



Bachelorarbeit

Untersuchung des
Skalierungsverhaltens eines
Raspberry Pi-Clusters
unter Verwendung von
HPC-Benchmarks

Judith Greif





Bachelorarbeit

Untersuchung des
Skalierungsverhaltens eines
Raspberry Pi-Clusters
unter Verwendung von
HPC-Benchmarks

Judith Greif

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Christian Straube



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

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





Abstract

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

Tritt der Raspberry Pi in die Welt der Supercomputer ein, muss er sich auch mit ihren Spielregeln messen lassen. Die vorliegende Arbeit untersucht das Skalierungsverhalten eines Raspberry Pi-Clusters mit 20 Raspberry Pi-Knoten bei der parallelen Ausführung der HPC-Benchmarks HPL und STREAM. Als Parameter werden Performance (Ausführungsrate und Ausführungsdauer) und Energieverbrauch betrachtet.

Die Ergebnisse zeigen ein kohärentes und erwartungsgemäßes Skalierungsverhalten des Clusters bezüglich Stromverbrauch, Performance bei der Ausführung von HPL und Performance bei der Ausführung von STREAM auf bis zu 17 Raspberry Pi-Knoten. Der Versuchsaufbau ist weiterer Schritt auf dem Weg der Ausführung von parallelen und HPC-Anwendungen auf dem Raspberry Pi und erweitert dessen Einsatzmöglichkeiten.



Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen und Begriffsbildung	3
2.1	Definition: <i>Benchmarking</i>	3
2.2	Definition: <i>Performance</i>	4
2.3	Definition: <i>Energieverbrauch</i>	5
2.4	Benchmarks	5
2.4.1	Linpack	5
2.4.2	Whetstone	7
2.4.3	STREAM	8
2.5	Spezifikation des RPi Modell B	9
2.5.1	Physischer Aufbau	9
2.5.2	Betriebssystem	9
2.6	Raspberry Pi-Cluster als Testumfeld	9
2.6.1	Physischer Aufbau	10
2.6.2	Betriebssystem und Dateisystem	10
2.6.3	Implikationen für den Versuchsaufbau	11
3	Versuchsaufbau und -ablauf	13
3.1	Zielsetzung, Aufbau und Art der Messung	13
3.1.1	Versuchsaufbau Ri-Einzelrechner	13
3.1.2	Versuchsaufbau Bramble	14
3.2	Strommessung	20
3.3	Fehlerbehebung	22
3.4	Ergebnisse	23
3.4.1	HPL: Performance	24
3.4.2	STREAM: Performance	25
3.4.3	Stromverbrauch	26
4	Interpretation	33
4.1	HPL: Performance	33
4.2	STREAM: Performance	33
4.3	Stromverbrauch	37
5	Zusammenfassung und Ausblick	39
	Abbildungsverzeichnis	41
	Literaturverzeichnis	43





1 Einleitung

Seit Beginn seiner Entwicklung im Jahr 2009 boomt der Minicomputer Raspberry Pi (im Folgenden als *RPi* bezeichnet). Er erhielt z.B. den Designpreis INDEX: award 2013

(vgl. <http://designtoimprovelife.dk/category/s15-award2013>), wurde als Innovation des Jahres bei den T3 Gadget Awards 2012 ausgezeichnet (vgl. <http://www.t3.com/news/t3-gadget-awards-2012-award-winners>) und zum Gadget of the Year 2012 des Linux Journal gewählt (vgl. <http://www.linuxjournal.com/slideshow/readers-choice-2012>). Im Februar dieses Jahres war das Modell B über 2,5 Millionen Mal verkauft worden.



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.

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



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. Sie können nur herangezogen werden, wenn hierfür geeignete Implementierungen existieren.

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

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

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

1 Einleitung

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

Um den Bezugsrahmen zu verdeutlichen, werden in Kapitel 2 zunächst grundlegende Definitionen geklärt. Insbesondere werden die ausgewählten Benchmarks vorgestellt (vgl. Kap. 2.4) und die Spezifikationen von RPi (vgl. Kap. 2.5) und RPi-Cluster erläutert (vgl. Kap. 2.6). Versuchsaufbau und -ablauf werden mit Blick auf Auswahl und Anpassung der RPi-spezifischen Parameter im folgenden Kapitel dargestellt (vgl. Kap. 3). Schließlich werden die Messergebnisse dargestellt (vgl. Kap. 3.4) und interpretiert (vgl. Kap. 4). Den Abschluss bilden eine Zusammenfassung der Untersuchungsergebnisse und ein Ausblick (vgl. Kap. 5).



2 Grundlagen und Begriffsbildung

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

Bei näherer Betrachtung erscheint es schwierig, sich einen Überblick über die existierenden Benchmarks zu verschaffen. Vieles, was von den Anwendern als „Benchmark“ bezeichnet wird, stellt sich als selbst geschriebene Routine heraus, mit der z.B. die Performance der Grafikkarte getestet werden soll. Auf solche Routinen darf sich die Untersuchung nicht stützen. Im Folgenden wird daher auf grundlegende Begriffe der Untersuchung eingegangen. Anschließend werden die verwendeten Benchmarks (vgl. 2.4) und ihre Ausführung auf dem RPi erläutert (vgl. 2.6.3).



2.1 Definition: Benchmarking

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

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

Das *Computer Lexikon 2012* kennt folgende Definition:

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

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

Auswahl der Implementierungen und Gestaltung der Testumgebung zu berücksichtigen (vgl. Kap. 2.5, 2.6 und 2.6.3).

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



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

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

2.2 Definition: Performance

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

The performance of a computer is a complicated issue, a function of many 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 folgende Aspekte:

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

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

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

Mit *Performance* wird **daher hier** die **Rechenleistung** in diesem abgegrenzten Sinn bezeichnet, d.h. die Datenverarbeitungsgeschwindigkeit eines spezifischen Systems, nicht dessen Leistung im physikalischen Sinn. Dabei werden die **Ausführungsrate** und die **Ausführungszeit** als maßgebliche Zahlen betrachtet.




2.3 Definition: Energieverbrauch

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

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

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

2.4 Benchmarks

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

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

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

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

2.4.1 Linpack

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

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

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

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

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

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

Dort findet sich auch eine vertiefte Darstellung des Linpack-Benchmarks.

Funktionsweise

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

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

Linpack auf dem Raspberry Pi

Bereits kurz nach Verkaufsstart des RPi wurden Implementierungen von Linpack für den RPi bereit gestellt und Ergebnisse von Testläufen im Internet veröffentlicht. Somit liegen bereits Vergleichswerte für einen RPi-Einzelrechner vor. Auch Implementierungen von Linpack für verteilte Systeme unabhängig von den Top500-Rankings sind im Umlauf. Der hier verwendete Quellcode findet sich unter <http://www.netlib.org/benchmark/hpl1>.



2.4.2 Whetstone

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

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

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

Funktionsweise

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

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

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

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

Whetstone auf dem Raspberry Pi

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

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

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

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

2.4.3 STREAM

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

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

Funktionsweise

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

STREAM durchläuft bei seiner Ausführung vier Module (Copy, Scale, Add und Triad) und liefert für jedes Modul die durchschnittliche, maximale und minimale Ausführungszeit sowie die Ausführungsrate. Jedes Modul wird mehrfach ausgeführt, um verlässliche Mittelwerte zu erhalten. Folgende Berechnungen werden dabei durchgeführt:

1. Copy: $a(i) = b(i)$
2. Scale: $a(i) = q * b(i)$
3. Add: $a(i) = b(i) + c(i)$
4. Triad: $a(i) = b(i) + q * c(i)$

Zwischen dem Programmcode der Module bestehen Abhängigkeiten, um starke Optimierungen durch den Compiler zu verhindern (vgl. [McC05]). Quellcode und Ergebnisse für STREAM auf verschiedenen Systemen finden sich unter <http://www.cs.virginia.edu/stream>.

STREAM auf dem Raspberry Pi

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

2.5 Spezifikation des RPi Modell B

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

2.5.1 Physischer Aufbau

Der RPi ist mit einem Broadcom BCM2835 als System on a Chip ausgestattet. Es enthält CPU (ein ARM1176JZFS-Prozessor) und GPU (ein Broadcom VideoCore IV-Koprozessor, vgl. [Lan12]). Auffällig ist, dass die CPU des RPi (ein ARM-Chip, wie er auch in Mobiltelefonen zur Anwendung kommt), relativ schwach ist im Vergleich zur GPU. Diese unterstützt Full HD-Auflösung und das Hardware-beschleunigte Rendern verschiedener Videoformate. Das Modell B verfügt über 512 MB SDRAM. Dieser kann nicht erweitert werden.

