

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

Christian Straube

Abgabetermin: 31. März 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 niedrigschwelliger 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 aufführen lassen und evaluiert die Ergebnisse. Ein Schwerpunkt liegt dabei auf dem Energieverbrauch des Clusters bei unterschiedlichen Versuchsaufbauten.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Hintergrund	1
1.2 Fragestellung	1
1.3 Vorgehensweise	1
1.4 Struktur	2
2 Grundlagen und Begriffsbildung	3
2.1 Definition: <i>Benchmarking</i>	3
2.2 Definition: <i>Leistung</i>	4
2.3 Definition: <i>Performance</i>	4
2.4 Benchmarks	5
2.4.1 Linpack	5
2.4.2 Whetstone	6
2.4.3 STREAM	7
2.5 Spezifikation des RPi Modell B	9
2.5.1 Physischer Aufbau	9
2.5.2 Betriebssystem	10
2.6 Raspberry Pi-Cluster als Testumfeld	10
2.6.1 Physischer Aufbau	10
2.6.2 Betriebssystem und Filesystem	11
2.6.3 Zugriff auf die Cluster-Komponenten	11
3 Versuchsaufbau und -ablauf	13
3.1 Zielsetzung	13
3.2 Aufbau und Art der Messung	13
3.2.1 Versuchsaufbau Ri-Einzelrechner	14
3.2.2 Versuchsaufbau Bramble	14
3.3 Ergebnisse	18
3.3.1 RPi-Einzelrechner: Linpack 100, Whetstone und STREAM	19
3.3.2 Bramble: HPLinpack	19
3.3.3 Bramble: STREAM	20
4 Interpretation	25
4.1 HPLinpack/Bramble	25
4.2 STREAM/Bramble	27
4.3 Vergleich Bramble/RPi-Einzelrechner	28
4.4 Einordnung in Top500	29
4.5 Grenzen des Versuchsaufbaus	29
5 Zusammenfassung und Ausblick	33

Inhaltsverzeichnis

6 Anhang	35
6.1 Shellskripte	35
6.2 Ergebnisse der ausgewählten Benchmarks auf dem RPi-Einzelrechner	45
6.2.1 Linpack 100	45
6.2.2 Whetstone	47
6.2.3 STREAM	48
Abbildungsverzeichnis	51
Literaturverzeichnis	53

1 Einleitung

Seit Beginn seiner Entwicklung im Jahr 2009 booms 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 einzelnen 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 Performance eines RPi-Einzelrechners ermittelt. Anschließend wird die Performance eines RPi-Clusters bei Ausführung der Benchmarks evaluiert. Im Fokus steht dabei der Energieverbrauch, 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.

1 Einleitung

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 gepowerter 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 gepowerter 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 gematcht 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 auf dem RPi-Einzelrechner und dem Cluster 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. Eine solche Routine kann für einen Vergleich mit HPC-Rechnern herangezogen werden. Im Folgenden wird daher auf grundlegende Begriffe eingegangen, die für die nachfolgende Untersuchung von Bedeutung sind. Anschließend werden die verwendeten Benchmarks (vgl. 2.4) und ihre Anpassung an den 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

2 Grundlagen und Begriffsbildung

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 mit *Benchmarking* kurz das **Standardisieren von Arbeit** bezeichnet.

2.2 Definition: Leistung

Die physikalische Größe *Leistung* ist als *Energie pro Zeit* definiert. In der Informatik hingegen wird damit meist die *Rechenleistung* bezeichnet, also Datenverarbeitungsgeschwindigkeit, Geschwindigkeit der eingesetzten Maschinenteile sowie einzelner Anwendungen.

Wichtige Aspekte bei der Leistungsbewertung eines Rechnersystems sind das zu untersuchende System, die Workload und die Methode zur Leistungsermittlung (vgl. [Rec06]), außerdem

[...] die *Leistungskenngrößen* oder *-maßzahlen* (*performance metrics*), die für das vorliegende Rechnersystem und die Ziele der Leistungsanalyse relevant sind [Rec06].

Die hier verwendete Definition greift den ersten Aspekt auf: Mit *Leistung* ist die **Time to completion** gemeint, d.h. die Zeit, die ein Prozess oder ein Programm(teil) bis zum erfolgreichen Abschluss benötigt. Bei Rechenberg wird dies als „*Programmlaufzeit*“ [Rec06] bezeichnet¹.

2.3 Definition: Performance

Für den Begriff *Performance* im Zusammenhang mit Rechnern existieren verschiedene Definitionen, häufig gleichbedeutend mit der *Rechenleistung* oder nicht klar davon abgegrenzt. Erklärungen wie „*Zeitverhalten von Programmen und Geräten*“ oder „*Leistungsfähigkeit eines Computersystems*“ scheinen der Fragestellung nicht ganz gerecht zu werden:

The performance of a computer is a complicated issue, a function of many inter-related 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 eine genauere Definition:

¹Andere Klassen von Kenngrößen, die neben der Zeit zur Leistungsbewertung herangezogen werden, sind *Durchsatz* und *Auslastung*. Abweichende zeitliche Kenngrößen sind z.B. *Bedienzeit* (*Service time*) oder *TTR* (*Time to Response*) (vgl. ebd.).

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 werden Performance eines RPis und eines RPi-Clusters ermittelt und der Leistungsfähigkeit eines Supercomputers gegenübergestellt. Der Maßstab für die Performance ist dabei die *Time to Completion* eines gegebenen Benchmark-Programms². *Performance* soll daher hier als ***Arbeit pro Zeit*** verstanden werden.

2.4 Benchmarks

Aus der Fülle an existierenden Benchmarks wurden drei für die Untersuchung ausgewählt. Kriterien waren dabei Erprobtheit, Verlässlichkeit und Angemessenheit für das zu untersuchende System³. Außerdem muss gewährleistet sein, dass der Benchmark überhaupt auf das gewählte System anwendbar und an dieses anpassbar ist⁴.

Mit dem Ziel, das Skalierungsverhalten eines RPi-Clusters zu untersuchen, wurden drei etablierte HPC-Benchmarks ausgewählt, 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 man zwischen der Linpack-Library und dem Linpack-Benchmark unterscheiden: Die Library ist eine numerische Programmbibliothek zum Lösen von linearen Gleichungssystemen. Sie wurde inzwischen von anderen Bibliotheken abgelöst, galt aber lange als Standard⁵. Der Linpack-Benchmark basiert auf zwei Programmen dieser Bibliothek⁶. Er wurde hauptsächlich von Jack Dongarra, einem der Autoren der Linpack-Bibliothek, entwickelt und wird seit den 1970er Jahren zur Klassifizierung von Rechnern verwendet⁷. 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

²Vgl. hierzu Curnow 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].“

³Vgl. Weickers Kriterien für die Auswahl eines Benchmarks: „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].“

⁴Aus diesem Grund mussten z.B. SHOC und LLNL IOR ausscheiden. SHOC ist nicht lauffähig auf dem RPi, da Open CL nicht unterstützt wird. LLNL IOR wiederum benötigt ein POSIX-, MPIIO- oder HDF5-Interface. Zum Ausscheiden von Whetstone auf dem RPi-Cluster vgl. Kap. 2.4.3.

⁵Für die genaue Spezifikation von Linpack vgl. [DLP03]. Der Sourcecode für Linpack 1000 in Fortran findet sich unter <http://www.netlib.org/benchmark/1000d>.

⁶„Linpack was designed out of a real, purposeful program that is now used as a benchmark [Wei90].“

⁷„Over the years additional performance data was added [...] and today the collection includes over 1300 different computer systems [DLP03].“

2 Grundlagen und Begriffsbildung

Computing genannt⁸.

Funktionsweise

Linpack ist ein Benchmark zur Ermittlung der CPU-Performance. Dazu werden Fließpunkt-Operationen auf einer Matrix⁹, die intern in eine lineare Darstellung umgewandelt wird, durchgeführt und das Ergebnis in *FLOPS* („Floating Point Operations Per Second“) ausgegeben¹⁰. Dabei erreicht der derzeit leistungsfähigste Supercomputer, *Tianhe-2 (Milky-Way-2)*, z.B. einen *Rmax*-Wert („Maximal LINPACK performance achieved“) von 33862.7 TFLOPS. SuperMUC, der jüngst auf Platz 10 der Top500 gerankt wurde, erreicht einen *Rmax*-Wert von 2897.0 TFLOPS.

