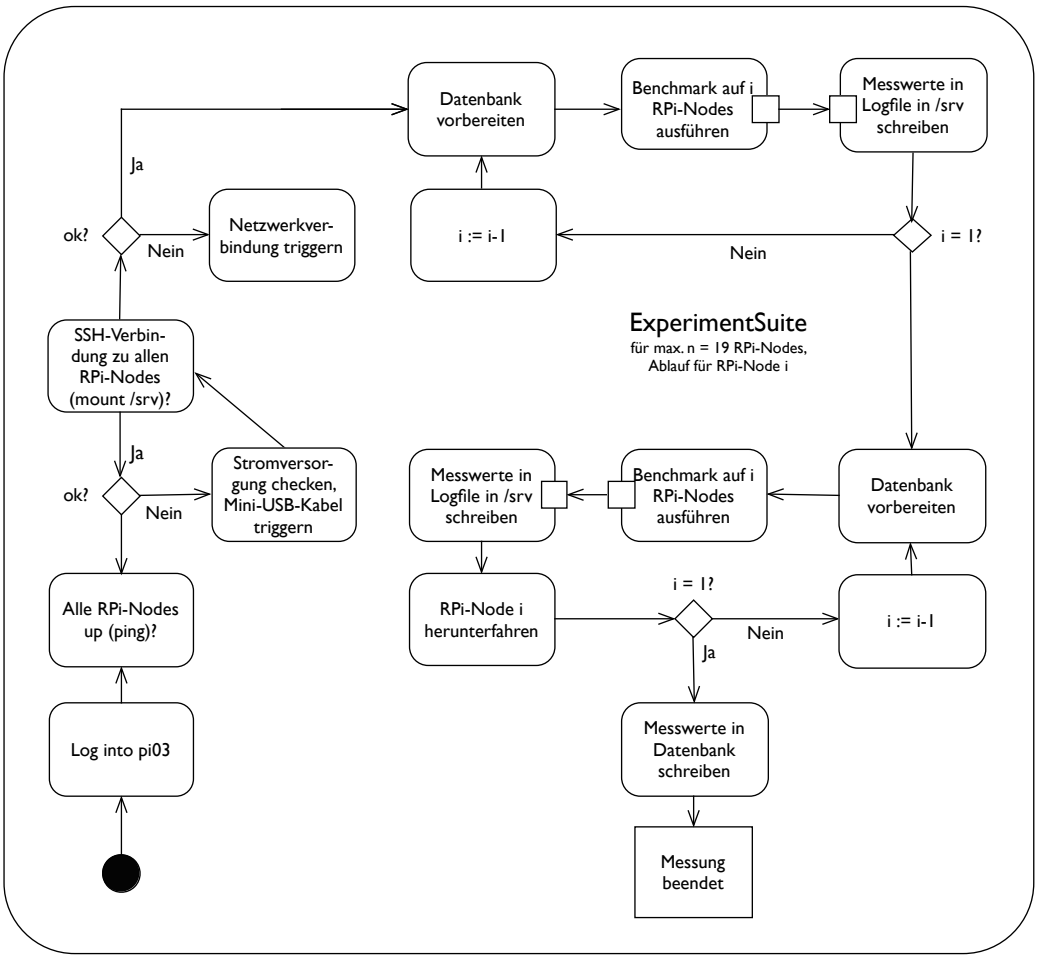


Abbildung 3.2: Aktivitätsdiagramm der ExperimentSuite.

iii. Shutdown von RPi-Node n.

iv. Loggen der Egebnisdaten.

d) Parsen der Logfiles für die Datenbank-Eingabe.

e) Schreiben der Messergebnisse in die Datenbank.

Dazu wurden drei Shellskripte `startBenchmarks.sh` (Schritte 1–3), `STREAM.sh` und `hpl-2.1.sh` (Schritt 4 für den jeweiligen Benchmark) erstellt. Der Quellcode findet sich in Kap. 6.1.

#### Einlesen der Messwerte in eine geeignete Datenstruktur

Schließlich stellt sich die Frage, wo und in welcher Form die Ergebnisdaten zweckmässigerweise abgelegt werden. Die Entscheidung fiel zu Gunsten einer MySQL-Datenbank auf `careme`. Sie orientiert sich an einem vorgegebenen Datenbankschema, das die Bramble-ExperimentSuites in einen größeren Versuchsaufbau integriert. Während der praktischen

Arbeit wurde das Schema schrittweise an die tatsächlichen Erfordernisse angepasst. Z.B. wurde für jedes ausgeführte Teilmodul eines Benchmarks ein Parameter definiert, der Unix-Timestamp seines Ausführungsendes ermittelt und zusammen mit dem jeweiligen Messwert in die Datenbank eingelesen.

Um dem gewählten Datenbankschema gerecht zu werden, mussten neben der Tabelle mit den Messergebnissen weitere Tabellen befüllt und über eine N2M-Tabelle verknüpft werden (vlg. 3.2, „Datenbank vorbereiten“):

1. Benennung und Beschreibung des Benchmarks
2. Benennung und Beschreibung des Versuchsaufbaus
3. Konfiguration des Versuchsaufbaus

Im ersten Schritt werden die statischen Konfigurationen für STREAM und HPLinpack festgelegt, was durch zwei weitere Shellskripte `loadGeneratorConfigHpl.sh` und `loadGeneratorConfigStream.sh` (vgl. Kap. 6.1) realisiert wurde. Die dynamischen Schritte pro ExperimentSuite (Benennung, Beschreibung und Konfiguration des Versuchsaufbaus, Einlesen der Messwerte und Verknüpfung von ExperimentSuite mit Benchmark-Konfiguration) wurden in die Ausführungsskripte der Benchmarks integriert.

## 3.3 Ergebnisse

Der nächste Abschnitt präsentiert die Untersuchungsergebnisse mit Schwerpunkt auf dem Bramble. Für die Durchführung standen auf Grund der oben beschriebenen Probleme nur 17 RPi-Nodes zuverlässig zur Verfügung. Zur besseren Lesbarkeit der Ergebnisse wurde entschieden, beide Benchmarks auf gleich vielen RPi-Nodes auszuführen. Da HPLinpack mindestens vier Prozessoren bzw. Prozesse benötigt, wurde die Anzahl der Prozessoren für STREAM hieran angepasst. Beide Benchmarks wurden demnach auf n=16 RPi-Nodes ausgeführt, `pi03` dient als Ausführungsknoten und wird in der Messung nicht berücksichtigt.

### 3.3.1 RPi-Einzelrechner: Linpack 100, Whetstone und STREAM

Bei der Ausführung von Linpack 100, Whetstone und STREAM in den ausgewählten Implementierungen erreichte der RPi-Einzelrechner folgende Ergebnisse:

1. **Linpack 100**
   - **Ausführungsrate:** 41.31 MFLOPS = 0.04131 GFLOPS
2. **Whetstone**
   - **Ausführungsrate:** 255.154 MWIPS
   - **Ausführungszeit:** 10.190 s
3. **STREAM**
   - **Ausführungsrate:**
     a) COPY: 274.4. MB/s
     b) SCALE: 209.3 MB/s

c) ADD: 287.2 MB/s

d) TRIAD: 271.1 MB/s

- **Ausführungszeit:**

a) COPY: 0.586838 s

b) SCALE: 0.766437 s

c) ADD: 0.838107 s

d) TRIAD: 0.886793 s

Detaillierte Ergebnisdateien finden sich im Anhang (vgl. Kap. 6.2).

### 3.3.2 Bramble: HPLinpack

Die folgenden Diagramme zeigen die Ergebnisse für HPLinpack auf dem Bramble mit zwei Ausgabeparametern: Time to Completion und CPU-Performance in MFLOPS, jeweils skaliert auf 16–4 RPi-Nodes. Diagramme 3.3 und 3.4 zeigen die Ergebnisse für n RPi-Nodes aktiv/16 RPi-Nodes powered (d.h. alle). Diagramme 3.5 und 3.6 zeigen das Verhalten bei n RPis aktiv/n RPis powered.