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

2.5.2 Betriebssystem

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

2.6 Raspberry Pi-Cluster als Testumfeld

Seit einiger Zeit lässt sich die Tendenz beobachten, eine größere Anzahl von Raspberry Pis zu einem Cluster zu koppeln (weitere Projekte werden in [CCB⁺13], [Kie13], [Bal12] und [Ou13] dargestellt). Die hier verwendete Architektur wird im Folgenden kurz vorgestellt. Eine detaillierte Darstellung liefert [Kli13].

2.6.1 Physischer Aufbau

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

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

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

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

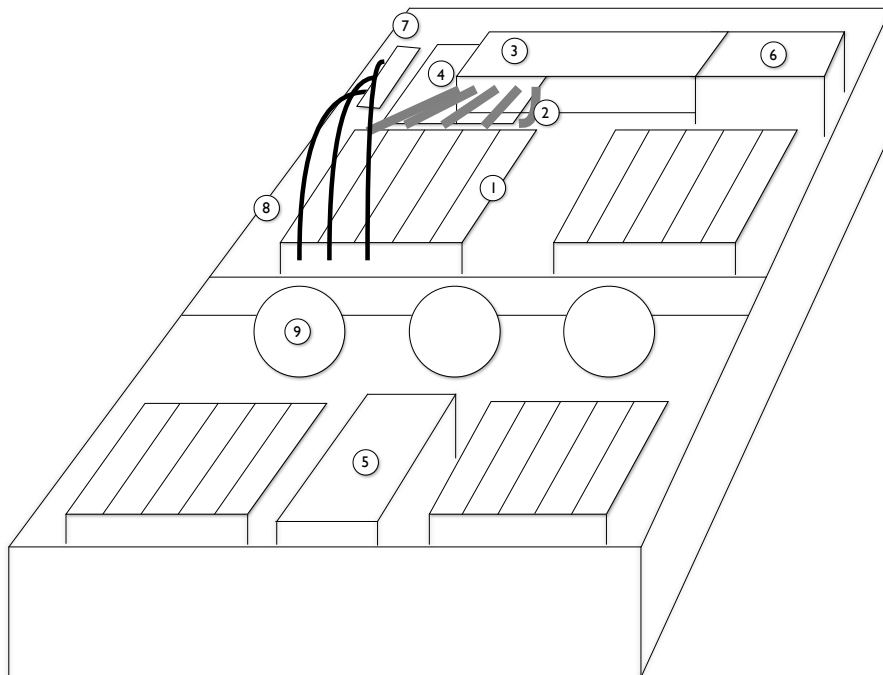


Abbildung 2.1: Schematischer Aufbau des hier verwendeten Bramble.

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

2.6.2 Betriebssystem und Dateisystem

Auf den RPi-Knoten ist das oben genannte Betriebssystem Raspbian installiert (vgl. Kap. 2.5.2), auf **careme** eine Standard-Debian-Version. Ein Charakteristikum eines Beowulf-Clusters ist, dass es kein Shared Memory-Interface und keine Cache-Kohärenz gibt. Somit stellt sich die Frage nach der Zeit- und Datensynchronisation der RPi-Knoten und des Servers sowie den Möglichkeiten des gemeinsamen Speicherzugriffs.

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

2.6.3 Implikationen für den Versuchsaufbau

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

Netzzugriff

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

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



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

Ausführung verteilter Anwendungen

Der Versuchsaufbau sieht vor, verteilte Anwendungen auf unterschiedlichen Anzahlen von RPi-Knoten auszuführen. Zur verteilten Berechnung von Anwendungen auf mehreren CPUs steht auf dem Bramble die MPI-Implementierung MPICH in der Version 3.0.4 zur Verfügung. Um die CPUs bzw. RPi-Knoten und die Anzahl der darauf auszuführenden Prozesse zu spezifizieren, muss ein entsprechendes Machinefile erstellt und im gescherten Verzeichnis abgelegt werden. Dieses wird der Ausführung von MPICH mit **mpiexec** als Parameter übergeben (vgl. Kap. 3.1.2).



3 Versuchsaufbau und -ablauf

Das folgende Kapitel beschreibt Ziele der Messung, Versuchsaufbau und -durchführung. Anschließend werden die erzielten Messergebnisse dargestellt.

3.1 Zielsetzung, Aufbau und Art der Messung

Es stellen sich zwei Fragen: Welcher Art ist die Messung und welche Voraussetzungen müssen hierfür erfüllt sein?

Ziel der Untersuchung sind Erkenntnisse über das Skalierungsverhalten des Bramble unter der Arbeitslast von HPL und STREAM. Dazu werden wie in Kap. 2 beschrieben zwei Maßzahlen betrachtet: Performance und Energieverbrauch.

Die Benchmarks werden auf $n - 4$ RPi-Knoten des Bramble ausgeführt. Maximal 19 RPi-Knoten werden dafür herangezogen (vgl. Kap. 2.6.3). Alle Messungen werden zweimal durchgeführt: Mit Stromanschluss der nicht beteiligten RPi-Knoten (Messreihe 1) und ohne (Messreihe 2). Für beide Benchmarks werden zwei Ergebnisparameter betrachtet: Ausführungsrate in GFLOPs und Ausführungszeit in s für HPL, Ausführungsrate in MB/s und durchschnittliche Ausführungszeit in s für STREAM. Anschließend werden die Ergebnisse der Messreihen gegenübergestellt (vgl. Kap. 3.4).

Für den Versuchsaufbau sind demnach folgende Aspekte von Bedeutung: Modifikation und Zeitsynchronisation der RPi-Knoten, Skalierung der Messung auf $n - 4$ RPi-Knoten, automatisierte Durchführung der Messung, Einlesen der Messwerte in eine geeignete Datenstruktur und Strommessung.

3.1.1 Versuchsaufbau Ri-Einzelrechner

Viele Nutzer stellen sich nach Inbetriebnahme eines RPi-Einzelrechners die Frage nach Swap-Speicher und Übertakten (vgl. [Pow12]). Beides liegt nahe, da das Modell B nur über 512 MB Arbeitsspeicher verfügt. Die CPU-Taktfrequenz beträgt 700 MHz.

Im Praxisbetrieb wurde gezeigt, dass ein Übertakten der CPU auf bis zu 1 GHz gefahrlos möglich ist (vgl. z.B. <http://www.raspberrypi.org/introducing-turbo-mode-up-to-50-more-performance-for-free/>). Bei den verwendeten RPi-Knoten wurde davon Abstand genommen. Es erschien wenig zielführend, die Komponente zu manipulieren, deren Performance evaluiert werden soll¹. Für Linpack 100 und Whetstone wurden allerdings bereits Ergebnisse mit auf 1 MHz übertakteter CPU veröffentlicht (vgl. <http://www.roylongbottom.org.uk/Raspberry%20Pi%20Benchmarks.htm>).

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

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

Das Betriebssystem Raspbian legt standardmäßig eine Swap-Datei `/var/swap` auf der SD-Karte an (vgl. <http://raspberrypi.stackexchange.com/questions/70/how-to-set-up-swap-space>). Hierbei zeigen sich Probleme: Erstens können häufige Schreibzugriffe die SD-Karte beschädigen. Zweitens sind Schreibzugriffe auf die SD-Karte langsam, was die Performance des Systems bei hoher Arbeitsspeicherlast beeinträchtigen kann (vgl. [Pow12]).

Das gilt auch für die Allokierung einer Swap-Partition auf der SD-Karte. Diese Möglichkeit spielt daher in der Praxis keine Rolle.

Bei der Verwendung von zRAM wird ein Teil des Arbeitsspeichers komprimiert und als Swap-Speicher genutzt. Hierbei werden keine Zugriffe auf die SD-Karte notwendig (vgl. [Pow12]).

Auf dem `careme` war kein Swap-Speicher allokiert worden (vgl. [Kli13]). Entgegen der Beschreibung bei war nur auf einem RPi-Knoten eine Swap-Datei vorhanden. Um gleiche Testbedingungen auf allen RPi-Knoten zu schaffen, wurde sie deaktiviert.

3.1.2 Versuchsaufbau Bramble

Diagramm 3.1 zeigt den Versuchsaufbau pro Messreihe: Stromversorgung, Netzanschluss und Strommessgerät als physische Komponenten und MySQL-Datenbank als logische Komponente.



Modifikation der RPi-Knoten

Zu Beginn der Untersuchung zeigte sich, dass einige Mini-USB-Kabel zur Stromversorgung der RPis defekt waren. Sie wurden durch funktionsfähige Kabel ersetzt.