Beim Vergleich Linpack-Ergebnissen ist die Größe des Arrays bzw. der Matrix von Bedeutung, da eine Änderung wegen geringerer Datenlokalität zu starken Abweichungen der Ergebnisse führen kann¹¹. Wichtig ist das beim Vergleich der Ergebnisdaten verschiedener Rechnerarchitekturen. I.d.R. werden daher o.g. standardisierte Größen verwendet, auch wenn der Quellcode eine Veränderung der Matrixgröße 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 veröffentlicht¹². Somit liegen bereits Vergleichswerte für einen RPi-Einzelrechner vor. Auch Implementierungen von Linpack für verteilte Systeme unabhängig von den Top500-Rankings sind im Umlauf¹³. Zu prüfen wird sein, wie diese für den RPi-Cluster nutzbar gemacht werden können.

2.4.2 Whetstone

Auch Whetstone ist als Benchmark im HPC-Bereich und zur Klassifizierung von Einzelrechnern seit vielen Jahren etabliert¹⁴. Er wurde 1976 von Roy Longbottom u.A. entwickelt und gilt als erstes Programm, das jemals explizit für das Benchmarking industrieller Standards designet wurde¹⁵. Wie Linpack misst er die CPU-Performance.

⁸ „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].“ Eine detaillierte Beschreibung des Sourcecode von HPLinpack findet sich ebd..

⁹ LINPACK 100: 100×100 -Matrix; LINPACK 1000: 1000×1000 -Matrix; HPLinpack: $n \times n$ -Matrix (vgl. ebd.).

¹⁰ Genau genommen ist es so, dass zwar hauptsächlich, aber nicht ausschließlich Floating Point-Operationen ausgeführt werden. 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]).

¹¹ Vgl. ebd..

¹² Die hier verwendete Implementierung in C für den RPi-Einzelrechner findet sich unter http://www.roylongbottom.org.uk/Raspberry_Pi_Benchmarks.zip.

¹³ Vgl. <http://www.netlib.org/benchmark/hpl>.

¹⁴ „[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].“

¹⁵ Vgl. ebd..

Funktionsweise

Die Funktionsweise von Whetstone ähnelt Linpack. Allerdings werden nicht nur Fließkomma-Berechnungen ausgeführt, sondern auch mathematische Funktionen, Integer-Arithmetik, If-Statements etc. kommen zur Anwendung¹⁶. Jedes dieser als genuin betrachteten Module wird in eine For-Schleife eingebettet und viele Male hintereinander ausgeführt. Die Ausgabe der Berechnungsergebnisse ist Teil des Programms¹⁷. Ein Framework bildet den äußeren Programmrahmen und steuert die Ausgaben der einzelnen Module. Die Ergebnisse wurden zunächst in *Whetstone Instructions per Second* gemessen, heutige Implementierungen liefern Ergebnisse in MFLOPS¹⁸.

Whetstone auf dem Raspberry Pi

Whetstone erschien wie für den RPi gemacht, da er sich als Standard für Mini-Computer etabliert hat¹⁹. Er wurde vom Entwickler selbst für den RPi adaptiert und auf diesem getestet²⁰. Auch hier liegen also bereits Vergleichswerte für einen RPi-Einzelrechner vor.

Wie für die meisten etablierten HPC-Benchmarks existieren Implementierungen in zahlreichen höheren Programmiersprachen wie C, C++, Fortran und Java²¹. Auch hier wird aus nahe liegenden Gründen auf eine Implementierung in C für den RPi-Einzelrechner zurückgegriffen²². 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 häufig nicht ausreicht, die Leistung eines Rechners oder Rechnersystems lediglich an Hand der CPU-Performance zu bewerten. Zudem lässt sich eine Tendenz beobachten, dass Rechnersysteme gezielt für die Ausführung bestimmter Benchmarks wie HPLinpack optimiert werden, um z.B. in Rankings bessere Wertungen zu erreichen.

Die Zunahme an Prozessorleistung führt jedoch nicht notwendigerweise zu einer Verbesserung der Gesamtleistung eines Systems. Gerade bei Supercomputern besteht ein potenzieller

¹⁶Ein Grund hierfür ist, dass die ursprüngliche Programmversion nicht komplex genug war, um ein durchschnittliches FORTRAN-Programm zu simulieren. Gegenüber der ursprünglichen Implementierung in ALGOL 60 war ein damals üblicher FORTRAN-Compiler in der Lage, Zwischenergebnisse in schnellen Registern abzuspeichern und darauf zurückzugreifen, sodass die gemessene CPU-Leistung deutlich höher ausfiel als erwartet (vgl. [CW76]).

¹⁷Allerdings: „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 (ebd.).“

¹⁸Für eine detaillierte Darstellung der Entwicklung und Implementierung von Whetstone vgl. ebd.. Hier findet sich auch der ursprüngliche Sourcecode in ALGOL 60.

¹⁹Ein Grund hierfür ist, dass Whetstone in den 70er Jahren für Maschinen mit signifikant geringerer Rechenleistung entwickelt worden war. Auch 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 (ebd.).“

²⁰Vgl. <http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm>.

²¹Vgl. z.B. <http://freespace.virgin.net/roy.longbottom>.

²²Der hier verwendete Sourcecode findet sich ebenfalls unter http://www.roylongbottom.org.uk/Raspberry_Pi_Benchmarks.zip.

2 Grundlagen und Begriffsbildung

Flaschenhals oft nicht in der mangelnden Leistung der einzelnen Rechnerkerne, sondern in der Performance von I/O-Zugriffen auf den nicht flüchtigen Speicher²³. Zudem besteht zwischen der Performance, die mit Benchmarks wie HPLinpack ermittelt wird, und den tatsächlichen Anforderungen heutiger Programme oft kein proportionaler Zusammenhang. D.h. Maschinen, die sehr gute Werte bei Linpack oder Whetstone erzielen, können diese unter der Workload konkreter Anwendungen unter Umständen oftmals nicht halten. Andererseits können deutlich leistungsschwächere Rechnersysteme bei bestimmten Anwendungen ähnlich gute Werte erreichen wie weitaus besser gerankte Maschinen aus den Top500, wenn diese größtenteils auf gecachten Daten arbeitet und wenige Zugriffe auf den nicht flüchtigen Speicher erfordert²⁴. Mittlerweile ist STREAM auch Bestandteil der HPCChallenge Benchmark-Suite, ein Framework mit dem Ziel, realistische und variable Anforderungen aus der Anwendungspraxis stärker in die Bewertung von HPC-Rechnersystemen einzubeziehen²⁵.

Funktionsweise

Um unter diesen Voraussetzungen das Verhältnis von CPU-Performance zu Speichergeschwindigkeit in die Bewertung der Rechenleistung einzubeziehen, wird der STREAM-Benchmark eingesetzt. STREAM ist ein synthetischer Benchmark und basiert auf einem Programm aus der wissenschaftlichen Praxis von John McCalpin, das die dauerhafte Speicher-Bandbreite²⁶ für angrenzende Speicherzugriffe langer Vektoren²⁷ ermittelt. Die Vektoren bzw. Arrays müssen lang genug sein, dass sie nicht im 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) festlegt²⁸. STREAM durchläuft bei seiner Ausführung vier Module (Copy, Scale, Add und Triad) und liefert für jedes Modul die durchschnittliche, maximale und minimale Time to Completion 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 verhindern²⁹.

STREAM auf dem Raspberry Pi

STREAM kann, anders als etwa HPLlinpack, auch auf weniger als 4 CPUs ausgeführt werden. So wurden die ersten Ergebnisse für einen RPi-Einzelrechner bereits 2012 veröffentlicht³⁰. Im ursprünglichen Versuchsaufbau waren Linpack bzw. HPLinpack sowie Whetstone als Load-Generatoren vorgesehen. Da sich trotz intensiver Recherche keine Implementierung von Whetstone zur verteilten Ausführung auf einem Raspbian-Betriebssystem finden ließ, musste Whetstone durch STREAM ersetzt werden. Dort war eine Version in Fortran bereits vorinstalliert³¹. Auf dem RPi-Einzelrechner wurde eine Implementierung in C verwendet³².

²³Vgl. [McC95].

²⁴Vgl. ebd.. Dieses Phänomen zeigte sich im HPC-Bereich seit ca. 1990 und wurde von Eugene Brooks bei der Supercomputing-Konferenz 1989 als „attack of the killer micros“ bezeichnet (vgl. ebd.).