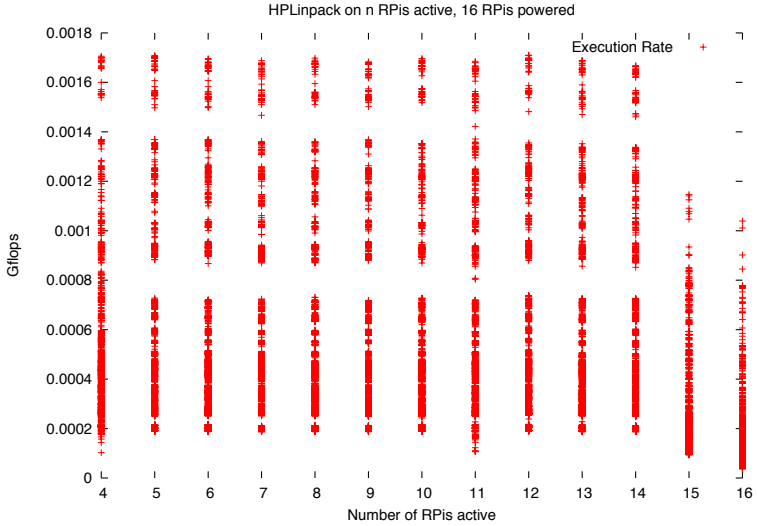


Abbildung 3.3: Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

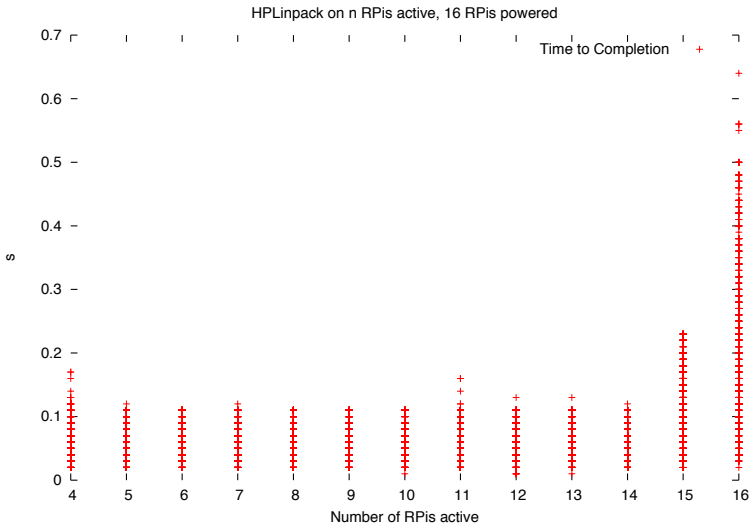


Abbildung 3.4: Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

### 3.3.3 Bramble: STREAM

Die folgenden Diagramme zeigen die Ergebnisse von STREAM auf dem Bramble für die Module Copy, Scale, Add und Triad, jeweils mit zwei Ausgabeparametern und skaliert auf 16–4 RPi-Nodes. Diagramme 3.7 und 3.8 zeigen die Ergebnisse für n RPi-Nodes aktiv/16 RPi-Nodes powered, Diagramme 3.9 und 3.10 für n RPi-Nodes aktiv/n RPi-Nodes powered.

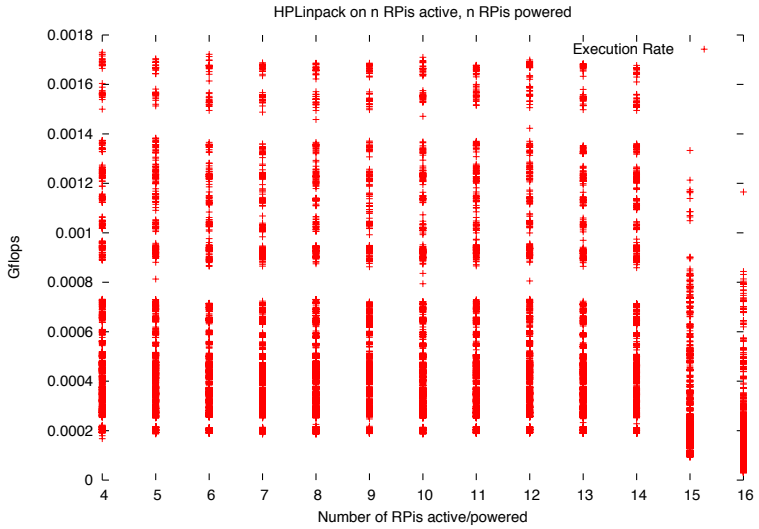


Abbildung 3.5: Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.

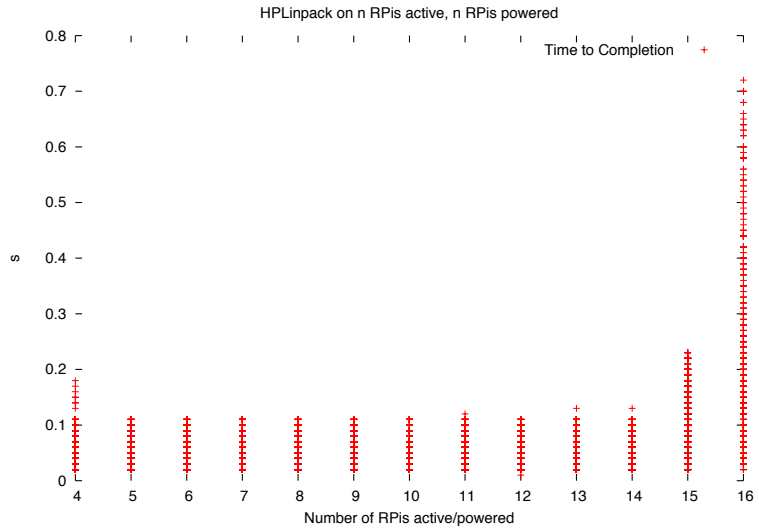


Abbildung 3.6: Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.

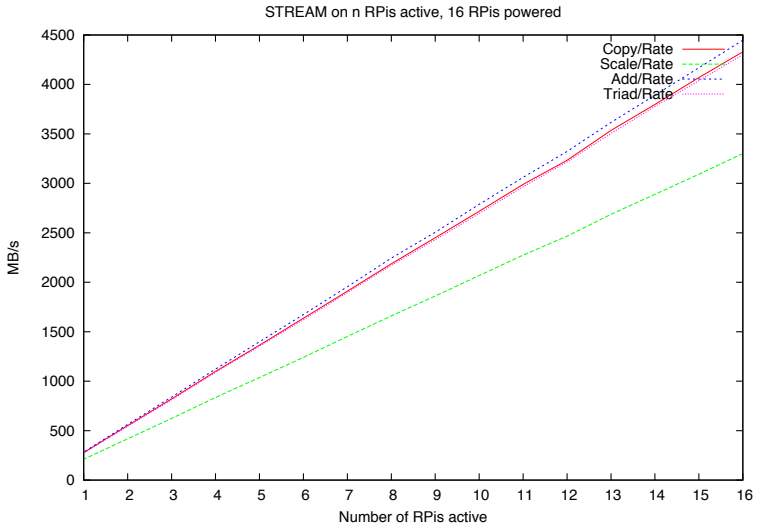


Abbildung 3.7: Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

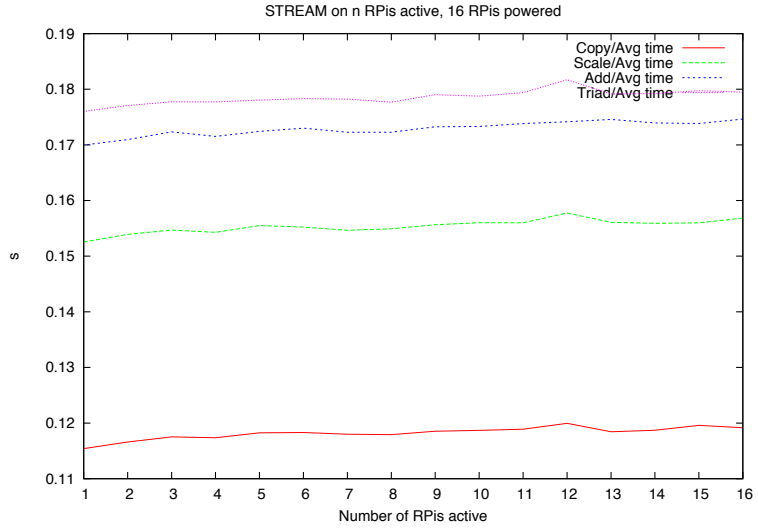


Abbildung 3.8: Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

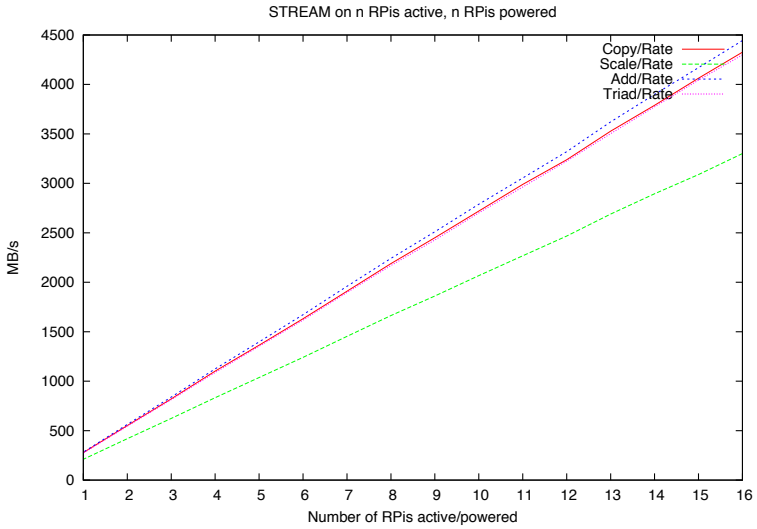


Abbildung 3.9: Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.