Zeitsynchronisation der RPi-Knoten

Der RPi-Einzelrechner besitzt aus Kostengründen keine Systemuhr (vgl. [Sch13]), sondern synchronisiert sich beim Booten gegen einen NTP-Server im Internet. Für die parallele Ausführung eines Programms auf mehreren Rechnerkernen ist die Zeitsynchronisation der RPi-Knoten untereinander essentiell. Auf dem Bramble wird das durch einen OpenNTP-Server auf `careme` realisiert, gegen den sich die RPi-Knoten synchronisieren (vgl. [Kli13]).

Skalierung der Messung auf $n - 4$ RPi-Knoten

Die Ausführung eines Benchmark-Programms mit einer bestimmten Anzahl aktiver und angeschalteter RPi-Knoten wird als *ExperimentSuite* bezeichnet werden. Diagramm 3.2 zeigt, welche Schritte aus Benutzersicht zu ihrer Durchführung erforderlich sind. Messreihe 1 beginnt mit der höchsten Anzahl aktiver RPi-Knoten begonnen und iteriert einmal über alle Knoten. In Messreihe 2 werden nach jedem Iterationsschritt nicht mehr aktive RPi-Knoten abgeschaltet. Jede Ausführung eines Benchmark-Programms läuft prinzipiell gleich ab, während zu Beginn und an Ende spezielle Vorkehrungen zu treffen sind.

Konfiguration der Datenbank

Messergebnisse, Konfigurationen von Benchmarks und ExperimentSuites werden in der MySQL-Datenbank `rpIWerte` auf `careme` abgelegt. Dafür war ein Datenbankschema vorgegeben



Abbildung 3.1: Komponentendiagramm des Versuchsaufbaus.

worden, das die ExperimentSuites in einen größeren Versuchsaufbau integriert. Es wurde während der praktischen Arbeit an die tatsächlichen Erfordernisse angepasst: Für jedes Modul eines Benchmarks wird ein Messparameter definiert, der Unix-Zeitstempel seines Ausführungsendes ermittelt und mit dem jeweiligen Messwert in die Datenbank eingelesen. Für jede ExperimentSuite werden Anfangs- und Endzeitpunkt als Unix-Zeitstempel ermittelt und in die Datenbank eingelesen. Die Konfiguration erfolgt in vier Schritten:

1. Name und Beschreibung des Benchmarks.

2. Beschreibung des Versuchsaufbaus.

3. Konfiguration des Versuchsaufbaus.

4. Verknüpfung von Benchmark-Konfiguration und Versuchsaufbau.





Abbildung 3.2: Aktivitätsdiagramm einer ExperimentSuite.

Diagramm 3.3 visualisiert Schritt 1. Die statischen Konfigurationen für STREAM und HPL werden festgelegt. Dazu wurden zwei Shellskripte `loadGeneratorConfigHpl.sh` und `loadGeneratorConfigStream.sh` erstellt. Sie werden zu Beginn des Versuchsaufbaus einmal ausgeführt. Pro Benchmark muss in drei Tabellen jeweils ein Eintrag erstellt werden: In der Tabelle `LoadGenerator` wird ein Eintrag mit Name und Beschreibung des Benchmarks erstellt. In der Tabelle `ENUM_LoadGeneratorConfigurationKey` wird ein Eintrag mit dem Konfigurationsschlüssel des Benchmarks erstellt. In der Tabelle `LoadGeneratorConfiguration` wird ebenfalls ein Eintrag mit dem Konfigurationsschlüssel erstellt. Die dynamischen Schritte 2–4 werden einmal pro ExperimentSuite ausgeführt: Name, Beschreibung und Konfiguration des Versuchsaufbaus, Verknüpfung von ExperimentSuite und Benchmark-Konfiguration. Sie werden in **Diagramm 3.4** veranschaulicht und sind in die Ausführungsskripte der Benchmarks integriert.

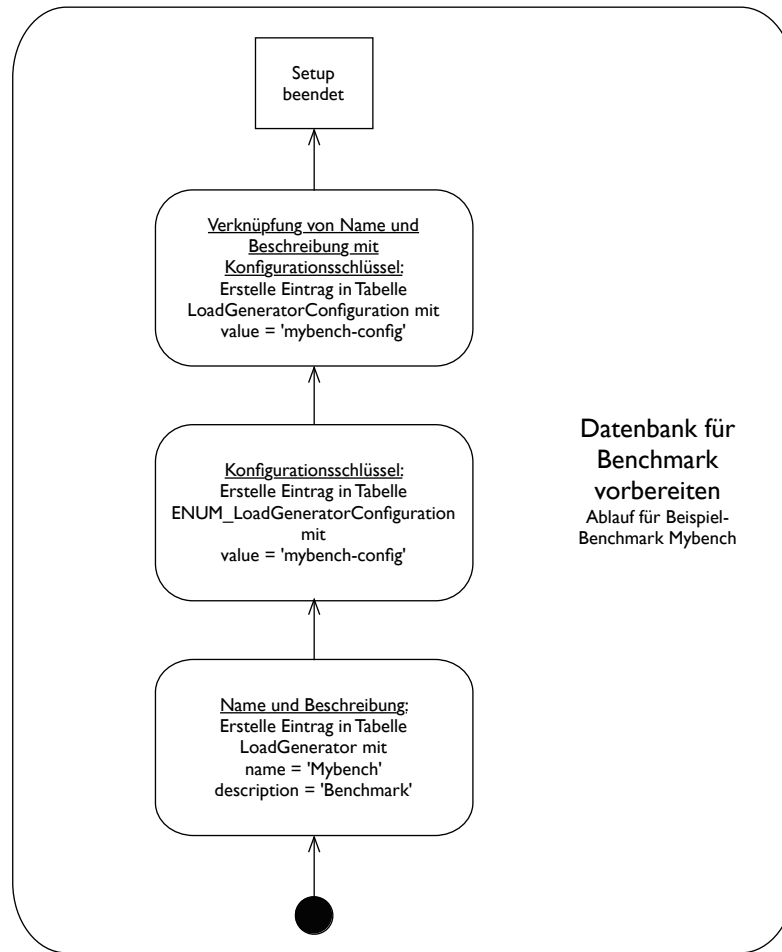


Abbildung 3.3: Aktivitätsdiagramm zur Konfiguration der Datenbank für einen Benchmark.

Automatisierte Durchführung der Messung auf $n - 4$ RPi-Knoten

Die automatisierte Durchführung der Messung erfolgt durch Shellskripte. Sie werden auf dem Berechnungsknoten pi03 ausgeführt und enthalten folgende Schritte:

1. Erstellen eines Machinefile, ggf. Löschen des alten.

Zur parallelen Ausführung eines Programms auf mehreren CPUs durch MPICH wird `mpiexec` mit dem Parameter `-machinefile` aufgerufen. Er verweist auf eine Datei, die die Reihenfolge der zu verwendenden CPUs und die Anzahl der darauf auszuführenden Prozesse spezifiziert. Es dürfen nur so viele Prozesse vergeben werden, wie angeschaltete RPi-Knoten zur Verfügung stehen. Die CPUs müssen in der Reihenfolge angegeben werden, in der die RPi-Knoten heruntergefahren werden. Ein eventuell veraltetes `Machinefile` wird gelöscht.

2. Einhängen des geteilten Verzeichnisses auf allen RPis.

Alle Dateien der ExperimentSuites (Binaries, Bibliotheken etc.) liegen im geteilten

Verzeichnis `/srv` bzw. `/srv/nfs-share`. Daher muss das geteilte Verzeichnis auf jedem verwendeten RPi-Knoten eingehängt sein.

3. Navigation ins Arbeitsverzeichnis.

Alle Shellskripte zur Konfiguration und Ausführung der ExperimentSuites liegen in einem Verzeichnis `/srv/experimentsuite`. Die Ergebnisdateien werden ebenfalls dort abgelegt.

4. Iteration über n ausgewählte Benchmarks:

a) Erstellen von Logdateien.

Für die Ausgabe der Benchmarks werden Logdateien vorbereitet.

b) Iteration über n RPi-Knoten:

i. Datenbank für ExperimentSuite vorbereiten.

Diagramm 3.4 visualisiert Schritt b) i. Für jede Ausführung eines Benchmark-Programms wird ein Eintrag in der Tabelle `ExperimentSuite` mit Granularität, Ziel und Startzeitpunkt erstellt. In der Tabelle `ExperimentSuite-Configuration` werden zwei Einträge mit Anzahl aktiver und angeschalteter RPi-Knoten erstellt. Beide Tabellen werden über die Tabelle `N2M-loadConf3expSuite` verknüpft. Darin wird ein Eintrag mit ID des Benchmarks und ID der ExperimentSuite erstellt.

ii. Parallele Ausführung auf n RPi-Knoten.