²⁵Vgl. [LDK⁺05].

²⁶Engl. „sustainable memory bandwidth [McC95]“

²⁷Engl. „sustainable memory bandwidth for contiguous, long-vector memory accesses (ebd.)“.

²⁸Vgl. ebd. sowie [McC05].

²⁹Vgl. [McC05]. Quellcode und Ergebnisse für STREAM finden sich unter <http://www.cs.virginia.edu/stream>.

³⁰Vgl. http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html.

³¹Der Quellcode findet sich unter http://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.f.

³²Der Quellcode findet sich unter <http://www.cs.virginia.edu/stream/FTP/Code/stream.c>.

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 $85.60\text{ mm} \times 53.98\text{ mm} \times 17\text{ mm}$ tatsächlich nur geringfügig. Auf dieser Platine sind alle Komponenten verbaut, die den RPi zu einem voll funktionsfähigen Rechner machen:

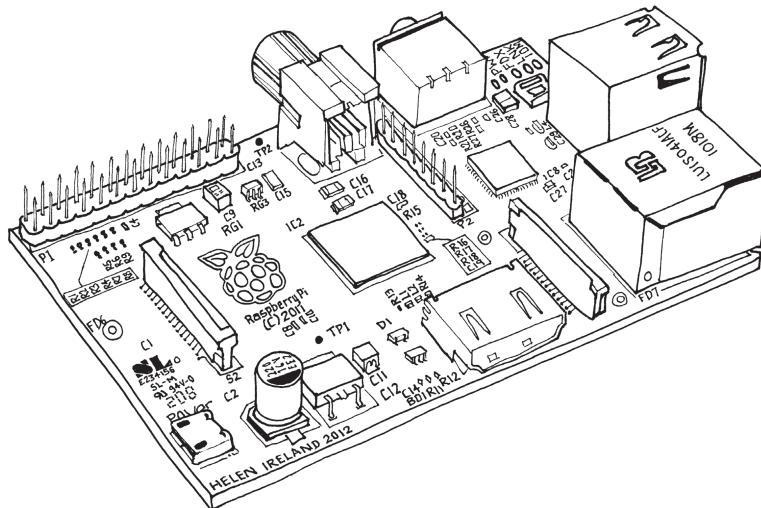


Abbildung 2.1: Ein Raspberry Pi Modell B (Quelle: [BCH⁺12]).

2.5.1 Physischer Aufbau

SoC, CPU, GPU und RAM

Der RPi ist mit einem Broadcom BCM2835 als SoC ausgestattet. Es enthält CPU (ein ARM1176JZFS-Prozessor) und GPU (ein Broadcom VideoCore IV-Koprozessor)³³. Auffällig ist, dass die CPU des RPi (ein ARM-Chip, wie er häufig in Mobiltelefonen verbaut ist), relativ schwach ist im Vergleich zur GPU, die Full HD-Auflösung und das hardwarebeschleunigte Rendern verschiedener Videoformate unterstützt. Das Modell B verfügt über 512 MB SDRAM³⁴.

Schnittstellen

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 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 mindestens 700 mA/5 V benötigt³⁵. Für den Versuchsaufbau sind vor allem Ethernetanschluss, Mini-USB und SD-Karte von Bedeutung. Beim RPi-Einzelrechner erfolgt der Netzanschluss über

³³Vgl. [Lan12].

³⁴Gegenüber dem Modell A mit 256 MB (vgl. ebd.). Dieser 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]).

³⁵Vgl. ebd.. Der Vollständigkeit halber sei auch das Vorhandensein einer GPIO mit sechs Anschlüssen und jeweils 26 Pins erwähnt. Darüber können, vergleichbar einem Arduino-Board, z.B. LEDs, Sensoren und Displays angesteuert werden.

einen DSL-Router. Die Stromversorgung geschieht über die Steckdose, während beim Cluster Stromversorgung und Netzanschluss zentral über den Server des RPi-Clusters erfolgen. RPi-Einzelrechner und jeder RPi-Node verfügen über eine eigene 4 GB Class 10 SD-Karte (auf die Cluster-Architektur wird in Kap. 2.6 genauer eingegangen).

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 Linux³⁶. Als Betriebssystem für den RPi-Einzelrechner wurde Raspbian gewählt, die offizielle Distribution der Raspberry Pi Foundation³⁷. Auf den Einsatz von NOOBS³⁸ oder anderer zusätzlicher Bootloader wurde verzichtet³⁹.

2.6 Raspberry Pi-Cluster als Testumfeld

In den letzten Monaten lässt sich die Tendenz beobachten, eine größere Anzahl von Raspberry Pis zu einem Cluster zu koppeln⁴⁰. Die hier verwendete Architektur wird im Folgenden umrissen⁴¹.

2.6.1 Physischer Aufbau

Der RPi-Cluster besteht aus 20 RPi Modell B-Einzelrechnern (DNS-Namen pi01 – pi20), die jeweils über ein Ethernet-Kabel mit dem zentralen x86-Server (im Folgenden mit seinem DNS-Namen `careme` bezeichnet) verbunden sind. Er besteht hauptsächlich aus einem Mini-ITX-Mainboard und einer Gigabit Ethernet-Netzwerkkarte. Alle Komponenten befinden sich in einem modifizierten Tower-Metallgehäuse, das auf die Seite gedreht und oben offen gelassen wurde. Darin sind außerdem ein 24 Port Gigabit-Switch, die Stromversorgung des Servers und der RPi-Nodes sowie die Kühlung integriert⁴².

³⁶Vgl. [Pow12]. Daneben gibt es Implementierungen von RISC OS und Plan 9.

³⁷Vgl. <http://www.raspberrypi.org/faqs>. Das verwendete Image findet sich unter http://downloads.raspberrypi.org/raspbian_latest.

³⁸Die Raspberry Pi-Foundation empfiehlt besonders für Einsteiger die *New Out Of Box Software* und bietet entsprechend präparierte SD-Karten zum Kauf an (vgl. <http://www.raspberrypi.org/archives/tag/noobs>).

³⁹In diesem Fall fungiert die ausführbare Datei `start.elf`, die nach einer ausführbaren Datei `kernel.img` sucht und diese lädt, als Bootloader (vgl. [Kli13]).

⁴⁰Weitere Projekte werden in [CCB⁺13], [Kie13], [Bal12] und [Ou13] dargestellt.

⁴¹Für eine detaillierte Darstellung vgl. [Kli13].

⁴²Ein solches System relativ kostengünstiger Rechner mit einem BSD- oder Linux-Betriebssystem, die über IP kommunizieren, wird im Allgemeinen als *Beowulf* bezeichnet (vgl. [Kie13] und [Kli13]). Häufig dient es dem Ersatz oder der Simulation eines Supercomputers. Im Zusammenhang mit RPis ist auch der Begriff *Bramble* gebräuchlich, der im Folgenden verwendet wird (vgl. www.raspberrypi.org/archives/tag/bramble).

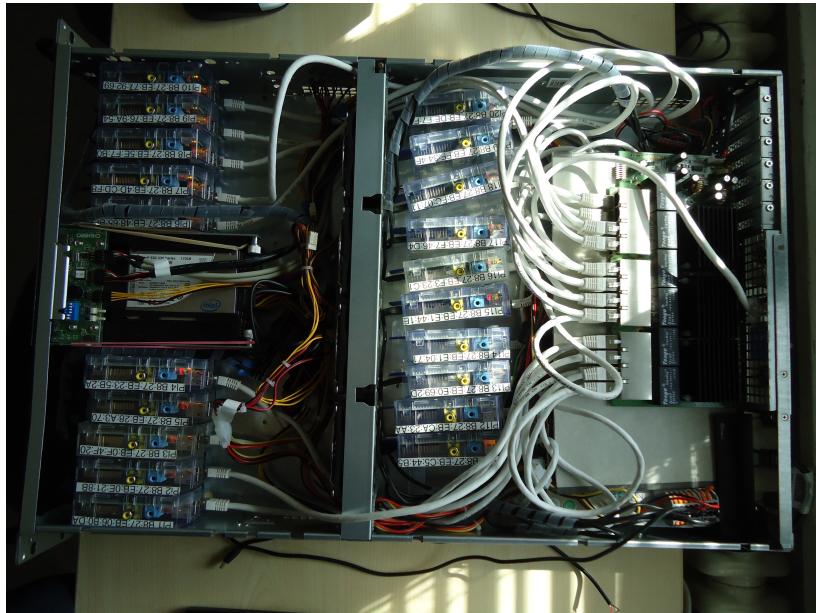


Abbildung 2.2: Der hier verwendete Bramble.