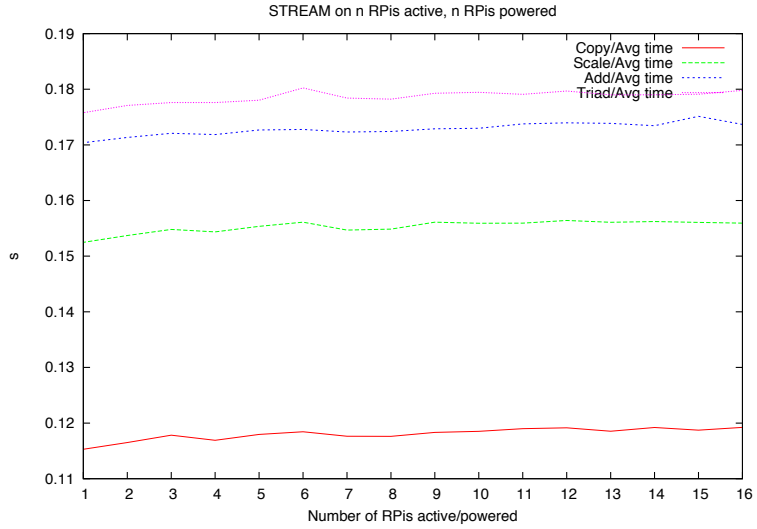


Abbildung 3.10: Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.

# 4 Interpretation

Das folgende Kapitel dient der Bewertung und Einordnung der ermittelten Messwerte. Dazu werden zunächst die Messwerte für alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered verglichen. Anschließend werden sie den Messwerten des RPi-Einzelrechners gegenübergestellt, sowohl den selbst ermittelten als auch den bisher publizierten. Abschließend werden eine Einordnung in die Top500-Liste vorgenommen und Grenzen des Versuchsaufbaus diskutiert.

## 4.1 HPLinpack/Bramble

Tabellen 4.1 und 4.2 stellen die Messergebnisse für beide Ausführungen von HPLinpack gegenüber:

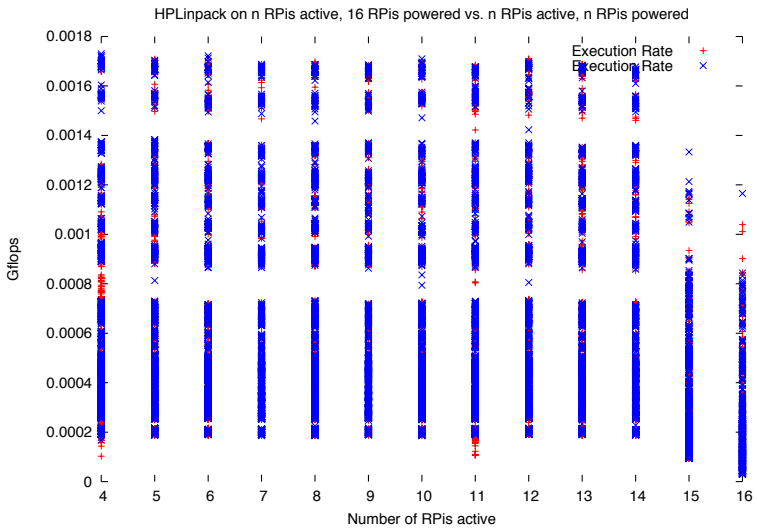


Abbildung 4.1: Ausführungsrate in GFLOPS für HPLinpack mit allen RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

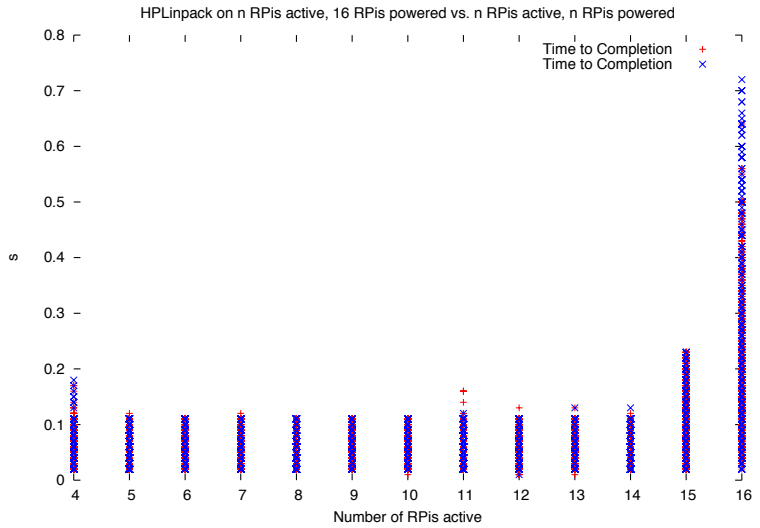


Abbildung 4.2: Ausführungszeit in s für HPLinpack alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

Die Diagramme lassen vermuten, dass keine signifiganten Unterschiede zwischen der Ausführung von HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered (rot) und n RPi-Nodes aktiv/n RPi-Nodes powered (blau) bestehen. Die folgende Tabelle stellt die minimalen, maximalen und durchschnittlichen Werte für Ausführungsrate und Ausführungsdauer beider Läufe gegenüber:

| | **16 RPis powered** | **n RPis powered** |
|---|---|---|
| **Ausführungsrate/Min:** | 3.65E-05 GFLOPS | 3.01E-05 GFLOPS |
| **Ausführungsrate/Max:** | 1.71E-03 GFLOPS | 1.73E-03 GFLOPS |
| **Ausführungsrate/Avg:** | 5.10E-04 GFLOPS | 5.15E-04 GFLOPS |
| **Ausführungsdauer/Min:** | 0.01 s | 0.01 s |
| **Ausführungsdauer/Max:** | 0.64 s | 0.72 s |
| **Ausführungsdauer/Avg:** | 0.07 s | 0.07 s |

Abbildung 4.3: Gegenüberstellung HPLinpack alle RPis powered vs. nur aktive RPis powered.

Leichte Unterschiede zeigen sich somit in der mimimalen, maximalen und durchschnittlichen Ausführungsrate sowie der maximalen Ausführungszeit. Dabei überschreitet die Abweichung nur bei der minimalen Ausführungsrate sowie der maximalen Ausführungszeit die Größenordnung einer Nachkommastelle.

Hieran ist von Bedeutung, dass das Herunterfahren nicht aktiver RPi-Nodes lediglich bei der maximalen und durchschnittlichen Ausführungsrate einen positiven Effekt erzielt. Die übrigen Abweichungen zeigen sogar einen negativen Effekt: Die mimimale Ausführungsrate sinkt um 6.40E-06 = 0.0000064 GFLOPS, die durchschnittliche Ausführungsrate um 5.00E-06 = 0.0000050 GFLOPS und die maximale Ausführungsdauer steigt um 0.08 s.

## 4.2 STREAM/Bramble

Noch deutlicher zeigt sich dieser Effekt bei STREAM:

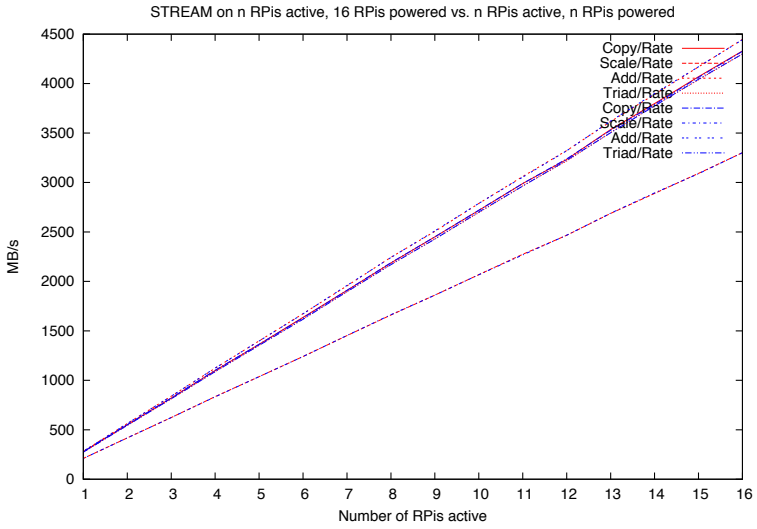


Abbildung 4.4: Ausführungsrate in MB/s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

Auch hier zeigen die nahezu deckungsgleichen Ergebnisdaten mit 16 RPis powered (rote) vs. n RPi-Nodes powered (blau), keinen nennenswerten Effekt des Herunterfahrens nicht aktiver RPi-Nodes auf Ausführungsrate und Ausführungsdauer. Die folgenden Tabellen 4.6 und 4.7 mit der Gegenüberstellung der maximalen, minimalen und durchschnittlichen Ausführungsrate und Ausführungsdauer der STREAM-Module bestätigen dies:

Unterschiede in der Größenordnung von mehr als einer Nachkommastelle zeigen sich in der maximalen und durchschnittlichen Ausführungsrate von COPY, der minimalen und maximalen Ausführungsrate von ADD, der maximalen und durchschnittlichen Ausführungsrate von TRIAD. Bei der Ausführungsdauer unterscheiden sich beide Runs in fast allen Werten geringfügig, jedoch nur in einer Funktion (minimale Ausführungsdauer für ADD) in mehr als vier Nachkommastellen. Wenn man in Betracht zieht, dass HPLinpack für die Ausführungsdauer Werte lediglich auf zwei Nachkommastellen genau ausgibt, sind diese Abweichungen bis auf letztgenannte vernachlässigbar.