Zur parallelen Ausführung benötigt `mpiexec` weitere Parameter: `-n` spezifiziert die Anzahl an parallelen Programmaufrufen, `-wdir` das Arbeitsverzeichnis, in das gewechselt werden soll, und `file` die ausführbare Datei (vgl. <http://www.mpich.org/static/docs/latest/www1/mpiexec.html>). Die parallele Ausführung von HPL auf vier RPi-Knoten erfolgt z.B. mit

```
mpiexec -n 4 -machinefile
/srv/libraries/etc/mpich-3.0.4-shared/machinefile_latest
-wdir /srv/benchmarks/bin/hpl-2.1/messung1
/srv/benchmarks/bin/hpl-2.1/messung1/xhpl >>
results/hpl-2.1_'date +%y%m%d'.txt
```

iii. Ermittlung des Ausführungsendes.

Nach der Ausführung wird der Endzeitpunkt als Unix-Zeitstempel ermittelt. Der entsprechende Eintrag in der Tabelle `ExperimentSuite` wird um diesen Wert ergänzt.

iv. Schreiben der Ergebnisdaten.

Die Ausgabe wird in die vorbereitete Logdatei geschrieben. Falls das Benchmark-Programm keinen Zeitstempel vorsieht (z.B. STREAM), wird dieser ermittelt und als zusätzliche Zeile hinzugefügt. Damit kann später für jeden Messwert ein Zeitstempel in die Datenbank eingelesen werden (vgl. Schritte d und e). Auch zum Abgleich der Messergebnisse von Benchmarks und Strommessgerät ist ein Zeitstempel erforderlich (vgl. Kap. 3.2).

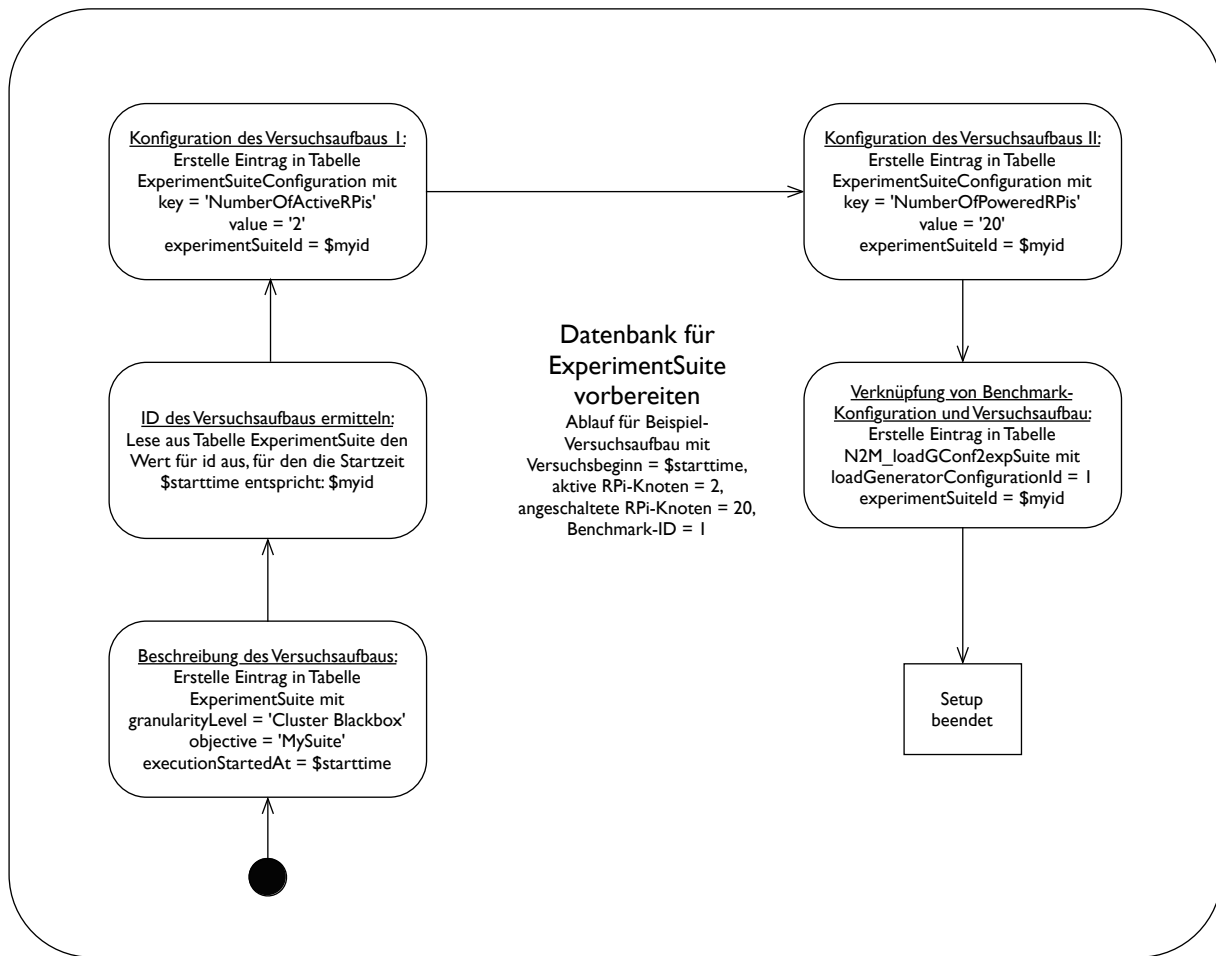


Abbildung 3.4: Aktivitätsdiagramm zur Konfiguration der Datenbank für eine ExperimentSuite.

c) Iteration über n RPi-Knoten:

i. Datenbank für ExperimentSuite vorbereiten.

Die Datenbank wird wie in Schritt b) i. beschrieben vorbereitet. Der Unterschied zu Messreihe 1 besteht in der Anzahl angeschalteter RPi-Knoten ($n+1$ statt 20).

ii. Parallele Ausführung des auf n RPi-Knoten.

Analog zu Schritt b) ii..

iii. Ermittlung des Ausführungsendes.

Analog zu Schritt b) iii..

iv. Herunterfahren nicht mehr aktiver RPi-Knoten.

In Messreihe 2 werden nicht mehr aktive RPi-Knoten heruntergefahren. Davon ausgenommen ist der Berechnungsknoten pi03.

v. Schreiben der Ergebnisdaten.

Analog zu Schritt b) iv..

d) Parsen der Logdateien für die Datenbank-Eingabe.

Die Ausgabe der Benchmark-Programme entspricht nicht dem Eingabeformat für die Datenbank. Daher wird pro Benchmark eine Datenbank-Eingabedatei erstellt. Sie enthält pro ExperimentSuite eine Zeile mit Messwerten und Zeitstempel.

Eine Beispielzeile für STREAM:

```
Copy: 3649.3 0.167595 0.166607 0.168846 Scale: 3428.0 0.178749  
0.177361 0.180439 Add: 4749.8 0.192614 0.192010 0.193745 Triad:  
4630.1 0.197780 0.196974 0.199436 Unixtime: 1396608095
```

Eine Beispielzeile für HPL:

```
WR00L2L2 29 1 2 2 0.09 1.965e-04 HPL_pdgesv() end time Wed Mar  
26 15:39:05 2014
```

e) Messergebnisse in Datenbank einlesen.

Falls die Zeitstempel in der Datenbank-Eingabedatei noch nicht dem Format des Unix-Zeitstempels entsprechen (vgl. <http://unixhelp.ed.ac.uk/CGI/man-cgi?date>), werden sie konvertiert. Für jeden Messwert wird ein Eintrag in der Tabelle `MeasurementValue` mit Parameter, Messwert, Zeitstempel und ID der ExperimentSuite erstellt.

Zur Durchführung dieser Schritte wurden die Shellskripte `startBenchmarks.sh`, `STREAM.sh`, `wrapHpl.sh`, `hplMessreihe1.sh` und `hplMessreihe2.sh` erstellt.

3.2 Strommessung

Der Stromverbrauch wird mit dem Strommessgerät *Energenie EGM-PWM-LAN* (vgl. <http://energenie.com/item.aspx?id=6736&lang=de>) ermittelt. Es wird im Folgenden als *Energenie* bezeichnet. Es wird zwischen Steckdose und Bramble angebracht und über ein LAN-Kabel mit dem Netzwerk verbunden.

Für den Versuchsaufbau war dem Energenie eine statische IP-Adresse zugewiesen und ein DNS-Eintrag erstellt worden. Damit kann im lokalen Netzwerk auf das Gerät und die ermittelten Messwerte zugegriffen werden. Der Zugriff kann unter anderem durch eine Webbrowser- und eine Windows-Anwendung erfolgen. Damit die Software das Gerät erkennt, muss es sich im selben Subnetz befinden wie der zugreifende Rechner.