2.6.2 Betriebssystem und Filesystem

Auf den RPi-Nodes läuft das o.g. Betriebssystem Raspbian, auf `careme` eine Standard-Debian-Version. Ein Charakteristikum eines Beowulf-Clusters bzw. Bramble ist, dass es im Gegensatz zu einem Supercomputer kein Shared Memory-Interface und keine Cache-Kohärenz gibt. Die Kommunikation zwischen den einzelnen Komponenten via Ethernet ist zudem relativ langsam. Es stellt sich daher die Frage nach dem Filesystem und der Synchronisierung der einzelnen Nodes.

Um das Filesystem der einzelnen Nodes mit dem NFS des Servers zu synchronisieren, wurde hier eine Read-Only-Schicht für die Slaves (Nodes) und eine Read-Write-Schicht für den Master (Server) verwendet⁴³. Um trotzdem den gemeinsamen Zugriff von Server und RPi-Nodes auf ein Speichermedium zu ermöglichen, wurde ein geshartes Verzeichnis `/srv` bzw. `/srv/nfs-share` eingerichtet.

2.6.3 Zugriff auf die Cluster-Komponenten

Welche Möglichkeiten gibt es nun, verteilte Anwendungen auf dem Bramble auszuführen bzw. überhaupt die einzelnen Komponenten anzusprechen? Bei einem RPi-Einzelrechner nutzt man normalerweise einen USB-Port und den HDMI-Ausgang als `stdin` und `stdout`. Eine andere Möglichkeit besteht darin, den RPi über den Ethernet-Port mit einem Router zu verbinden und über SSH darauf zuzugreifen⁴⁴.

Der Zugriff auf den Bramble erfolgt aus dem internen Netzwerk über IP und SSH. Die einzelnen RPi-Nodes werden von `careme` aus über SSH adressiert (private Adressen 10.0.0.2

⁴³Vgl. [Kli13].

⁴⁴Diese Variante wurde für das hier verwendeten Testgerät gewählt auf Grund von Problemen beim Anschluss älterer Monitore ohne HDMI-Eingang (vgl. z.B. <http://www.raspberrypi.org/forum/viewtopic.php?f=91&t=34061>).

2 Grundlagen und Begriffsbildung

bis 10.0.0.21). Es gibt auf jedem RPi-Node den Raspbian-Standard-User `pi`, der hier nicht verändert wurde⁴⁵. Um z.B. auf das gesharte Verzeichnis `/srv` zuzugreifen, sind `root`-Berechtigungen erforderlich⁴⁶.

Beim hier verwendeten Versuchsaufbau wurde auf `careme` mit eingeschränkten Rechten als User `rpi-user` gearbeitet, auf den einzelnen RPis als `root`. Das bedeutet, dass alle Änderungen im gesharten Verzeichnis notwendigerweise von einem der RPi-Nodes aus vorgenommen werden müssen. Dazu wurde vorab RPi-Node `pi03` als Berechnungs-Node definiert, von dem aus alle Skripte ausgeführt, Programme installiert werden etc.. Da die Skripte zur Ausführung der Benchmarks häufig das Herunterfahren einzelner RPi-Nodes vorsehen (vgl. Kap. 6), wurden die Benchmarks grundsätzlich auf maximal n=19 RPi-Nodes ausgeführt (vgl. Kap. 3). 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-Nodes und die Anzahl der darauf auszuführenden Prozesse zu spezifizieren, muss ein entsprechendes Machinefile erstellt und im gesharten Verzeichnis abgelegt werden, das der Ausführung von MPICH mit `mpexec` als Parameter übergeben wird (vgl. Kap. 3).

⁴⁵Auf dem RPi-Einzelrechner wurden aus Sicherheitsgründen Username und Passwort für den Standard-User geändert.

⁴⁶Auf Schwierigkeiten dieser Einteilung und Lösungsansätze wird in Kap. 3.2.2 eingegangen.

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 Workload der ausgewählten HPC-Benchmarks erwartet werden.

3.1 Zielsetzung

Für das Skalierungsverhalten des Bramble unter der Workload von Linpack und STREAM wird wie in Kap. 2.2 beschrieben die Time to Completion als ausschlaggebende Maßzahl betrachtet. Die ausgewählten Benchmarks Linpack, Whetstone und STREAM werden zunächst auf einem vom RPi-Cluster unabhängigen RPi-Einzelrechner ausgeführt. Dann wird versucht, sie in einer passenden MPI-Implementierung auf dem Bramble lauffähig zu machen. Anschließend erfolgt eine Gegenüberstellung mit den Ergebnissen der Benchmark-Autoren (vgl. Kap. 4.3) und denen tatsächlicher Supercomputer (vgl. Kap. 4.4).

Alle Messungen werden auf 1 – n RPi-Nodes des Bramble mit n=19 RPi-Nodes (vgl. Kap. 2.6.3). Alle Messungen werden zweimal durchgeführt: Mit Netzwerkanschluss der nicht beteiligten RPi-Nodes an `careme` 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 mit und ohne Netzwerkanschluss der nicht beteiligten RPi-Nodes werden anschließend gegenübergestellt (vgl. Kap. 3.3).

Der Schwerpunkt der Untersuchung liegt auf dem Bramble. Da sich seine Systemarchitektur wie in Kap. 2.6 erläutert stark von einem RPi-Einzelrechner unterscheidet und unterschiedliche Implementierungen der Benchmarks verwendet werden, können die Ergebnisse des RPi-Einzelrechner lediglich als grobe Richtlinie dienen.

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 Time to Completion wird pro Benchmark durch zwei zeitabhängige Ausgabeparameter gemessen. Drei Versuchsszenarien kommen zur Anwendung: RPi-Einzelrechner, Bramble mit 19–1 RPi-Nodes aktiv/19 powered, Bramble mit 19–1 RPi-Nodes aktiv/19–1 RPi-Nodes powered.

Für den Versuchsaufbau sind daher folgende Aspekte von Bedeutung: Mögliche Modifikationen der RPi-Nodes (Hardware, OS-Konfiguration), Zeitsynchronisation der RPi-Nodes, Skalierung der Messung auf 19–n RPi-Nodes, automatisierte Durchführung der Messung auf 19–n RPi-Nodes sowie Einlesen der Messwerte in eine geeignete Datenstruktur.

3.2.1 Versuchsaufbau Ri-Einzelrechner

Für die meisten User eines RPi-Einzelrechners stellt sich nach Auswahl, Installation und Konfiguration des Betriebssystems die Frage nach Swap Space und Übertakten. Beides liegt auf der Hand, da das Modell B des RPi nur über 512 MB Arbeitsspeicher verfügt. Die CPU-Leistung ist mit 700 MHz ebenfalls eher niedrig.

Im Praxisbetrieb zeigte sich, dass ein Übertakten der CPU auf bis auf 1 GHz gefahrlos möglich ist. Beim hier verwendeten Testgerät wurde darauf verzichtet, um die Vergleichbarkeit mit den Versuchsergebnissen des Bramble nicht zu gefährden¹.

Das gewählte Betriebssystem Raspbian verwendet standardmäßig eine Swap-Datei `/var/swap`, die den Adressraum des Arbeitsspeichers bei Überlast erweitert. Hierbei gibt es zwei Probleme: Erstens können ständige Schreibzugriffe auf Dauer die SD-Karte beschädigen², zweitens sind Schreibzugriffe auf die SD-Karte sehr langsam, was die Performance des Systems bei hoher Arbeitsspeicherlast beeinträchtigen kann. Daher wurde auf die Swap-Datei verzichtet³.

Die Zeitsynchronisation des RPi erfolgt beim Booten durch Abgleich mit einem NTP-Server. Da er nicht mit weiteren Nodes oder einem zentralen Server synchronisiert werden muss, erübrigt sich hier eine weitere Zeitsynchronisation. Skalierung der Messung und automatisierte Durchführung sind ebenfalls nur für den Bramble von Bedeutung. Auf das Einlesen der Werte in die Datenbank des Bramble wurde auf Grund des unterschiedlichen Versuchsaufbaus ebenfalls verzichtet.