Die Abweichungen sind auch hier nur teilweise erwartungsgemäß: Lediglich die durchschnittliche Ausführungsrate für TRIAD sinkt beim Herunterfahren nicht aktiver RPi-Nodes um 0.5 MB/s. Maximale und durchschnittliche Ausführungsrate für COPY, minimale und maximale Ausführungsrate für ADD und maximale Ausführungsrate von TRIAD sinken gegenüber der Ausführung mit 16 RPi-Nodes powered. Die Steigerung der minimalen Ausführungsdauer von ADD um 0.001062 s gegenüber 16 RPi-Nodes zeigt ebenfalls einen negativen Effekt des Herunterfahrens.

Schlussfolgernd lässt sich feststellen, dass das Herunterfahren nicht aktiver RPi-Nodes wie bei HPLinpack keinen nennenswerten Einfluss auf Ausführungsrate und Ausführungsdauer der STREAM-Module hat.

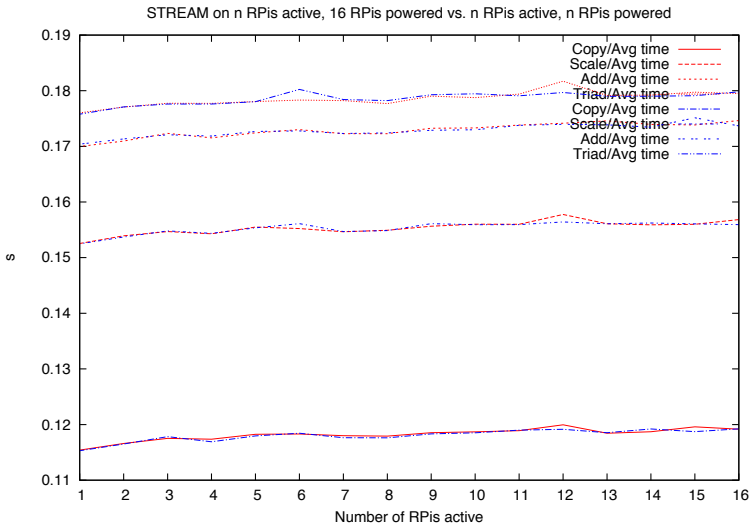


Abbildung 4.5: Ausführungszeit in s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

## 4.3 Vergleich Bramble/RPi-Einzelrechner

Eine Gegenüberstellung der Messwerte von Bramble und RPi-Einzelrechner ist nur mit Einschränkungen möglich. Wie bereits dargestellt bestehen große Unterschiede zwischen den Betriebssystemen und den verwendeten Implementierungen der Benchmarks, die zudem teilweise unterschiedliche Ergebnisparameter haben. Tabellen 4.8, 4.9 und 4.10 stellen die Ergebnisse von Linpack, Whetstone und STREAM auf Bramble, RPi-Einzelrechner den Ergebnissen der Benchmark-Autoren[1] gegenüber.

[1] Quellen: `http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm` und `http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html`.

| | **16 RPis powered** | **n RPis powered** |
|---|---|---|
| **Copy/Rate Min:** | 278.0 MB/s | 278.2 MB/s |
| **Copy/Rate Max:** | 4330.0 MB/s | 4326.2 MB/s |
| **Copy/Rate Avg:** | 2310.5 MB/s | 2309.4 MB/s |
| **Scale/Rate Min:** | 210.4 MB/s | 210.2 MB/s |
| **Scale/Rate Max:** | 3301.2 MB/s | 3301.5 MB/s |
| **Scale/Rate Avg:** | 1757.2 MB | 1757.3 MB/s |
| **Add/Rate Min:** | 283.2 MB/s | 282.9 MB/s |
| **Add/Rate Max:** | 4447.1 MB/s | 4443.5 MB/s |
| **Add/Rate Avg:** | 2367.4 MB/s | 2368.1 MB/s |
| **Triad/Rate Min:** | 274.0 MB/s | 274.1 MB/s |
| **Triad/Rate Max:** | 4300.0 MB/s | 4298.9 MB/s |
| **Triad/Rate Avg:** | 2292.9 MB/s | 2293.4 MB/s |

Abbildung 4.6: Gegenüberstellung STREAM (Ausführungsrate) alle RPis powered vs. nur aktive RPis powered.

## 4.4 Einordnung in Top500

Die Klassifizierung der Supercomputer in der Top500-Liste basiert auf den Ergebnissen von HPLinpack (Ausführungsrate in GFLOPS). Eine fiktive Einordnung des Bramble in die aktuelle Bestenliste vom November 2013[2] zeigt Schaubild 4.11.

## 4.5 Grenzen des Versuchsaufbaus

Der dargestellte Versuchsaufbau stößt in drei Bereichen an seine Grenzen:

1. **Vergleichbarkeit der Benchmark-Implementierungen**
   Die ausgewählten Benchmarks liefern teilweise höchst unterschiedliche Parameter als Rückgabewerte, je nachdem ob eine MPI-Implementierung oder eine Single-CPU-Implementierung verwendet wird. Das gilt besonders fÜr Linpack: Einziger Rückgabewert für Linpack 100 ist die Ausführungsrate in MFLOPS, während HPLinpack für jeden der 864 Tests Ausführungsrate in GFLOPS und Ausführungsdauer in s zurückgibt. Messwerte unterschiedlicher Implementierungen, die notwendigerweise auf einem Cluster und einem Einzelrechner verwendet werden, sind daher nur bedingt aussagekräftig. STREAM hingegen unterscheidet nicht zwischen den Ausgabewerten seiner unterschiedlichen Implementierungen, was die Vergleichbarkeit deutlich verbessert.

2. **Vergleichbarkeit des Benchmark-Designs**
   Auch zwischen den Designs der Benchmarks bestehen große Unterschiede. Whetstone in der gewählten, an den RPi-Einzelrechner angepassten Implementierung liefert nach wie vor lediglich Ergebnisse in MWIPS. STREAM liefert standardmäßig Durchschnittswerte für die Ausführungszeit jedes Moduls. HPLinpack testet deutlich mehr Module als die vorgenannten Benchmarks, liefert jedoch nur Einzelergebnisse für Ausführungsrate und -dauer, keine Mittelwerte. Vergleichbarkeit zwischen den Ergebnissen unter-

[2] Vgl. `http://www.top500.org/list/2013/11`.

| | 16 RPis powered | n RPis powered |
|---|---|---|
| **Copy/Time Min:** | 0.115415 s | 0.115293 s |
| **Copy/Time Max:** | 0.119951 s | 0.119217 s |
| **Copy/Time Avg:** | 0.118210 s | 0.118054 s |
| **Scale/Time Min:** | 0.152540 s | 0.152485 s |
| **Scale/Time Max:** | 0.157763 s | 0.155317 s |
| **Scale/Time Avg:** | 0.155374 s | 0.155317 s |
| **Add/Time Min:** | 0.169977 s | 0.1710393 s |
| **Add/Time Max:** | 0.174637 s | 0.175129 s |
| **Add/Time Avg:** | 0.172898 s | 0.172851 s |
| **Scale/Time Min:** | 0.176024 s | 0.175788 s |
| **Scale/Time Max:** | 0.179734 s | 0.179793 s |
| **Scale/Time Avg:** | 0.178582 s | 0.178587 s |

Abbildung 4.7: Gegenüberstellung STREAM (Ausführungszeit) alle RPis powered vs. nur aktive RPis powered.

| | Bramble/16 | Bramble/n | RPi | RPi/Autor |
|---|---|---|---|---|
| **Ausführungsrate:** | 0.00051 | 0.00052 GFLOPS | 0.04131 GFLOPS | 0.04184 GFLOPS |

Abbildung 4.8: Linpack auf Bramble und RPi-Einzelrechnern.

| | Bramble/16 | Bramble/n | RPi | RPi/Autor |
|---|---|---|---|---|
| **Ausführungsrate:** | – | – | 255.154 MWIPS | 270.460 MWIPS |
| **Ausführungsdauer:** | – | – | 10.190 s | 9.983 s |

Abbildung 4.9: Whetstone auf RPi-Einzelrechnern.

| | Bramble/16 | Bramble/n | RPi | RPi/Autor |
|---|---|---|---|---|
| **Copy/Rate:** | 2310.5 MB/s | 2309.4 MB/s | 274.4 MB/s | 209.5 MB/s |
| **Copy/Avg time:** | 0.11821 s | 0.1181 s | 0.5868 s | 0.1549 s |
| **Scale/Rate:** | 1757.2 MB/s | 1757.3 MB/s | 209.3 MB/s | 190.1 MB/s |
| **Scale/Avg time:** | 0.1554 s | 0.1553 s | 0.7664 s | 0.1703 s |
| **Add/Rate:** | 2367.4 MB/s | 2368.1 | 287.2 MB/s | 259.1 MB/s |
| **Add/Avg time:** | 0.1729 s | 0.1729 s | 0.8381 s | 0.1943 s |
| **Triad/Rate:** | 2292.9 MB/s | 2293.4 MB/s | 271.1 MB/s | 248.8 MB/s |
| **Triad/Avg time:** | 0.1786 s | 0.1786 s | 0.8868 s | 0.2002 s |

Abbildung 4.10: STREAM auf Bramble und RPi-Einzelrechnern.

schiedlicher Benchmarks ist daher noch weniger gegeben als zwischen den verschiedenen Implementierungen eines Benchmarks, selbst wenn sie dieselbe Komponente testen.