Die Software musste einmalig für die Benutzung des Energenie eingerichtet werden. Dazu wurden nach der Installation des Programms *PowerManager* IP-Adresse und ein Gerätenamen eingetragen. In der Webbrowser-Anwendung wurde das Standardpasswort geändert und die NTP-Zeitsynchronisation aktiviert.

Das Energieie misst Spannung in Volt, elektrischen Strom in Ampère, Leistung in Watt und Arbeit in Kilowattstunden. Die Messgenauigkeit innerhalb des Messbereichs von 50W – 2500W beträgt $\pm 2\%$ bzw. $\pm 1W$ (vgl. [scr14]).

Für den Versuchsaufbau erwies sich die Webbrowser-Anwendung als ungeeignet. Zwar werden auf einer grafischen Oberfläche die jeweils aktuellen Werte für Spannung, elektrischen Strom, Leistung und Arbeit angezeigt, doch es können keine Messwerte für eine Zeitspanne abgerufen werden.

Das Programm PowerManager bietet erweiterte Funktionalitäten, z.B. Export der Messwerte über eine Zeitspanne als **xls**-Datei. Pro Messung werden drei Werte angegeben: Beginn der Messung, Ende der Messung und Leistung in Watt. Außerdem werden die Messwerte in einer SQLite-Datenbank **database.sqlite** im Verzeichnis **C:\Program Data\PowerManagerDatabase** auf der zugreifenden Windows-Maschine abgelegt. Pro Messung werden zusätzliche Werte abgelegt wie Widerstand in Ohm und Frequenz in Hertz. Für die ExperimentSuites sind die Einträge **Stamp** (Zeitstempel als Integer-Wert) und **P** (Leistung in Watt als Double-Wert) in der Tabelle **LogValues** von Bedeutung. Die Entscheidung für das Auslesen der SQLite-Datenbank gegenüber dem **xls**-Export fiel wegen der feineren Granularität der Zeitstempel.

Das Diagramm 3.5 zeigt, welche Schritte aus Benutzersicht pro ExperimentSuite zur Strommessung notwendig sind:

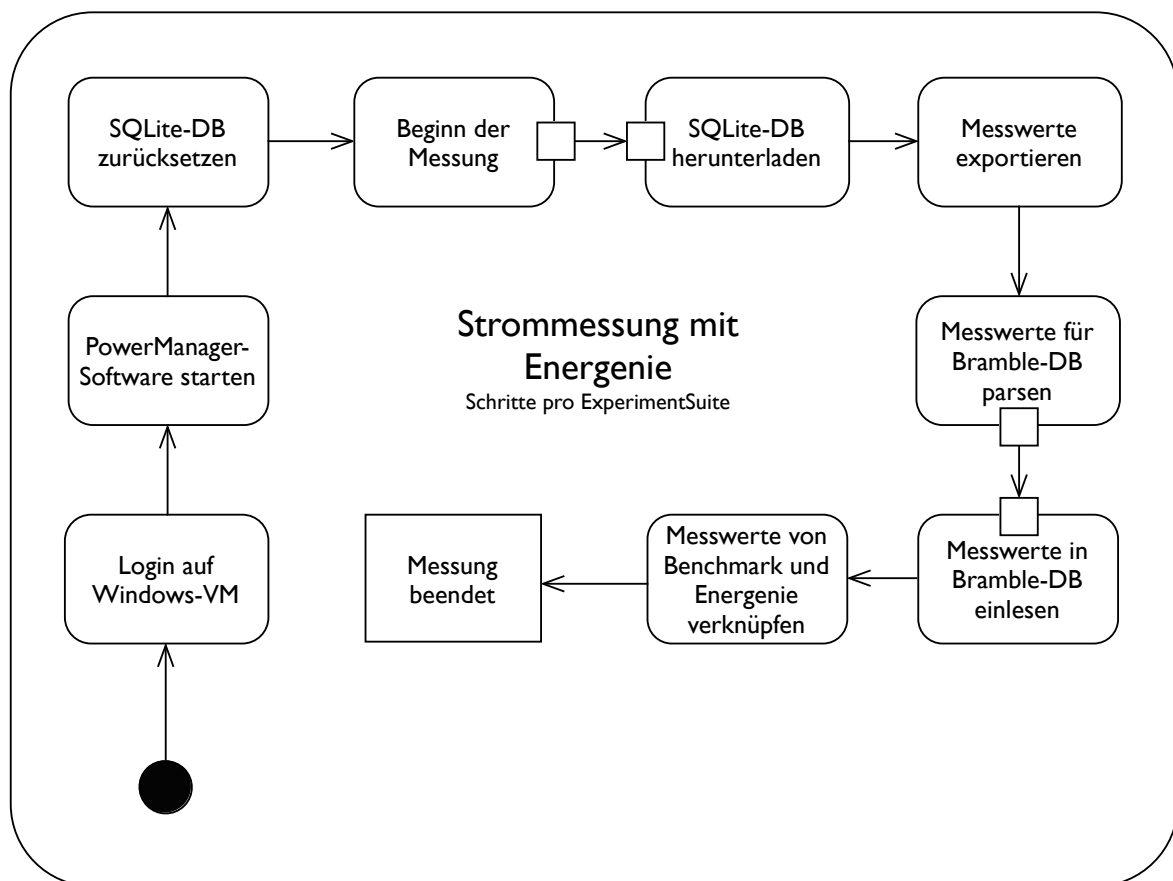


Abbildung 3.5: Aktivitätsdiagramm der Strommessung für eine ExperimentSuite.

1. Login auf Windows-Maschine.

Das Programm PowerManager wird nur für ein Windows-Betriebssystem (Windows XP oder höher) angeboten. Für die Ausführung war eine virtuelle Windows-Maschine (Windows 7) im lokalen Netzwerk zur Verfügung gestellt worden.

2. Start der PowerManager-Software.

3. Start der grafischen Aufzeichnung.

Optional ist es möglich, die aktuellen Messwerte für die Leistung auf einer grafischen Oberfläche anzeigen zu lassen. Es hat sich zur Überwachung der Messung als sinnvoll erwiesen.

4. Herunterladen der SQLite-Datenbank.

Zur weiteren Verarbeitung der Messwerte wird die Datenbank von der Windows-Maschine heruntergeladen.

5. Export der Messwerte.

Zur weiteren Verarbeitung der Messwerte wird die Tabelle `LogValues` exportiert.

6. Parsen der Messwerte für MySQL-Datenbank.

Zum Einlesen der Messwerte in `rpiWerte` müssen sie in ein passendes Format gebracht werden. Hierfür wurde ein Shellskript `parseEnergyValues.sh` erstellt. Es erstellt eine Eingabedatei, die pro Messung eine Zeile mit Leistung und Unix-Zeitstempel enthält.

7. Einlesen der Messwerte in MySQL-Datenbank.

Für jede Messung muss ein Eintrag in der Tabelle `MeasurementValue` mit Parameter, Messwert, Zeitstempel und ID des Energienie erstellt werden. Hierfür wurde ein Shellskript `writeEnergyValues.sh` erstellt.

8. Verknüpfung der Messwerte von Energienie und Benchmark-Programm.

Im letzten Schritt müssen die Messwerte des Energienie mit den IDs der ExperimentSuites der jeweiligen Messreihe verknüpft werden. Hierzu wurde ein Shellskript `matchEnergyValues.sh` erstellt.

3.3 Fehlerbehebung

Während der Versuchsdurchführung zeigten sich Fehlerquellen. Folgende Fehlerfälle mussten untersucht und behoben werden:

1. Defekte Hardware.

Neben den defekten Mini-USB-Kabeln hielten mehrere SD-Karten den häufigen Schreibzugriffen nicht stand. Sie mussten durch neue SD-Karten ersetzt werden.

2. RPi-Knoten nicht erreichbar (ping).

Häufig reagierte ein einzelner RPi-Knoten nicht auf ein `ping` von einem anderen RPi-Knoten oder von `careme` aus (Fehlermeldung: `Destination Host Unreachable`), obwohl die Status-LED Netzwerkaktivität anzeigte. Als einzige Lösung erwies sich Ziehen und wieder Einstecken des Mini-USB-Kabels. War das nicht erfolgreich, musste der Vorgang mit dem Netzkabel wiederholt werden. Nach einigen Minuten war der Zielknoten i.d.R. wieder mit `ping` erreichbar.

3. RPi-Knoten 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-Knoten aus eine SSH-Verbindung aufzubauen. Auch hier musste das Netzkabel gezogen und wieder eingesteckt werden. Nach einigen Minuten war der Zielknoten wieder erreichbar.

Gelegentlich erfolgte ein überraschender Passwortprompt für `root` beim Versuch, eine SSH-Verbindung von einem anderen RPi-Knoten aus aufzubauen. Der RSA Public Key von `root` musste in die Datei `~/.ssh/authorized_keys` auf dem Zielknoten eingetragen werden.