3.2.2 Versuchsaufbau Bramble

Das folgende 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. Der Bramble wird als „Black Box“ betrachtet, d.h. es werden keine Messwerte der einzelnen RPi-Nodes, sondern des Clusters als Gesamtsystem betrachtet.

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 worden⁴. Im Praxisbetrieb zeigten sich

¹Es erscheint wenig zielführend, die Komponente zu tunen, deren Performance man durch Benchmarking evaluieren möchte. Für eine spätere Untersuchung ist dies jedoch nicht ausgeschlossen. Z.B. wäre es interessant zu ermitteln, ob man den relativ hohen Stromverbrauch des Bramble bei Niedriglast (vgl. [Kli13]) durch Untertakten der einzelnen CPUs senken kann. Ergebnisse für Linpack 100 und Whetstone bei Übertakten der CPU auf 1 MHz wurden bereits veröffentlicht (vgl. <http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm>).

²Vgl. [Pow12].

³Sie kann mit dem Befehl `sudo update-rc.d dphys-swapfile remove` deaktiviert werden. Eine weitere Möglichkeit des Swapping ist die Verwendung von zRAM, sodass ein Teil des Arbeitsspeichers komprimiert und als Swap Space genutzt wird (vgl. [Pow12]). Die Allokierung einer auf Unix-Systemen üblicherweise genutzten Swap-Partition ist nicht sinnvoll, da auch hierdurch die Lebensdauer der SD-Karte durch häufige Schreibzugriffe verkürzt wird.

⁴Vgl. [Kli13].

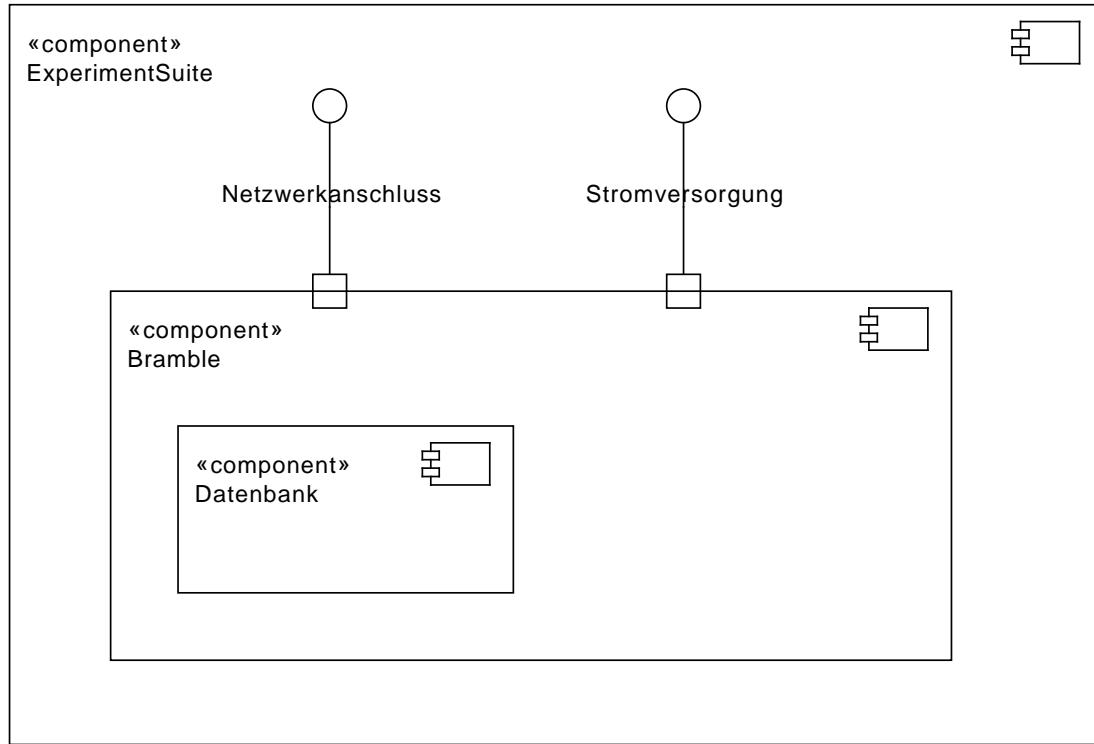


Abbildung 3.1: Komponentendiagramm des Versuchsaufbaus.

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.

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 Einsticken 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.

3 Versuchsaufbau und -ablauf

Dieses Problem ließ sich 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

Ein wichtiger Aspekt bei der parallelen Ausführung eines Programms auf mehreren Rechnerkernen ist die exakte Zeitsynchronisation. Auf dem Bramble wird die Zeitsynchronisation der RPi-Nodes durch einen auf `careme` installierten OpenNTP-Server realisiert. Die einzelnen RPi-Nodes synchronisieren sich gegen diesen Server⁵. Die Zeitsynchronisation der RPi-Nodes ist somit auch für die verteilte Ausführung der Benchmarks gewährleistet.

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 prinzipiell 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.

⁵Vgl. [Kli13].

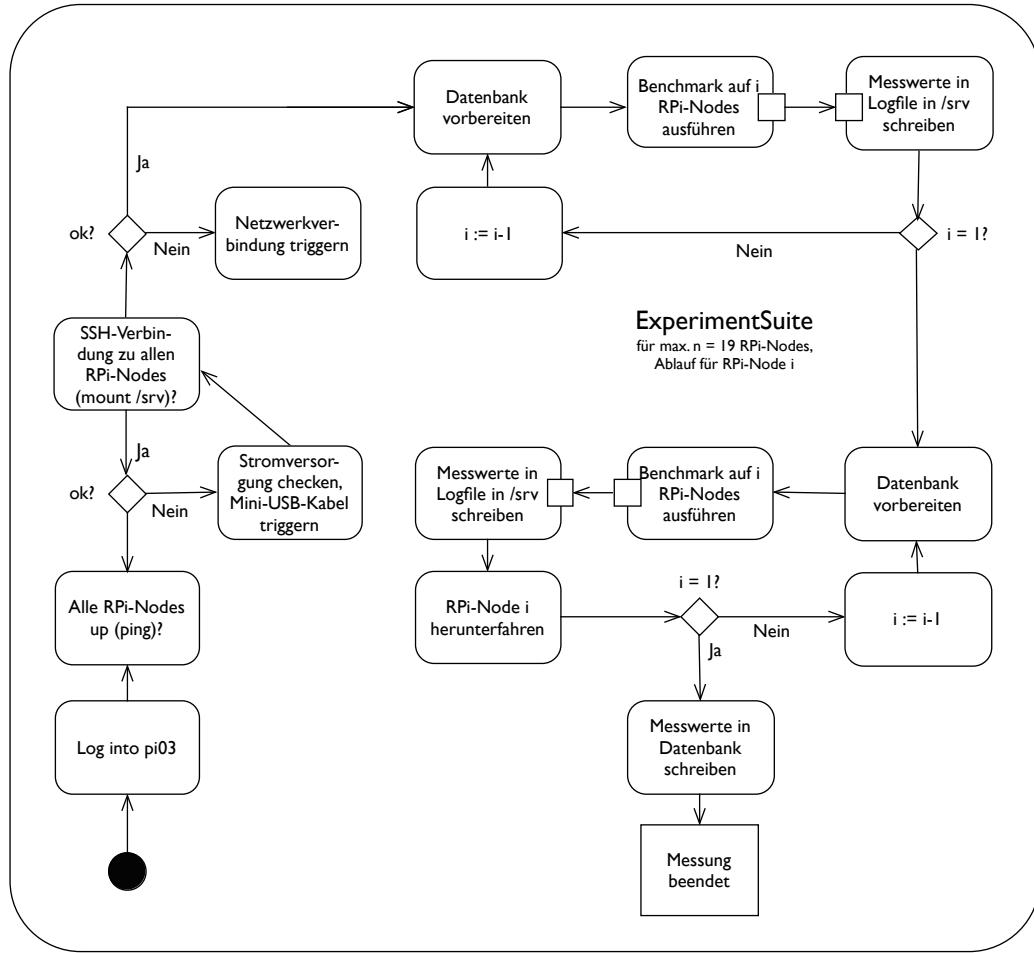


Abbildung 3.2: Aktivitätsdiagramm der ExperimentSuite.

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.