Das gilt insbesondere für HPLinpack zum Tragen. Die in Kap. 4.1 dargestellten Abweichungen sind deutlich geringer als die unterschiedlichen Ergebnisse der einzelnen Module einer einzigen Ausführung auf n RPi-Nodes. Auch wenn das dem Design des

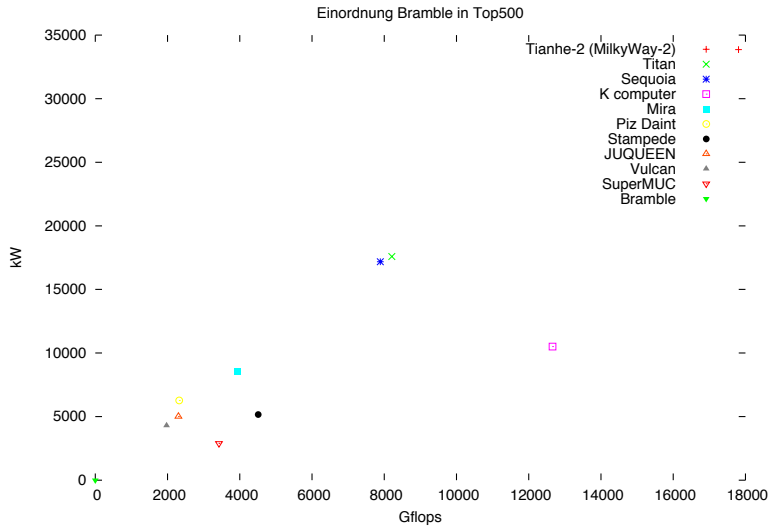


Abbildung 4.11: Einordnung des Bramble in Top500 Nov. 2013.

Benchmarks vorgesehen ist, ist die Interpretation an Hand selbst ermittelter Mittelwerte sicherlich weniger aussagekräftig als eine Interpretation von Mittelwerten als Teil des Designs.

3. **Zuverlässigkeit von Hardware, IP- und SSH-Kommunikation des Bramble**
   Schließlich erweist sich das Setup des RPi-Clusters als limitierender Faktor. Auch nach weitestgehendem Ausschluss aller in Kap. 3.2.2 genannten Störfaktoren blieb der Versuchsaufbau fehleranfällig. Die Durchführung erforderte i.d.R. die Anwesenheit einer Aufsichtsperson, um bei Störungen die LEDs zu kontrollieren, Kabel zu überprüfen etc.. Der aktuelle Status Bramble kann kaum als stabil bezeichnet werden und es ist nicht auszuschließen, dass Fehlerfälle stellenweise Einfluss auf die Messergebnisse nehmen.

   Erschwerend kommt hinzu, dass sich die RPi-Nodes des Bramble beim Herunterfahren anders verhalten als die eines RPi-Einzelrechners. Dieser schickt beim Aufruf von `shutdown` eine Broadcast-Message, dass das System jetzt heruntergefahren wird. Der Rechner kann sicher vom Stromnetz getrennt werden, wenn nur noch die rote Status-LED für die Stromversorgung leuchtet.

   Beim Bramble hingegen sind alle Status-LEDs weiterhin aktiv, auch wenn der RPi-Node nicht mehr auf ein `ping` reagiert, d.h. keine IP-Verbindung vom Server oder einem anderen RPi-Node aus aufgebaut werden kann. Der Broadcast einer Shutdown-Nachricht wurde nur nur gelegentlich festgestellt. Somit erscheint es schwierig, den Zeitpunkt festzustellen, an dem ein RPi-Node tatsächlich vom Netz gegangen ist. Der aktuelle Versuchsaufbau dürfte hierdurch jedoch nicht beeinträchtigt werden: Auch wenn die Status-LEDs weiterhin aktiv sind, ist der Rechner nicht mehr durch `ping` zu erreichen und somit nicht mehr im Netzwerk aktiv.

# 5 Zusammenfassung und Ausblick

> I have converted my Classic Benchmarks to run on the Linux based Raspberry Pi. These are Whetstone, [...], Linpack and Livermore Loops. [...] The Livermore Loops benchmark was used to accept the first supercomputer. So the main bragging rights are:
>
> In 1978, the Cray 1 supercomputer cost $7 Million, weighed 10,500 pounds and had a 115 kilowatt power supply. It was, by far, the fastest computer in the world. The Raspberry Pi costs around $70 (CPU board, case, power supply, SD card), weighs a few ounces, uses a 5 watt power supply and is more than 4.5 times faster than the Cray 1.
>
> My bragging rights are that I developed and ran benchmarks, including Whetstones, on Serial 1 Cray 1[1].

In der vorliegenden Arbeit wurde versucht, HPC-Benchmarks auf einem RPi-Cluster lauffähig zu machen und zu evaluieren. Dieser Proof of Concept war erfolgreich:

Die ausgewählten HPC-Benchmarks konnten mit Ausnahme von Whetstone nach dem Ausschluss vorhandener Störfaktoren auf dem Bramble ausgeführt werden. Die Experiment-Suite konnte mit kleinen Änderungen am bestehenden Datenbank-Schema in den übergeordneten Versuchsaufbau integeriert werden. Die erzielten Messwerte ergaben ein kohärentes Bild, wurden denen eines RPi-Einzelrechners gegenübergestellt sowie in den größeren Kontext der Top500-Rankings und der bisher erzielten Messwerte der Benchmark-Autoren eingeordnet.

Der Versuchsaufbau reiht sich damit Bestrebungen der letzten Monate, einen Beowulf-Cluster aus Raspberry Pi-Einzelrechnern für verteilte Berechnungen heranzuziehen. Wie Longbottom im obigen Zitat und die Projektleiter der Bramble-Projekte schreiben: Der Raspberry Pi ist in die Welt der Cluster und (zumindest ehemaligen) Supercomputer eingetreten[2] und erzielt in den betrachteten Kategorien CPU-Performance und Speicher-Bandbreite durchaus respektable Ergebnisse.

Schwierigkeiten zeigten sich vor allem in der vorhandenen Bramble-Infrastruktur, sowohl Hardware als auch IP/SSH-Kommunikation betreffend. Mögliche zukünftige Arbeiten umfassen daher zwei Felder:

Der vorhandene Bramble ist deutlich verbesserungsfähig. Vor allem die Hardware zeigt Schwächen, nicht nur die Stromversorgung betreffend. Der physische Aufbau ist extrem beengt, sodass Ziehen und erneutes Einstecken von Mini-USB- und Netzwerkkabeln nur eingeschränkt möglich ist und die Gefahr besteht, die übrige Hardware dabei zu beschädigen. Da die Unterbrechung der Stromversorgung die einzige Möglichkeit zum Reboot eines RPi

[1] Quelle: `http://www.raspberrypi.org/forum/viewtopic.php?f=31&t=44080`.
[2] Vgl. auch `http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html`.

ist und eine unterbrochene Netzwerkverbindung nur durch Triggern des Netzwerkkabels wieder hergestellt werden kann, ist dieser Punkt systemkritisch für zukünftige Untersuchungen. Abhilfe ließe sich u.U. durch den Einbau von Reset-Knöpfen auf den einzelnen RPi-Nodes schaffen[3]. Auch die Platzierung der RPi-Nodes in einem aufrecht stehenden Rack statt eines liegenden Metallgehäuses wäre von Vorteil[4], damit die häufig benötigten Anschlüsse besser zugänglich sind.

In einem grösseren Rahmen wäre es interessant, früher oder später auf eine MPI-Implementierung von Whetstone für Raspbian zurückgreifen zu können. Auch die Evaluierung weiterer, bisher nicht betrachteter HPC-Benchmarks steht noch aus. Die wichtigsten Erkenntnisgewinne sind jedoch zu erwarten, wenn in naher Zukunft hoffentlich ein offener Treiber für die RPi-GPU vorliegt. Nachdem Broadcom vor wenigen Wochen erstmals die vollständige Spezifikation der GPU zur Verfügung gestellt hat[5], hat die Raspberry Pi Foundation einen entsprechenden Wettbewerb ausgeschrieben[6]. Die Ergebnisse und neuen Einsatzmöglichkeiten der bisher kaum anprogrammierbaren GPU, die deutlich leistungsfähiger ist als die hier schwerpunktmäßig untersuchte CPU, dürfen mit Spannung erwartet werden.

[3] Vgl. z.B. `http://raspi.tv/2012/making-a-reset-switch-for-your-rev-2-raspberry-pi`.
[4] Vgl. [Kie13] und [CCB+13].
[5] Vgl. `http://blog.broadcom.com/chip-design/android-for-all-broadcom-gives-developers-keys-\discretionary{-}{}{}to-the-videocore-kingdom/`.
[6] Vgl. `http://www.raspberrypi.org/competition-rules`.