Gelegentlich erfolgte die Fehlermeldung `WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!`. Der RSA Public Key des anfragenden RPi-Knotens musste in der Datei `~/.ssh/known_hosts` auf dem Zielknoten korrigiert werden.

4. Geteiltes Verzeichnis nicht eingehängt.

Beim Neustart eines RPi-Knotens wurde meist das geteilte Verzeichnis nicht eingehängt (Fehlermeldung z.B. `-bash: /srv/libraries/etc/.sharedprofile: No such file or directory`). Es musste mit `mount /srv` auf dem betreffenden RPi-Knoten eingehängt werden.

5. Bash-Befehle werden nicht erkannt. Gelegentlich wurden auf einzelnen RPi-Knoten häufig verwendete Bash-Befehle nicht erkannt (Fehlermeldung z.B. `mpiexec: command not found`). Es musste ein Logout und erneuter Login auf dem betreffenden RPi-Knoten erfolgen.

Der Ausschluss dieser Fehlerfälle wurde weitestgehend in die Vorbereitung einer ExperimentSuite integriert: Überprüfung der Netzwerkverbindung aller RPi-Knoten mit `ping` vom Berechnungsknoten `pi03` aus, Einhängen des geteilten Verzeichnisses und damit gleichzeitig die Überprüfung der SSH-Verbindung zu allen RPi-Knoten (vgl. Kap. 3.1.2).

Spontan auftretende Fehler konnten nicht im Vorfeld ausgeschlossen werden. Die Durchführung einer ExperimentSuite erforderte daher immer die Anwesenheit einer Aufsichtsperson, um z.B. Kabel und Status-LEDs zu überprüfen. Trat ein spontaner Fehler auf, wurde die Durchführung abgebrochen, bereits erstellte Datenbankeinträge gelöscht und von Neuem mit den Vorbereitungen begonnen. Auch das Trennen nicht mehr aktiver RPi-Knoten vom Stromnetz musste manuell erfolgen. Jeder Neustart eines RPi-Knoten musste ebenfalls von Hand veranlasst werden, da nur durch Ziehen und erneutes Einstecken des Mini-USB-Kabels ein Neustart erreicht wird (vgl. Kap. 5).

Die Messergebnisse wurden dadurch nicht beeinflusst, da eine Messreihe bei Störungen abgebrochen und von Neuem begonnen wurde. Für zukünftige Versuchsaufbauten erscheint eine entsprechende Anpassung der Cluster-Architektur sinnvoll (vgl. Kap. 5).

3.4 Ergebnisse

Der folgende Abschnitt stellt die Untersuchungsergebnisse für HPL und STREAM auf dem Bramble dar. Da keine Implementierung von Whetstone für MPICH existiert, war im Verlauf der Untersuchung vorgegeben worden, Whetstone nicht zu berücksichtigen. HPL in der verwendeten Implementierung benötigt mindestens vier CPUs oder parallele Prozesse. Hier wurde jeder CPU, d.h. jedem RPi-Knoten, genau ein Prozess zugewiesen. Zur Ausführung

von HPL werden somit mindestens vier angeschaltete und aktive RPi-Knoten benötigt. Zur besseren Lesbarkeit wurde die Ausführung von STREAM daran angepasst. Auch STREAM wurde auf mindestens vier RPi-Knoten parallel ausgeführt.

3.4.1 HPL: Performance

HPL in der verwendeten Implementierung (vgl. Kap. 2.4.1) liefert folgende Ausgabe:

Zunächst werden die verwendeten Eingabe- und Ausgabeparameter erläutert. Die Ausgabeparameter sind invariabel: **Gflops** (Ausführungsrate in GFLOPs) und **Time** (Ausführungsdauer in Sekunden). **Gflops** wird auf 3, **Time** auf 2 Nachkommastellen genau angegeben.

Ein Teil der Eingabeparameter ist invariabel wie **eps** (Maschinengenauigkeit). Die anderen werden der Datei **HPL.dat** entnommen, die im selben Verzeichnis wie die ausführbare Datei liegen muss. Die entscheidenden Eingabeparameter sind Problemgröße bzw. Ordnung des zu lösenden linearen Gleichungssystems **N**, Blockgröße **NB**², Größe des Prozessnetzes (**Ps** und **Qs**)³, Panel-Faktorisierungsstrategie **PFACTs**⁴ und Teilpanel-Faktorisierungsstrategie **RFACTs**⁵ (vgl. <http://www.netlib.org/benchmark/hpl/algorithm.html>). Um möglichst gute Resultate für die Performance zu erzielen, muss eine sinnvolle Kombination dieser Parameter gefunden werden. Folgende Werte erwiesen sich als sinnvoll:

1. **N**: Die Problemgröße sollte so gewählt werden, dass der Hauptspeicher zu ca. 80% mit der Matrix gefüllt wird. Sie sollte mindestens einige 100 betragen und ein Vielfaches von 96 sein. Bei einer parallelen Ausführung ist die Gesamtmenge an Hauptspeicher entscheidend. Sie verhält sich proportional zu N^2 .

Wie in Kap. 2.5.1 beschrieben, verfügt der RPi über 512 MB SDRAM. Wie bei [Kli13] dargestellt, werden davon ca. 50 MB durch das Betriebssystem belegt, sodass noch ca. 450 MB pro RPi-Knoten zur Verfügung stehen.

Um die Proportionalitätskonstante für die Wahl von **N** zu ermitteln, wurde mit verschiedenen Vielfachen von 96 experimentiert und die Belegung des Hauptspeichers mit **top** kontrolliert. Für $n = 4$ RPi-Knoten wurde die beste Performance mit **N** = 2880 erzielt. Daraus wurde die Proportionalitätskonstante als

$$k = 8294400/1800 = 4608$$

berechnet.

Daraus ergeben sich folgende Werte für **N**:

$n = 4$ RPi-Knoten:

Verfügbarer Hauptspeicher = $450 \text{ MB} \cdot 4 = 1800 \text{ MB}$

$N = 30 \cdot 96 = 2880$ (empirisch ermittelt)

²Die Lösung des linearen Gleichungssystems erfolgt durch L/U-Faktorisierung. Dazu wird eine $n \times n + 1$ -Koeffizientenmatrix der Ausgangsmatrix *A* erzeugt. *A* wird dazu in in Blöcke der Größe **NB** \times **NB** aufgeteilt.

³Die Blöcke werden zur Bearbeitung einem Netz aus Prozessoren der Größe **P** \times **Q** übergeben. *P* bezeichnet die Anzahl von Prozessoren in einer Spalte, *Q* die Anzahl von Prozessoren in einer Zeile des Netzes.

⁴Zur Unterteilung der Matrix in Submatrizen können drei verschiedene Algorithmen verwendet werden: Links-schauende, rechts-schauende und Crout-Faktorisierung (vgl. <http://www.netlib.org/benchmark/hpl/tuning.html>).

⁵Als Teilpanel-Faktorisierungsstrategien werden dieselben Algorithmen angeboten wie für **PFACTs**.

$n = 8$ RPi-Knoten:

Verfügbarer Hauptspeicher = $450 \text{ MB} \cdot 8 = 3600 \text{ MB}$

$N = \sqrt{4608 \cdot 3600} \approx 4073 \Rightarrow$ setze $N = 42 \cdot 96 = 4032$

$n = 16$ RPi-Knoten:

Verfügbarer Hauptspeicher = $450 \text{ MB} \cdot 16 = 7200 \text{ MB}$

$N = \sqrt{4608 \cdot 7200} = 5760 \Rightarrow$ setze $N = 60 \cdot 96 = 5760$

2. **NB:** Die Blockgröße wird in Abhängigkeit von der Cache-Größe gewählt, der möglichst vollständig gefüllt werden soll, aber nicht überlaufen darf. Es empfiehlt sich eine Quadratzahl zu wählen, ferner ein Vielfaches von 2.

Der Level 2-Cache von 128 KB des RPi ist für die GPU reserviert (vgl. <http://sandsoftwaresound.net/raspberry-pi/memory-hierarchy/>). Um ein Überlaufen des Level 1-Datencaches von 16 KB zu vermeiden, wurde $NB = 8$ gesetzt.

3. **Ps und Qs:** Es empfiehlt sich ein möglichst quadratisches Prozessnetz zu wählen, ferner ein Vielfaches von 2. Folgende Werte wurden daher gesetzt:

$n = 4$ RPi-Knoten: $Ps = 2, Qs = 2$

$n = 8$ RPi-Knoten: $Ps = 4, Qs = 2$

$n = 16$ RPi-Knoten: $Ps = 4, Qs = 4$

4. **PFACTs:** Die beste Performance wurde mit $PFACTs = \text{Left}$ erzielt.