3 Versuchsaufbau und -ablauf

- iii. Shutdown von RPi-Node n.
 - iv. Loggen der Ergebnisdaten.
- 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 zweckmäßigsterweise 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 (vgl. 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:**
 - COPY: 274.4. MB/s
 - SCALE: 209.3 MB/s
 - ADD: 287.2 MB/s
 - TRIAD: 271.1 MB/s
- **Ausführungszeit:**
 - COPY: 0.586838 s
 - SCALE: 0.766437 s
 - ADD: 0.838107 s
 - 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.

3 Versuchsaufbau und -ablauf

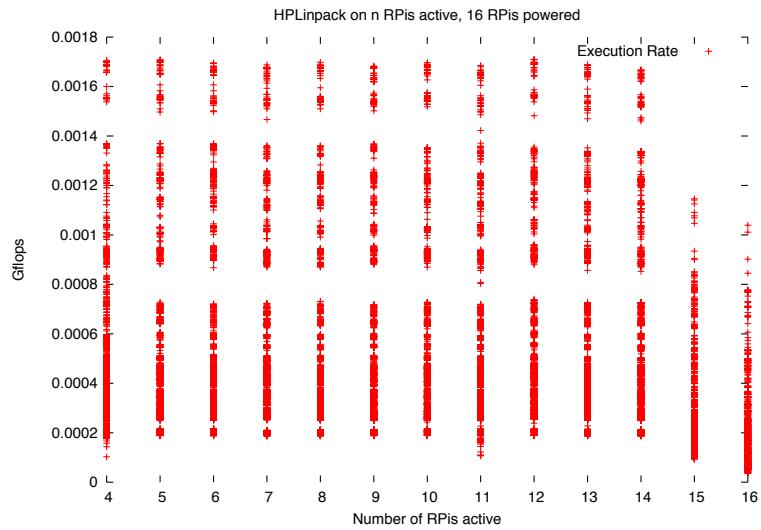


Abbildung 3.3: Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

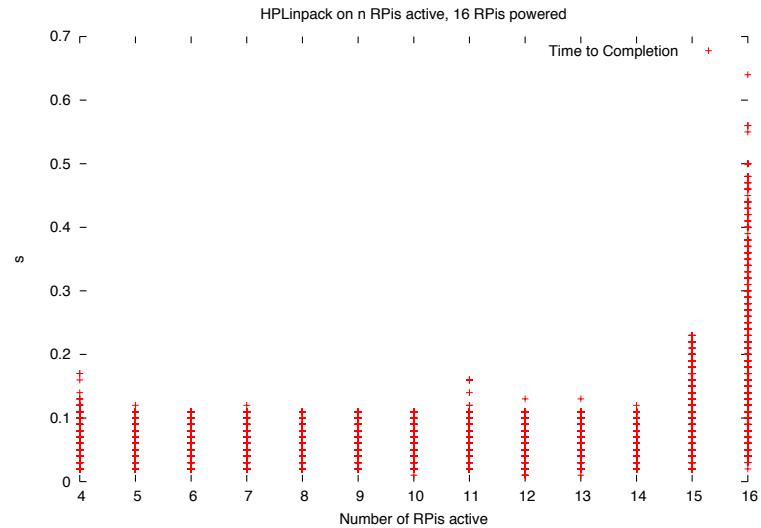


Abbildung 3.4: Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

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.

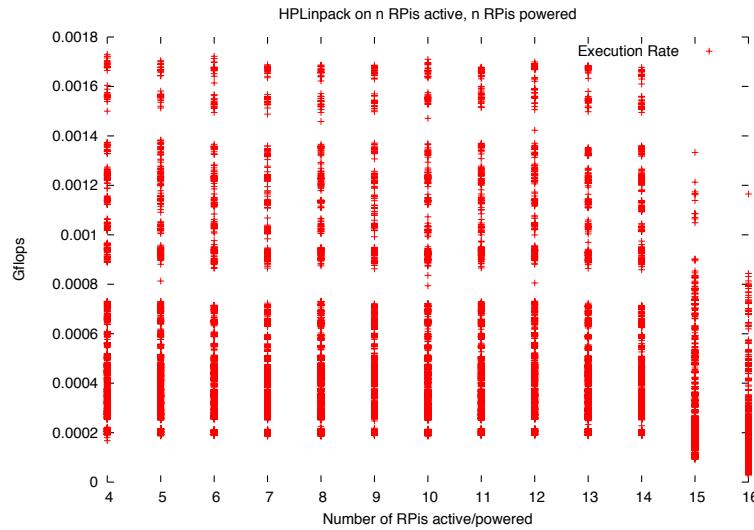


Abbildung 3.5: Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.

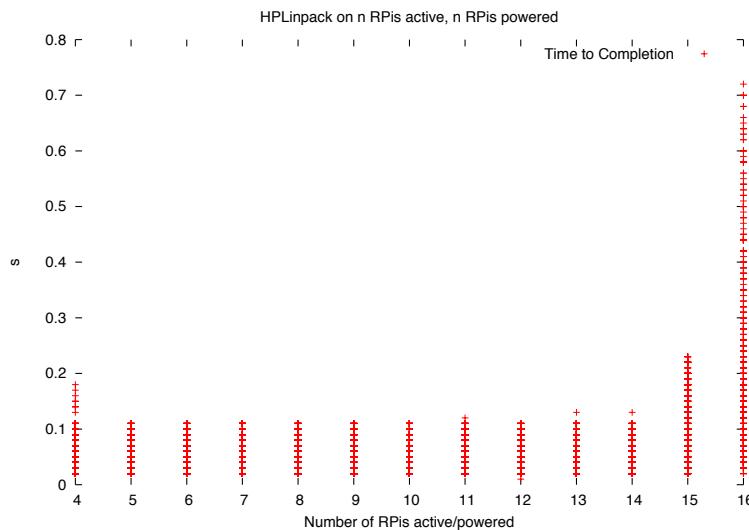


Abbildung 3.6: Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.

3 Versuchsaufbau und -ablauf

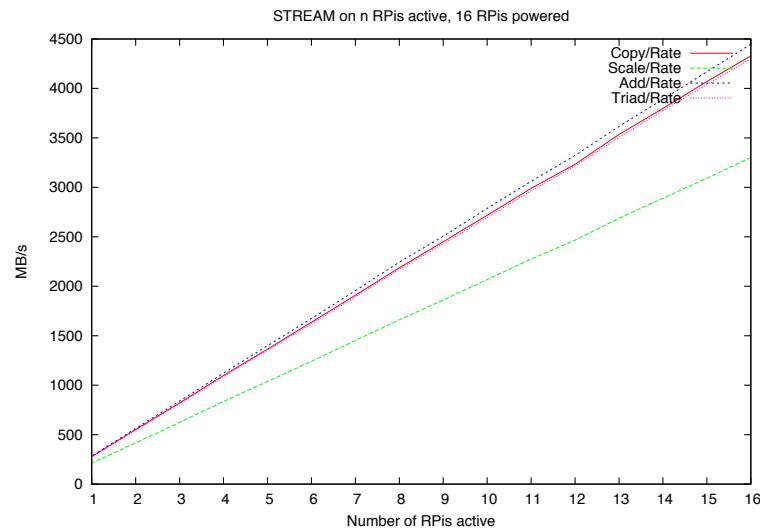


Abbildung 3.7: Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

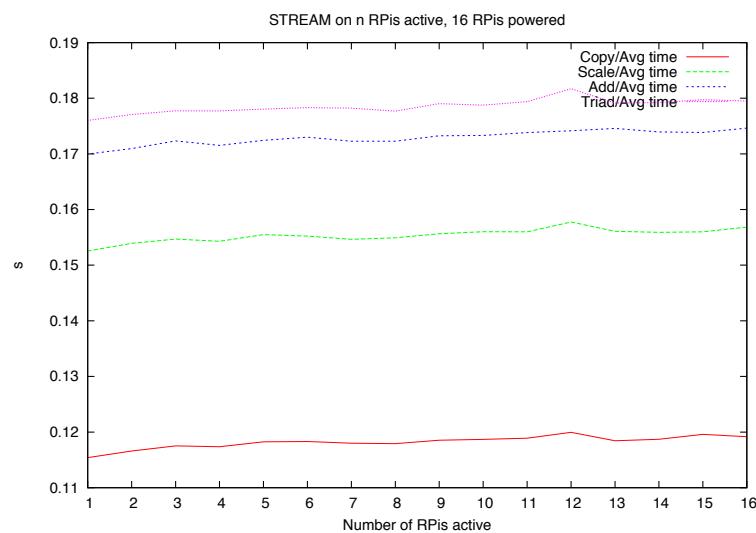


Abbildung 3.8: Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.