# 6 Anhang

## 6.1 Shellskripte

### startBenchmarks.sh

```
#!/bin/bash

# Start script for execution of n benchmark scripts on 20 RPi-Nodes

# remove old machinefile
rm -f /srv/libraries/etc/mpich-3.0.4-shared/machinefile

touch /srv/libraries/etc/mpich-3.0.4-shared/machinefile
for host in {pi0{1..9},pi{10..20}}
do
echo "$host" >> /srv/libraries/etc/mpich-3.0.4-shared/machinefile
done

# mount shared directory /srv on all RPis
for host in {pi0{1..9},pi{10..20}}
do
ssh root@$host 'mount /srv'
done

# navigate to execution directory
cd /srv/experimentsuite/

# loop over n benchmarks
for benchmark in STREAM hpl-2.1
do
./$benchmark.sh
done
```

### loadGeneratorConfigHpl.sh

```
#!/bin/bash

# Setup load generator configuration for hplinpack
# For documentation purpose
# For new setup: change 'hpl-config' into some other string

ssh rpi-user@careme<<'ENDSSH'
```

```
# mysql -B: don't use history file, disable interactive behavior
# mysql -s: silent mode
# mysql -e: execute query and exit

# define load generator
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
LoadGenerator (name,description) VALUES ('hpl-2.1','Benchmark')"

# define load generator configuration key
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
ENUM_LoadGeneratorConfigurationKey (\`value\`) VALUES ('hpl-config')"

# assign load generator configuration key to load generator configuration
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
LoadGeneratorConfiguration (\`key\`) VALUES ('hpl-config')"
ENDSSH
```

### loadGeneratorConfigStream.sh

```
#!/bin/bash

# Setup load generator configuration for STREAM
# For documentation purpose
# For new setup: change 'stream-config' into some other string

ssh rpi-user@careme<<'ENDSSH'
# mysql -B: don't use history file, disable interactive behavior
# mysql -s: silent mode
# mysql -e: execute query and exit

# define load generator
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
LoadGenerator (name,description) VALUES ('STREAM','Benchmark')"

# define load generator configuration key
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
ENUM_LoadGeneratorConfigurationKey (\`value\`) VALUES ('stream-config')"

# assign load generator configuration key to load generator configuration
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
LoadGeneratorConfiguration (\`key\`) VALUES ('stream-config')"
ENDSSH
```

### hpl-2.1.sh

```
#!/bin/bash
```

```
# Run Hplinpack on n downto 4 RPi-nodes, run again and shutdown RPi-node i afterwards

# create output files
touch results/hpl-2.1_‘date +%y%m%d‘.txt results/hpl-2.1_shutdown‘date +%y%m%d‘.txt

# loop over n RPis downto 4
for host in pi20 pi19 pi18 pi17 pi16 pi15 pi14 pi13 pi12 pi11 pi10 pi09 pi08
do
    n=${host/pi/}
    n=$(echo $n|sed ’s/^0*//’)
    echo "Number of active RPis: $n"
    echo "Number of powered RPis: 20"
    starttime=‘date +%s‘
    echo "Start time: $starttime"

# Setup experiment suite in database
# One experiment suite for every program run needed

    ssh rpi-user@careme<<ENDSSH

# initialize experiment suite
mysql -u rpi-user -prpiWerte rpiWerte -Bse "INSERT INTO
ExperimentSuite (granularityLevel,objective,executionStartedAt) VALUES
(’Cluster Blackbox’,’experiment no1’,$starttime)"
ENDSSH

# get experiment suite id
ssh rpi-user@careme<<ENDSSH
touch /tmp/myid.txt
mysql -u rpi-user -prpiWerte rpiWerte -Bse "SELECT id FROM
ExperimentSuite WHERE executionStartedAt=$starttime" > /tmp/myid.txt
ENDSSH

# read in experiment suite id from tmp file on careme
myid=$(ssh rpi-user@careme "cat /tmp/myid.txt")

# remove tmp file from careme
ssh rpi-user@careme "rm /tmp/myid.txt"

ssh rpi-user@careme<<ENDSSH
# insert number of active RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,
experimentSuiteId) VALUES (’NumberOfActiveRPis’,’${n}’,$myid)"

# insert number of powered RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
```

```
"INSERT INTO ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,
experimentSuiteId) VALUES (’NumberOfPoweredRPis’,’14’,$myid)"

# map load generator configuration to experiment suite
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO N2M_loadGConf2expSuite (loadGeneratorConfigurationId,experimentSuiteId)
VALUES (6,$myid)"
ENDSSH

echo "Database setup complete"

# log number of active RPis/powered RPis to results/hpl-2.1_‘date +%y%m%d‘.txt
echo "Active RPis: "$n"/Powered RPis: 20" >> results/hpl-2.1_‘date +%y%m%d‘.txt

# start benchmark on n RPis, log to results/hpl-2.1_‘date +%y%m%d‘.txt
    mpiexec -n $n -machinefile /srv/libraries/etc/mpich-3.0.4-shared/machinefile
-wdir /srv/benchmarks/bin/hpl-2.1 /srv/benchmarks/bin/hpl-2.1/xhpl >>
results/hpl-2.1_‘date +%y%m%d‘.txt

    echo "HPlinpack finished on $n RPis active, 20 RPis powered"
done

# loop over n RPis downto 4
for host in pi20 pi19 pi18 pi17 pi16 pi15 pi14 pi13 pi12 pi11 pi10 pi09 pi08
do
    n=${host/pi/}
    n=$(echo $n|sed ’s/^0*//’)
    echo "Number of active RPis: $n"
    echo "Number of powered RPis: $n"
    starttime=‘date +%s‘
    echo "Start time: $starttime"

# Setup experiment suite in database
# One experiment suite for every program run needed

# initialize experiment suite
    ssh rpi-user@careme<<ENDSSH

# initialize experiment suite
mysql -u rpi-user -prpiWerte rpiWerte -Bse "INSERT INTO
ExperimentSuite (granularityLevel,objective,executionStartedAt)
VALUES (’Cluster Blackbox’,’experiment no2’,$starttime)"
ENDSSH


# get experiment suite id
ssh rpi-user@careme<<ENDSSH
```

```
touch /tmp/myid.txt
mysql -u rpi-user -prpiWerte rpiWerte -Bse "SELECT id FROM
ExperimentSuite WHERE executionStartedAt=$starttime" > /tmp/myid.txt
ENDSSH

# read in experiment suite id from tmp file on careme
myid=$(ssh rpi-user@careme "cat /tmp/myid.txt")
# echo $myid

# remove tmp file from careme
ssh rpi-user@careme "rm /tmp/myid.txt"

ssh rpi-user@careme<<ENDSSH
# insert number of active RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,experimentSuiteId)
VALUES ('NumberOfActiveRPis','${n}',$myid)"

# insert number of powered RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,experimentSuiteId)
VALUES ('NumberOfPoweredRPis','14',$myid)"

# map load generator configuration to experiment suite
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse "INSERT INTO
N2M_loadGConf2expSuite (loadGeneratorConfigurationId,experimentSuiteId)
VALUES (6,$myid)"
ENDSSH

echo "Database setup complete"

# log number of active RPis/powered RPis to results/hpl-2.1_‘date +%y%m%d‘.txt
echo "Active RPis: "$n"/Powered RPis: 20" >> results/hpl-2.1_shutdown‘date +%y%m%d‘.txt

# start benchmark on n RPis, log to results/hpl-2.1_shutdown‘date +%y%m%d‘.txt
    mpiexec -n $n -machinefile /srv/libraries/etc/mpich-3.0.4-shared/machinefile
-wdir /srv/benchmarks/bin/hpl-2.1 /srv/benchmarks/bin/hpl-2.1/xhpl
>> results/hpl-2.1_shutdown‘date +%y%m%d‘.txt
    echo "HPlinpack finished on $n RPis active, $n RPis powered"
    ssh $host 'shutdown -hP 0'
done

# create input file for database
touch results/hpl-2.1_db‘date +%y%m%d‘.txt

# find all lines in first output file beginning with 'WR' (result lines)
# and print together with 3 following lines (timestamp lines)
```

```
grep '^WR' results/hpl-2.1_‘date +%y%m%d‘.txt -A 3
> results/hpl-2.1_db‘date +%y%m%d‘.txt

# find all lines in second output file beginning with 'WR' (result lines)
# and print together with 3 following lines (timestamp lines)
grep '^WR' results/hpl-2.1_shutdown‘date +%y%m%d‘.txt -A 3
>> results/hpl-2.1_db‘date +%y%m%d‘.txt

# remove all empty lines
sed -i '/^$/d' results/hpl-2.1_db‘date +%y%m%d‘.txt

# remove all lines containing "start" (only "end" needed)
sed -i '/start/d' results/hpl-2.1_db‘date +%y%m%d‘.txt

# remove all lines containing only '--' (result from grep -A)
sed -i '/--/d' results/hpl-2.1_db‘date +%y%m%d‘.txt

# transform database input file to lines containing space seperated values
sed -i 's/ \{2,\}/ /g' results/hpl-2.1_db‘date +%y%m%d‘.txt
sed -i 'N;s/\n/ /' results/hpl-2.1_db‘date +%y%m%d‘.txt

ssh rpi-user@careme<<'ENDSSH'
cat /srv/nfs-share/experimentsuite/results/hpl-2.1_db‘date +%y%m%d‘.txt |
cut -d' ' -f6,7,12,13,14,15 | while read line
do
    arr=($line)
    time=${arr[0]}
    echo "Time: $time"
    gflops=${arr[1]}
    echo "Gflops: $gflops"
    timestamp=${arr[@]:2}
    unixtime=$(date -d "${timestamp}" "+%s")
    echo $unixtime
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
    "INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
    VALUES ('Time',$time,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
    "INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
    VALUES ('Gflops',$gflops,$unixtime)"
done
ENDSSH
```