5. **RFACTs:** Die beste Performance wurde mit $RFACTs = \text{Crout}$ erzielt.

Offensichtlich müssen für unterschiedliche Anzahlen von Prozessoren unterschiedliche Eingabedateien verwendet werden. Daher wurde für jede HPL-ExperimentSuite ein eigenes Arbeitsverzeichnis erstellt und die ausführbare Datei zusammen mit der entsprechenden Eingabedatei dort abgelegt.

Die Diagramme 3.6 und 3.7 zeigen die Ergebnisse von Messreihe 1. Ausgabeparameter sind Ausführungsrate in GFLOPs und Ausführungsdauer in Sekunden, jeweils skaliert auf $16 - 4$ RPi-Knoten. Die Diagramme 3.8 und 3.9 stellen die Ergebnisse von Messreihe 1 und Messreihe 2 gegenüber. Die Ergebnisse von Messreihe 1 sind rot, die von Messreihe 2 blau dargestellt.



3.4.2 STREAM: Performance

STREAM in der verwendeten Implementierung (vgl. Kap. 2.4.3) liefert folgende Ausgabe:

Zunächst erfolgt eine Erläuterung der verwendeten Eingabe- und Ausgabeparameter. Die wichtigsten Eingabeparameter sind Anzahl an Prozessoren, Vektorlänge, Speicherbedarf und Anzahl an Iterationen für jedes Modul⁶. Für die Module Copy, Scale, Add und Triad (vgl. Kap. 2.4.3) werden die beste erzielte Ausführungsrate in MB/s (**Rate**) und die durchschnittliche (**Avg time**), minimale (**Min time**) und maximale Ausführungsdauer (**Max time**) in Sekunden ausgegeben. Die Ausführungsrate wird auf eine, die Ausführungsdauer auf 6 Nachkommastellen genau angegeben.

Die Diagramme 3.10 und 3.11 zeigen die Ergebnisse von Messreihe 1 für STREAM. Ausgabeparameter sind Ausführungsrate in MB/s und durchschnittliche Ausführungsdauer in Sekunden für alle Module, jeweils skaliert auf $19 - 4$ RPi-Knoten.

⁶Im Gegensatz zu HPL gibt es keine Eingabedatei, d.h. eine Veränderung ist nur direkt im Quellcode möglich (vgl. hierzu Kap. 4.2).

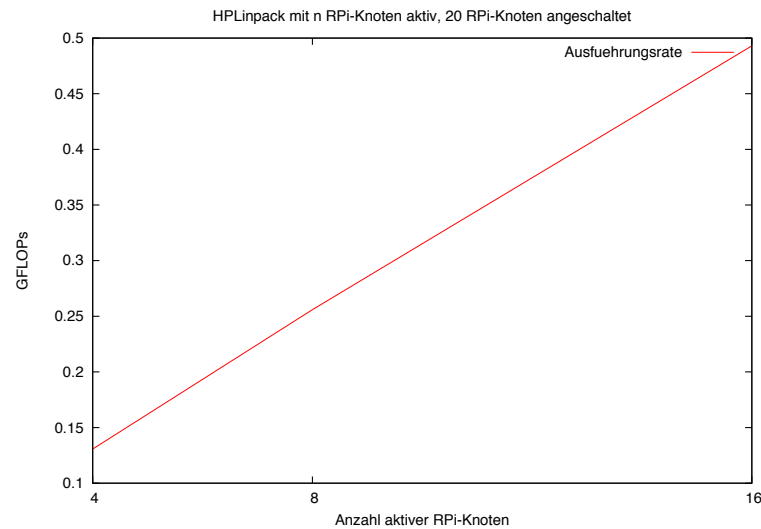


Abbildung 3.6: Ausführungsrate für HPL, Messreihe 1.

Die Diagramme 3.12 und 3.13 stellen die Ergebnisse von Messreihe 1 und Messreihe 2 gegenüber. Die Ergebnisse von Messreihe 1 sind rot, die von Messreihe 2 blau markiert.



3.4.3 Stromverbrauch

Die Diagramme 3.14 und 3.15 zeigen die Ergebnisse der Strommessung für HPL. Diagramm 3.14 zeigt den Stromverbrauch in Messreihe 1. Diagramm 3.15 stellt den Stromverbrauch von Messreihe 1 und Messreihe 2 gegenüber.

Die Diagramme 3.16 und 3.17 zeigen die Ergebnisse der Strommessung für STREAM. Diagramm 3.16 zeigt den Stromverbrauch in Messreihe 1. Diagramm 3.17 stellt den Stromverbrauch von Messreihe 1 und Messreihe 2 gegenüber.

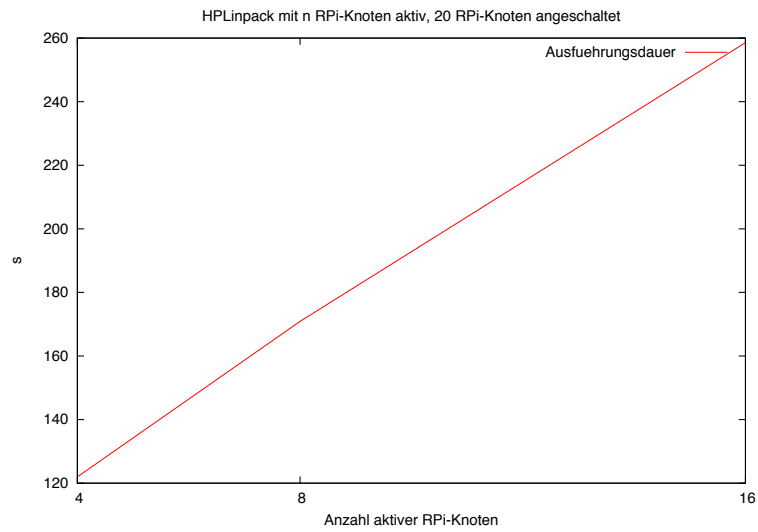


Abbildung 3.7: Ausführungsdauer für HPL, Messreihe 1.

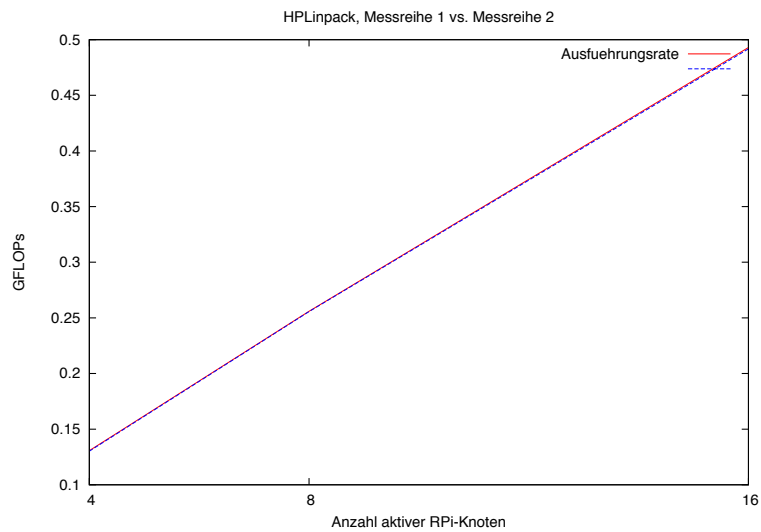


Abbildung 3.8: Ausführungsrate für HPL, Messreihe 1 und Messreihe 2.

3 Versuchsaufbau und -ablauf

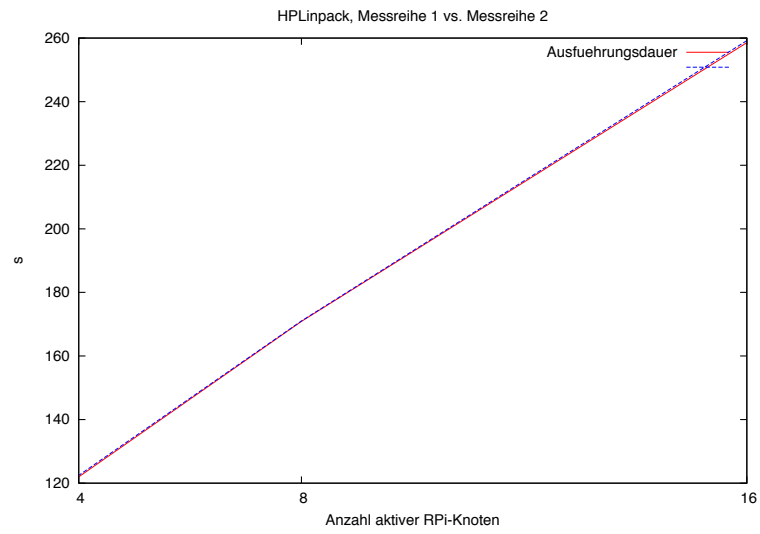


Abbildung 3.9: Ausführungsdauer für HPL, Messreihe 1 und Messreihe 2.