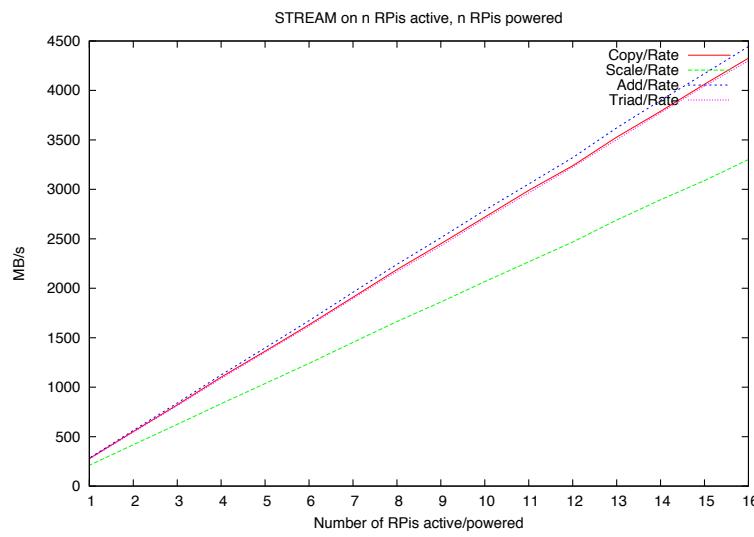


Abbildung 3.9: Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.

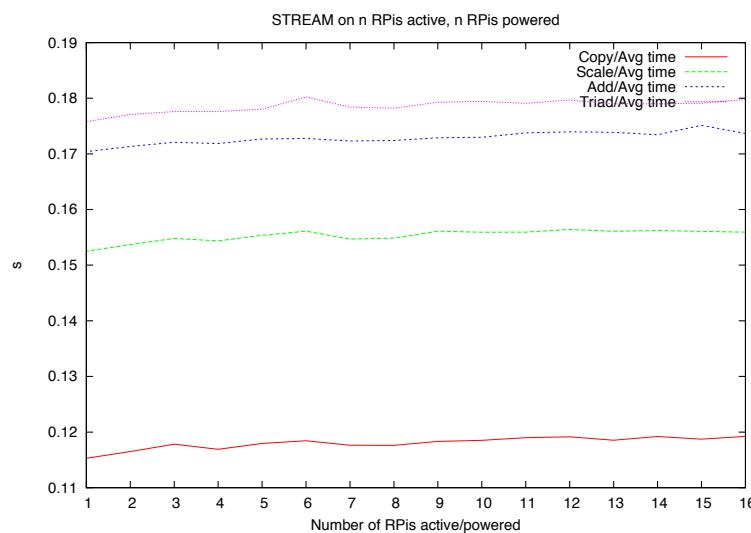


Abbildung 3.10: Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.

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:

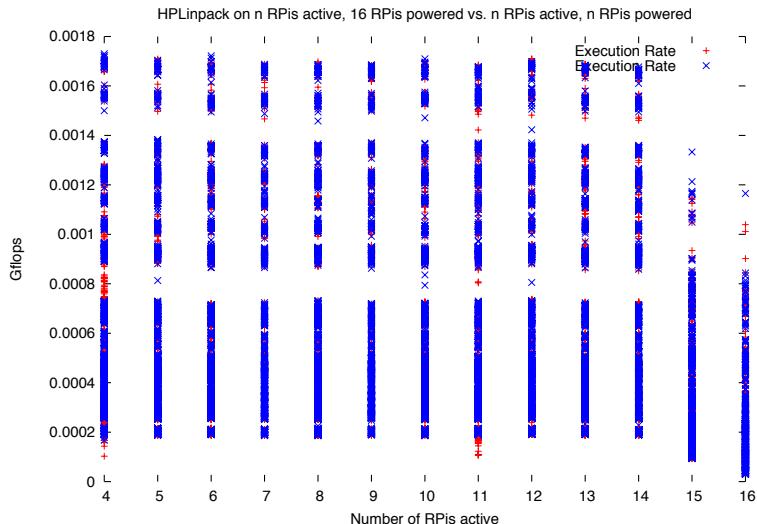


Abbildung 4.1: Ausführungsrate in GFLOPS für HPLinpack mit allen RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

4 Interpretation

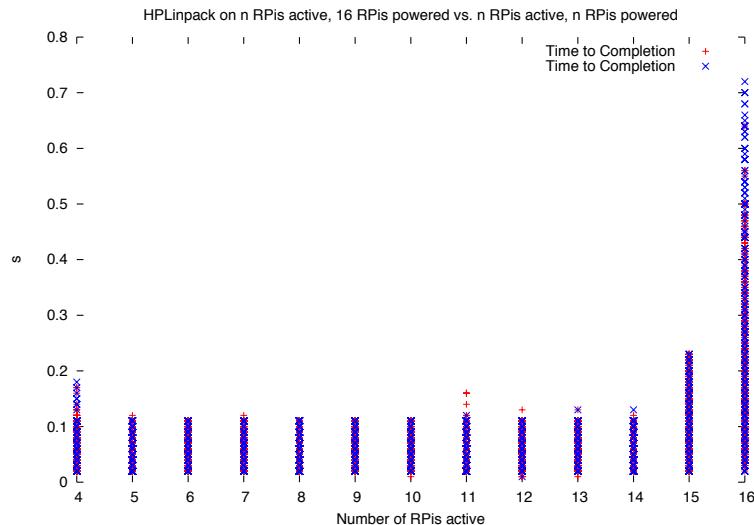


Abbildung 4.2: Ausführungszeit in s für HPLinpack alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

Die Diagramme lassen vermuten, dass keine signifikanten 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 Ausführungszeit 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
Ausführungszeit/Min:	0.01 s	0.01 s
Ausführungszeit/Max:	0.64 s	0.72 s
Ausführungszeit/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 minimalen, 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 minimale Ausführungsrate sinkt um $6.40E-06 = 0.0000064$ GFLOPS, die durchschnittliche Ausführungsrate um $5.00E-06 = 0.0000050$ GFLOPS und die maximale Ausführungszeit steigt um 0.08 s.

4.2 STREAM/Bramble

Noch deutlicher zeigt sich dieser Effekt bei STREAM:

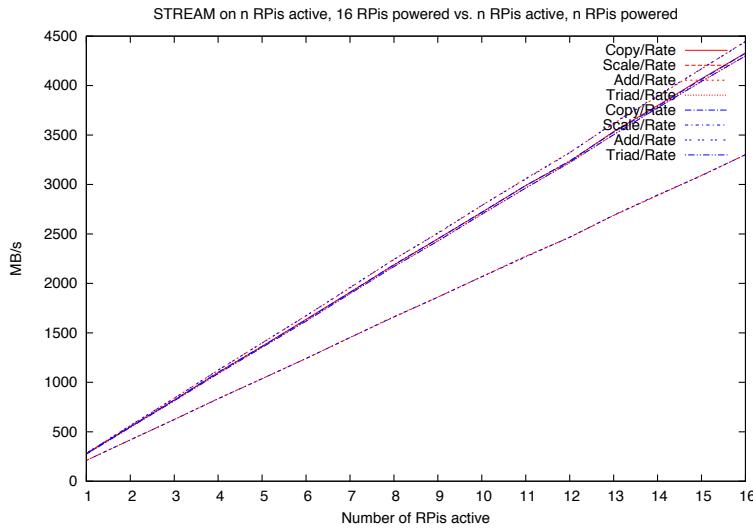


Abbildung 4.4: Ausführungsrate in MB/s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

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 Ausführungszeit. Die folgenden Tabellen 4.6 und 4.7 mit der Gegenüberstellung der maximalen, minimalen und durchschnittlichen Ausführungsrate und Ausführungszeit 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 Ausführungszeit unterscheiden sich beide Runs in fast allen Werten geringfügig, jedoch nur in einer Funktion (minimale Ausführungszeit für ADD) in mehr als vier Nachkommastellen. Wenn man in Betracht zieht, dass HPLinpack für die Ausführungszeit 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 Ausführungszeit 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 Ausführungszeit der STREAM-Module hat.