### STREAM.sh

```
#!/bin/bash

# Run STREAM on1 RPi-Nodes, run again and shutdown RPi-Node n a
```

```
# create output files
touch results/STREAM_‘date +%y%m%d‘.txt results/STREAM_shutdown‘date +%y%m%d‘.txt

# loop over n RPis
for host in pi20 pi19 pi18 pi17 pi16 pi15 pi14 pi13
pi12 pi11 pi10 pi09 pi08 pi07 pi06 pi05
do
    n=${host/pi/}
    n=$(echo $n|sed 's/^0*//')
    n=$(($n - 4))
#    if [ $n > 0 ]; then
echo "Number of active RPis: $n"
echo "Number of powered RPis: 16"
starttime=‘date +%s‘
echo "Start time: $starttime"

# Setup experiment suite in database
# One experiment suite for every program run needed

ssh rpi-user@careme<<ENDSSH
# initialize experiment suite
mysql -u rpi-user -prpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuite (granularityLevel,objective,executionStartedAt)
VALUES ('Cluster Blackbox','experiment no3',$starttime)"
ENDSSH

# get experiment suite id
ssh rpi-user@careme<<ENDSSH
touch /tmp/myid.txt
mysql -u rpi-user -prpiWerte rpiWerte -Bse
"SELECT id FROM ExperimentSuite WHERE executionStartedAt=$starttime" > /tmp/myid.txt
ENDSSH

# read in experiment suite id from tmp file on careme
myid=$(ssh rpi-user@careme "cat /tmp/myid.txt")
# echo $myid

# remove tmp file from careme
ssh rpi-user@careme "rm /tmp/myid.txt"
ssh rpi-user@careme<<ENDSSH

# insert number of active RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,
experimentSuiteId) VALUES ('NumberOfActiveRPis','${n}',$myid)"
```

```
# insert number of powered RPis
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuiteConfiguration (\\\‘key\\\‘,\\\‘value\\\‘,
experimentSuiteId) VALUES (’NumberOfPoweredRPis’,’16’,$myid)"

# map load generator configuration to experiment suite
mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO N2M_loadGConf2expSuite (loadGeneratorConfigurationId,experimentSuiteId)
VALUES (7,$myid)"
ENDSSH

echo "Database setup complete"

# start benchmark on n RPis, log to results/STREAM_‘date +%y%m%d‘.txt
mpiexec -n $n -machinefile /srv/libraries/etc/mpich-3.0.4-shared/machinefile
-wdir /srv/benchmarks/bin/STREAM /srv/benchmarks/bin/STREAM/stream
>> results/STREAM_‘date +%y%m%d‘.txt

# add unix timestamp to results file
echo "Unixtime: ‘date +%s‘" >> results/STREAM_‘date +%y%m%d‘.txt

echo "STREAM finished on $n RPis active, 16 RPis powered"
done


# loop over n RPis
for host in pi20 pi19 pi18 pi17 pi16 pi15 pi14 pi13
pi12 pi11 pi10 pi09 pi08 pi07 pi06 pi05
do
    n=${host/pi/}
    n=$(echo $n|sed ’s/^0*//’)
    n=$(($n - 4))
#    if [ $n > 0 ]; then
echo $n
echo "Number of active RPis: $n"
echo "Number of powered RPis: $n"
starttime=‘date +%s‘
echo "Start time: $starttime"

# Setup experiment suite in database
# One experiment suite for every program run needed
ssh rpi-user@careme<<ENDSSH
# initialize experiment suite
mysql -u rpi-user -prpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuite (granularityLevel,objective,executionStartedAt)
VALUES (’Cluster Blackbox’,’experiment no4’,$starttime)"
ENDSSH
```

```
# get experiment suite id
ssh rpi-user@careme<<ENDSSH
t ouch /tmp/myid.txt
mysql -u rpi-user -prpiWerte rpiWerte -Bse
"SELECT id FROM ExperimentSuite WHERE executionStartedAt=$starttime" > /tmp/myid.txt
ENDSSH

# read in experiment suite id from tmp file on careme
myid=$(ssh rpi-user@careme "cat /tmp/myid.txt")

# remove tmp file from careme
ssh rpi-user@careme "rm /tmp/myid.txt"

ssh rpi-user@careme<<ENDSSH
        # insert number of active RPis
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuiteConfiguration (\\\`key\\\`,\\\`value\\\`,experimentSuiteId)
VALUES ('NumberOfActiveRPis','${n}',$myid)"

    # insert number of powered RPis
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO ExperimentSuiteConfiguration (\\\`key\\\`,\\\`value\\\`,experimentSuiteId)
VALUES ('NumberOfPoweredRPis','${n}',$myid)"

    # map load generator configuration to experiment suite
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO N2M_loadGConf2expSuite (loadGeneratorConfigurationId,experimentSuiteId)
VALUES (7,$myid)"
ENDSSH

echo "Database setup complete"
# log number of active RPis/powered RPis to results/STREAM_`date +%y%m%d`.txt

# start benchmark on n RPis, log to results/STREAM_`date +%y%m%d`.txt
mpiexec -n $n -machinefile /srv/libraries/etc/mpich-3.0.4-shared/machinefile
-wdir /srv/benchmarks/bin/STREAM /srv/benchmarks/bin/STREAM/stream
>> results/STREAM_shutdown`date +%y%m%d`.txt

ssh $host 'shutdown -hP 0'

# add unix timestamp to results file
echo "Unixtime: `date +%s`" >> results/STREAM_shutdown`date +%y%m%d`.txt
echo "STREAM finished on $n RPis active, $n RPis powered"

done
```

```
# create input file for database
touch results/STREAM_db‘date +%y%m%d‘.txt

# find all lines in first output file beginning with ’Copy’ (first result line)
# and print together with 7 following lines
grep ’^Copy’ results/STREAM_‘date +%y%m%d‘.txt -A 7
> results/STREAM_db‘date +%y%m%d‘.txt

# find all lines in second output file beginning with ’Copy’ (result lines)
# and print together with 7 following lines (timestamp lines)
grep ’^Copy’ results/STREAM_shutdown‘date +%y%m%d‘.txt -A 7
>> results/STREAM_db‘date +%y%m%d‘.txt

# remove all lines containing only ’--’ (result from grep -A)
sed -i ’/--/d’ results/STREAM_db‘date +%y%m%d‘.txt

# remove all lines beginning with blank
sed -i ’/^ /d’ results/STREAM_db‘date +%y%m%d‘.txt

# insert new line delimiter before ’Copy’
sed -i ’s/Copy/|Copy/’ results/STREAM_db‘date +%y%m%d‘.txt

# transform to lines containing space seperated values
sed -i ’s/ \{2,\}/ /g’ results/STREAM_db‘date +%y%m%d‘.txt

# transform whole file into one line (for read line)
sed -i ’:a;N;$!ba;s/\n/ /g’ results/STREAM_db‘date +%y%m%d‘.txt

# substitute ’|’ with ’\n’ (new line delimiter)
sed -i ’s/|/\n/g’ results/STREAM_db‘date +%y%m%d‘.txt

# remove empty first line
sed -i ’/^$/d’ results/STREAM_db‘date +%y%m%d‘.txt

ssh rpi-user@careme<<’ENDSSH’
cat /srv/nfs-share/experimentsuite/results/STREAM_db‘date +%y%m%d‘.txt |
cut -d’ ’ -f2,3,7,8,12,13,17,18,22 | while read line
do
    # echo $line
    arr=($line)
    copy_rate=${arr[0]}
    # echo $copy_rate
    copy_time=${arr[1]}
    # echo $copy_time
    scale_rate=${arr[2]}
    # echo $scale_rate
    scale_time=${arr[3]}
```

```
    # echo $scale_time
    add_rate=${arr[4]}
    # echo $add_rate
    add_time=${arr[5]}
    # echo $add_time
    triad_rate=${arr[6]}
    # echo $triad_rate
    triad_time=${arr[7]}
    # echo $triad_time
    unixtime=${arr[8]}
    # echo $unixtime
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Copy/Rate’,$copy_rate,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Copy/Avg time’,$copy_time,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Scale/Rate’,$scale_rate,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Scale/Avg time’,$scale_time,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Add/Rate’,$add_rate,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Add/Avg time’,$add_time,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Triad/Rate’,$triad_rate,$unixtime)"
    mysql --user=rpi-user --password=rpiWerte rpiWerte -Bse
"INSERT INTO MeasurementValue (parameter,\‘value\‘,measuredAt)
VALUES (’Triad/Avg time’,$triad_time,$unixtime)"
done
ENDSSH
```