4 Interpretation

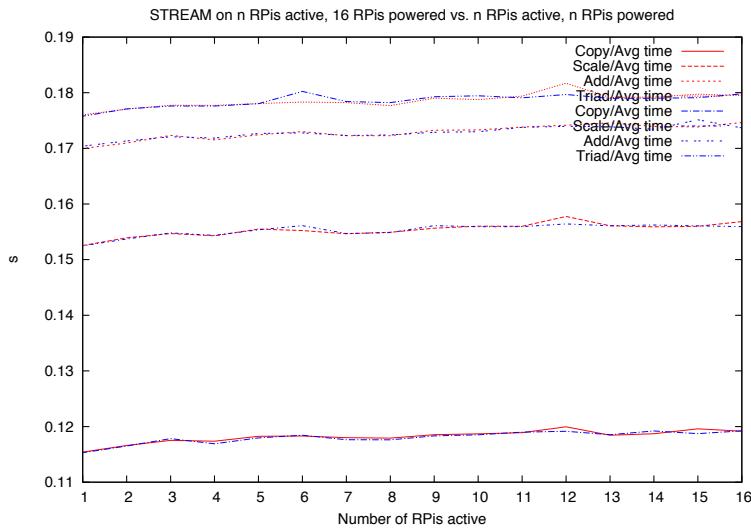


Abbildung 4.5: Ausführungszeit in s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.

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-Autoren¹ gegenüber.²

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

²Soweit ermittelbar, vgl. Kap. 2.4.3. Mit „Linpack“ ist HPLinpack für den Bramble und Linpack 100 für die RPi-Einzelrechner gemeint. Ausführungsrate und Ausführungsdauer auf dem Bramble sind als Mittelwerte angegeben. Die Ergebniswerte wurden jeweils auf die geringste Genauigkeit der vorhandenen Messdaten gerundet.

	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 2013³ 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 Ausführungszeit 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-

³Vgl. <http://www.top500.org/list/2013/11>.

4 Interpretation

	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
Ausführungszeit:	–	–	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

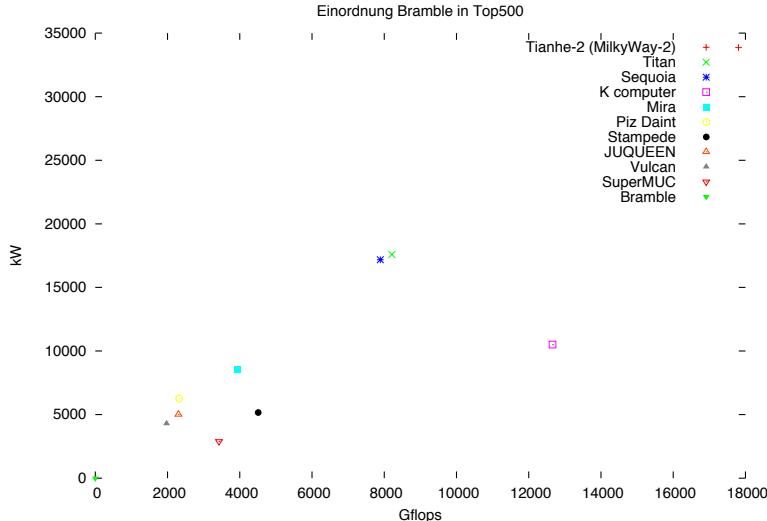


Abbildung 4.11: Einordnung des Bramble in Top500 Nov. 2013.

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-Nachricht, 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 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 1¹.

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 integriert 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 eingetreten² 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 engt, sodass Ziehen und erneutes Einsticken 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

¹Quelle: <http://www.raspberrypi.org/forum/viewtopic.php?f=31&t=44080>.

²Vgl. auch http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html.

5 Zusammenfassung und Ausblick

ist und eine unterbrochene Netzwerkverbindung nur durch Triggern des Netzwerkabzweigs 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 schaffen³. Auch die Platzierung der RPi-Nodes in einem aufrecht stehenden Rack statt eines liegenden Metallgehäuses wäre von Vorteil⁴, 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 hat⁵, hat die Raspberry Pi Foundation einen entsprechenden Wettbewerb ausgeschrieben⁶. Die Ergebnisse und neuen Einsatzmöglichkeiten der bisher kaum anprogrammierbaren GPU, die deutlich leistungsfähiger ist als die hier schwerpunktmäig untersuchte CPU, dürfen mit Spannung erwartet werden.

³Vgl. z.B. <http://raspi.tv/2012/making-a-reset-switch-for-your-rev-2-raspberry-pi>.

⁴Vgl. [Kie13] und [CCB⁺13].

⁵Vgl. <http://blog.broadcom.com/chip-design/android-for-all-broadcom-gives-developers-keys-discretionary{-}{-}{-}to-the-videoocore-kingdom/>.

⁶Vgl. <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'
```

6 Anhang

```
# 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)

```

6 Anhang

```
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 on 1 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
mpieexec -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]}
```

6.2 Ergebnisse der ausgewählten Benchmarks auf dem RPi-Einzelrechner

```
# 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++'
```

6 Anhang

Optimisation Opt 3 32 Bit

norm resid	resid	machepr	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

6.2 Ergebnisse der ausgewählten Benchmarks auf dem RPi-Einzelrechner

Average 45.92

#####

```
From File /proc/cpuinfo
Processor : ARMv6-compatible processor rev 7 (v61)
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 (v61)
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	1.00000000000000000000		690.831	0.039
N4 fixed point	12.000000000000000000		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	3.000000000000000000		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.
```

6.2 Ergebnisse der ausgewählten Benchmarks auf dem RPi-Einzelrechner

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

2.1 Ein Raspberry Pi Modell B (Quelle: [BCH ⁺ 12]).	9
2.2 Der hier verwendete Bramble.	11
3.1 Komponentendiagramm des Versuchsaufbaus.	15
3.2 Aktivitätsdiagramm der ExperimentSuite.	17
3.3 Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.	20
3.4 Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/16 RPi-Nodes powered.	20
3.5 Ausführungsrate in GFLOPS für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.	21
3.6 Ausführungszeit in s für HPLinpack auf n RPi-Nodes aktiv/n RPi-Nodes powered.	21
3.7 Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.	22
3.8 Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/16 RPi-Nodes powered.	22
3.9 Ausführungsrate in MB/s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.	23
3.10 Ausführungszeit in s für STREAM auf n RPi-Nodes aktiv/n RPi-Nodes powered.	23
4.1 Ausführungsrate in GFLOPS für HPLinpack mit allen RPi-Nodes powered vs. nur aktive RPi-Nodes powered.	25
4.2 Ausführungszeit in s für HPLinpack alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.	26
4.3 Gegenüberstellung HPLinpack alle RPis powered vs. nur aktive RPis powered.	26
4.4 Ausführungsrate in MB/s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.	27
4.5 Ausführungszeit in s für STREAM alle RPi-Nodes powered vs. nur aktive RPi-Nodes powered.	28
4.6 Gegenüberstellung STREAM (Ausführungsrate) alle RPis powered vs. nur aktive RPis powered.	29
4.7 Gegenüberstellung STREAM (Ausführungszeit) alle RPis powered vs. nur aktive RPis powered.	30
4.8 Linpack auf Bramble und RPi-Einzelrechnern.	30
4.9 Whetstone auf RPi-Einzelrechnern.	30
4.10 STREAM auf Bramble und RPi-Einzelrechnern.	30
4.11 Einordnung des Bramble in Top500 Nov. 2013.	31

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>
- [BCH⁺12] BEALE, Clive ; CROSTON, Ben ; HAGUE, Andrew u.a.: *The Raspberry Pi Education Manual*, Dezember 2012. – http://downloads.raspberrypi.org/Raspberry_Pi_Education_Manual.pdf
- [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://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>
- [Kie13] KIEPERT, Joshua: *Creating a Raspberry Pi-Based Beowulf 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>
- [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

Literaturverzeichnis

- [Ou13] OU, Jun: *Pi and the Sky*. 2013. – <https://www.duo.uio.no/bitstream/10852/37445/4/0u-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
- [Wei90] WEICKER, Reinhold: An Overview of Common Benchmarks. In: *Computer* (1990), S. 65–75. – http://dlojewski.dl.fupic.de/download/Diplomarbeit/Andere_Dok/Dhry_Whet/OverviewOfCommonBenchmarks.pdf