## 6.2 Ergebnisse der ausgewählten Benchmarks auf dem RPi-Einzelrechner

### 6.2.1 Linpack 100

```
########################################
Unrolled Double Precision Linpack Benchmark - Linux Version in ’C/C++’
```

```
Optimisation Opt 3 32 Bit

norm resid      resid           machep         x[0]-1          x[n-1]-1
   1.7    7.41628980e-14   2.22044605e-16  -1.49880108e-14  -1.89848137e-14

Times are reported for matrices of order          100
1 pass times for array with leading dimension of  201

      dgefa      dgesl      total     Mflops       unit      ratio
    0.01613    0.00057    0.01669      41.14     0.0486     0.2981

Calculating matgen overhead
        10 times   0.01 seconds
       100 times   0.14 seconds
       200 times   0.28 seconds
       400 times   0.57 seconds
       800 times   1.13 seconds
Overhead for 1 matgen      0.00141 seconds

Calculating matgen/dgefa passes for 1 seconds
        10 times   0.17 seconds
        20 times   0.35 seconds
        40 times   0.70 seconds
        80 times   1.40 seconds
Passes used         57

Times for array with leading dimension of 201

      dgefa      dgesl      total     Mflops       unit      ratio
    0.01609    0.00054    0.01663      41.29     0.0484     0.2970
    0.01603    0.00054    0.01658      41.43     0.0483     0.2960
    0.01610    0.00054    0.01664      41.25     0.0485     0.2972
    0.01609    0.00054    0.01663      41.29     0.0484     0.2970
    0.01603    0.00061    0.01663      41.28     0.0484     0.2970
Average                               41.31

Calculating matgen2 overhead
Overhead for 1 matgen      0.00137 seconds

Times for array with leading dimension of 200

      dgefa      dgesl      total     Mflops       unit      ratio
    0.01447    0.00054    0.01502      45.73     0.0437     0.2682
    0.01437    0.00051    0.01489      46.13     0.0434     0.2658
    0.01447    0.00051    0.01498      45.84     0.0436     0.2675
    0.01445    0.00051    0.01496      45.89     0.0436     0.2672
    0.01441    0.00051    0.01492      46.02     0.0435     0.2665
```

```
Average                                 45.92

##########################################

From File /proc/cpuinfo
Processor : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS  : 697.95
Features  : swp half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant   : 0x0
CPU part   : 0xb76
CPU revision   : 7

Hardware   : BCM2708
Revision   : 000f
Serial     : 00000000e98379f1


From File /proc/version
Linux version 3.6.11+ (dc4@dc4-arm-01) (gcc version 4.7.2 20120731 (prerelease)
(crosstool-NG linaro-1.13.1+bzr2458 - Linaro GCC 2012.08) )
#538 PREEMPT Fri Aug 30 20:42:08 BST 2013


Unrolled Double  Precision       41.31 Mflops

##############################################
```

### 6.2.2 Whetstone

```
##########################################
Single Precision C Whetstone Benchmark Opt 3 32 Bit, Sat Nov 30 15:22:14 2013

Calibrate
       0.04 Seconds          1   Passes (x 100)
       0.19 Seconds          5   Passes (x 100)
       0.96 Seconds         25   Passes (x 100)
       4.80 Seconds        125   Passes (x 100)

Use 260  passes (x 100)

From File /proc/cpuinfo
Processor : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS  : 697.95
Features  : swp half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
```

```
CPU architecture: 7
CPU variant   : 0x0
CPU part   : 0xb76
CPU revision   : 7

Hardware   : BCM2708
Revision   : 000f
Serial     : 00000000e98379f1


From File /proc/version
Linux version 3.6.11+ (dc4@dc4-arm-01) (gcc version 4.7.2 20120731 (prerelease)
(crosstool-NG linaro-1.13.1+bzr2458 - Linaro GCC 2012.08) )
#538 PREEMPT Fri Aug 30 20:42:08 BST 2013


          Single Precision C/C++ Whetstone Benchmark

Loop content                 Result              MFLOPS      MOPS   Seconds

N1 floating point     -1.12475013732910156        97.643              0.051
N2 floating point     -1.12274742126464844       100.883              0.346
N3 if then else        1.00000000000000000                  690.831    0.039
N4 fixed point        12.00000000000000000                  423.573    0.193
N5 sin,cos etc.        0.49911010265350342                    5.050    4.284
N6 floating point      0.99999982118606567        86.081              1.629
N7 assignments         3.00000000000000000                  498.602    0.096
N8 exp,sqrt etc.       0.75110864639282227                    2.724    3.551

MWIPS                                            255.154             10.190
```

### 6.2.3 STREAM

```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
```

```
Each test below will take on the order of 736819 microseconds.
   (= 736819 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:             274.4     0.586838     0.583109     0.599162
Scale:            209.3     0.766437     0.764583     0.772736
Add:              287.2     0.838107     0.835557     0.842724
Triad:            271.1     0.886793     0.885394     0.889097
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

# Abbildungsverzeichnis

# Literaturverzeichnis


[Bal12] Balakrishnan, Nikilesh: *Building and Benchmarking a Low Power ARM Cluster*. 2012. – http://www.epcc.ed.ac.uk/sites/default/files/Dissertations/2011-2012/Submission-1126390.pdf

[BL08] Buhl, Hans ; Laartz, Jürgen: Warum Green IT nicht ausreicht – oder: Wo müssen wir heute anpacken, damit es uns übermorgen immer noch gut geht. In: *Wirtschaftsinformatik* (2008), S. 261–265. – http://dx.doi.org/10.1365/s11576-008-0058-5

[CCB+13] Cox, Simon ; Cox, James ; Boardman, Richard u. a.: Iridis-pi: A Low-cost, Compact Demonstration Cluster. In: *Cluster Computing* (2013), S. 1–10. – http://dx.doi.org/10.1007/s10586-013-0282-7

[CW76] Curnow, Harold ; Wichmann, Brian: A Synthetic Benchmark. In: *The Computer Journal* (1976), S. 43–49. – http://http://comjnl.oxfordjournals.org/content/19/1/43.full.pdf

[DLP03] Dongarra, Jack ; Luszczek, Piotr ; Petitet, Antoine: The LINPACK Benchmark: Past, Present and Future. In: *Concurrency and Computation: Practice and Experience* (2003), S. 803–820. – http://onlinelibrary.wiley.com/doi/10.1002/cpe.728/pdf

[FH12] Fichter, Klaus ; Hintermann, Ralph: Energieverbrauch und Energiekosten von Servern und Rechenzentren in Deutschland. Aktuelle Trends und Einsparpotenziale bis 2015 / Borderstep Institut für Innovation und Nachhaltigkeit. Berlin, Mai 2012. – Forschungsbericht. – http://www.bitkom.org/files/documents/Kurzstudie__Borderstep_l_Rechenzentren(1).pdf

[Kie13] Kiepert, Joshua: *Creating a Raspberry Pi-Based Bewoulf Cluster*. 2013. – http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf

[Kli13] Klinger, Maximilian: *Evaluating the Feasibility and Performance of a Model Raspberry Pi Beowulf Cluster*. 2013. – http://www.nm.ifi.lmu.de/pub/Fopras

[Lan12] Langer, Alexander: *Raspberry Pi Handbuch*, Februar 2012. – http://raspberrycenter.de/handbuch/raspberry-pi-handbuch

[LB12] Lange, Walter ; Bogdan, Martin: *Entwurf und Synthese von Eingebetteten Systemen: Ein Lehrbuch*. München : Oldenbourg, 2012. – ISBN 3486718401

[LDK+05] Luszcek, Piotr ; Dongarra, Jack ; Koester, David u. a.: Introduction to the HPC Challenge Benchmark Suite / Lawrence Berkeley National Laboratory.

Berkeley, April 2005. – Forschungsbericht. – http://escholarship.org/uc/item/6sv079jp

[McC95] McCalpin, John: A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers. In: *IEEE TCCA Newsletter* (1995), S. 19–25. – http://www.cs.virginia.edu/~mccalpin/papers/balance

[McC05] McCalpin, John: *STREAM Benchmark*. 2005. – http://www.cs.virginia.edu/~mccalpin/STREAM_Benchmark_2005-01-25.pdf

[Ou13] Ou, Jun: *Pi and the Sky*. 2013. – https://www.duo.uio.no/bitstream/10852/37445/4/Ou-Jun.pdf

[Pow12] Powers, Shawn: The Open-source Classroom: Your First Bite of Raspberry Pi. In: *Linux Journal* (2012), S. 48–54. – http://dl.acm.org/ft_gateway.cfm?id=2422332&ftid=1329297&dwn=1&CFID=403460446&CFTOKEN=89580898

[Pre11] Prevezanos, Christoph: *Computer-Lexikon 2012*. München : Markt+Technik, 2011. – ISBN 9783827247285

[Rec06] Rechenberg, Peter: *Informatik-Handbuch*. München : Hanser, 2006. – ISBN 9783446401853

[Sch13] Schmidt, Maik: *Raspberry Pi. Einstieg, Optimierung, Projekte*. Heidelberg : dpunkt, 2013. – ISBN 9783864900327

[Wei90] Weicker, Reinhold: An Overview of Common Benchmarks. In: *Computer* (1990), S. 65–75. – http://dlojewski.dl.funpic.de/download/Diplomarbeit/Andere_Dok/Dhry_Whet/OverviewOfCommonBenchmarks.pdf