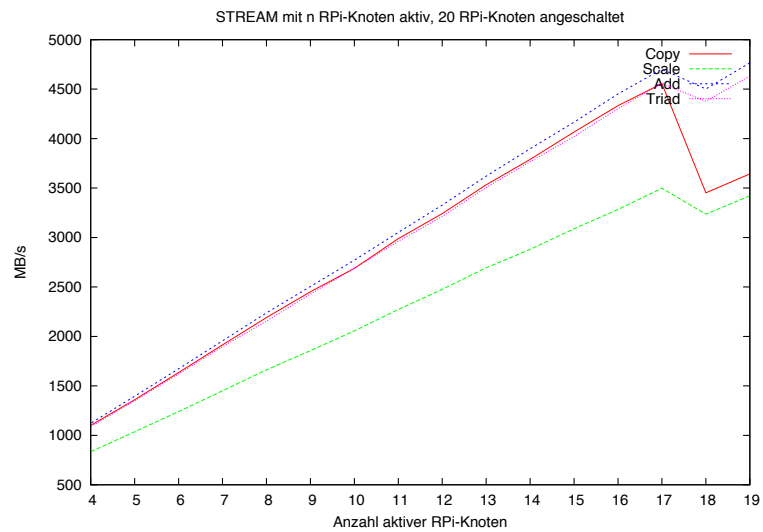


Abbildung 3.10: Ausführungsrate für STREAM, Messreihe 1.

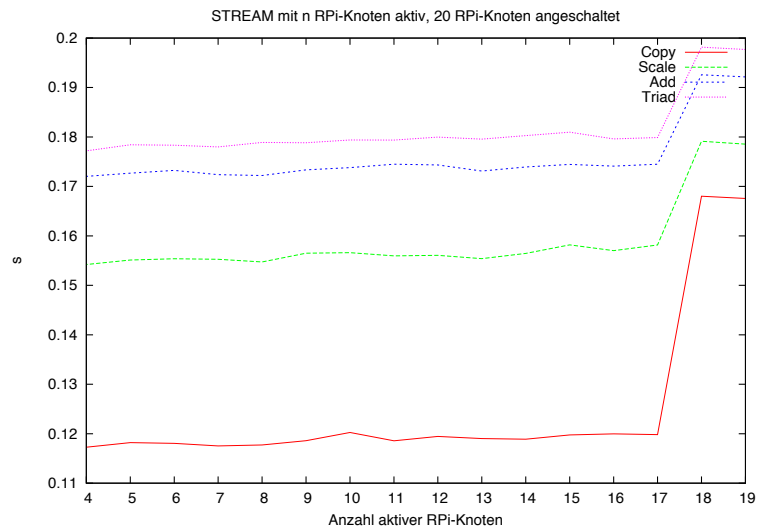


Abbildung 3.11: Ausführungsdauer für STREAM, Messreihe 1.

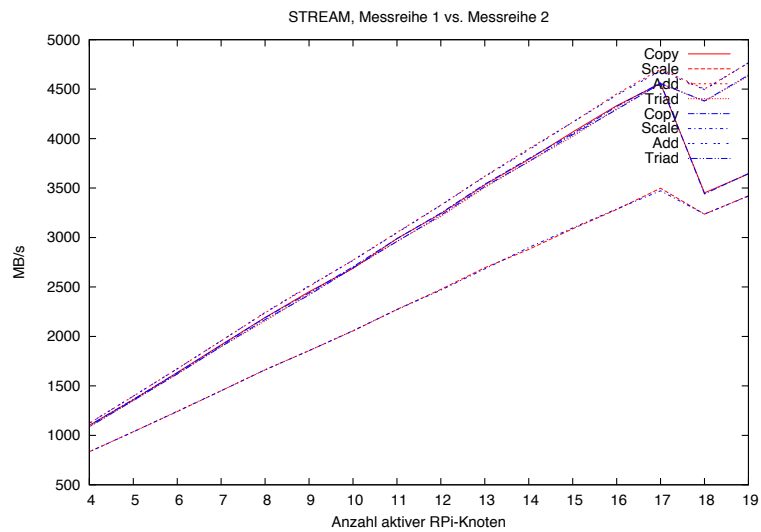


Abbildung 3.12: Ausführungsrate für STREAM, Messreihe 1 und Messreihe 2.

3 Versuchsaufbau und -ablauf

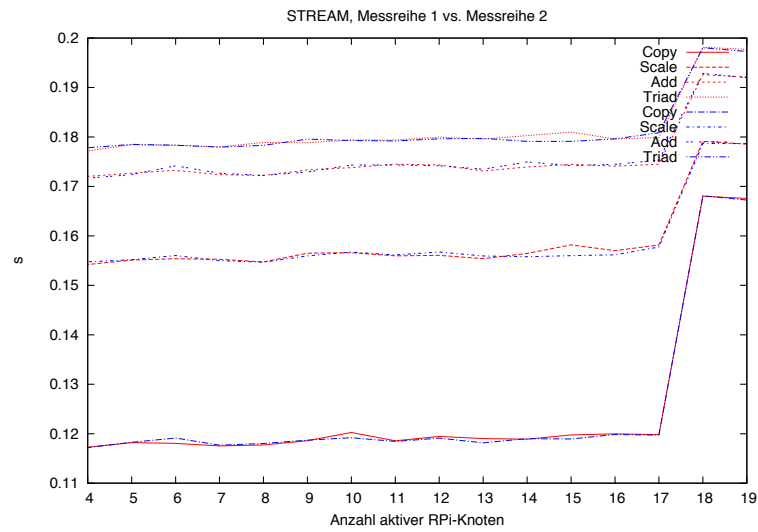


Abbildung 3.13: Ausführungsdauer für STREAM, Messreihe 1 und Messreihe 2.

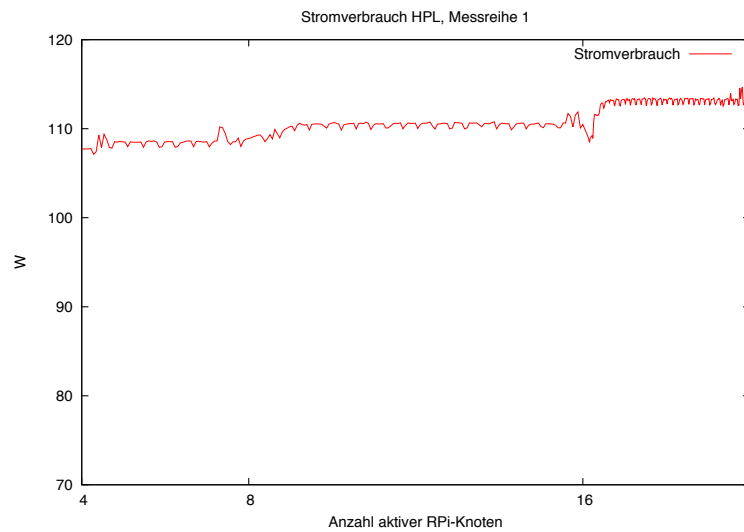


Abbildung 3.14: Stromverbrauch für HPL, Messreihe 1.

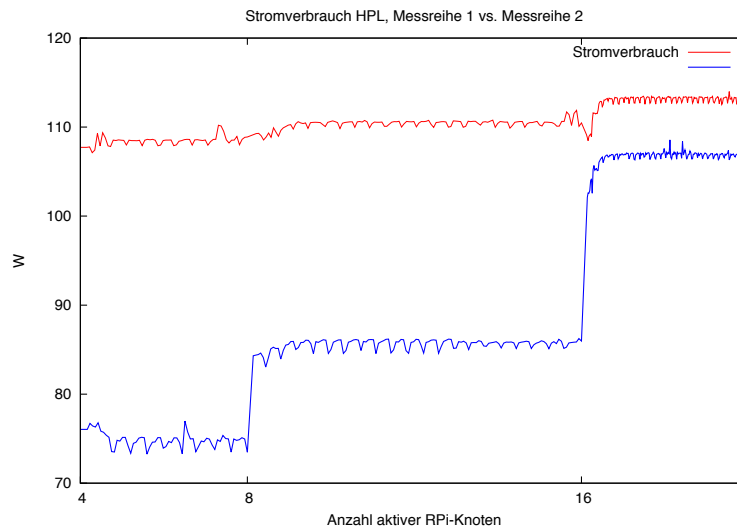


Abbildung 3.15: Stromverbrauch für HPL, Messreihe 1 und Messreihe 2.

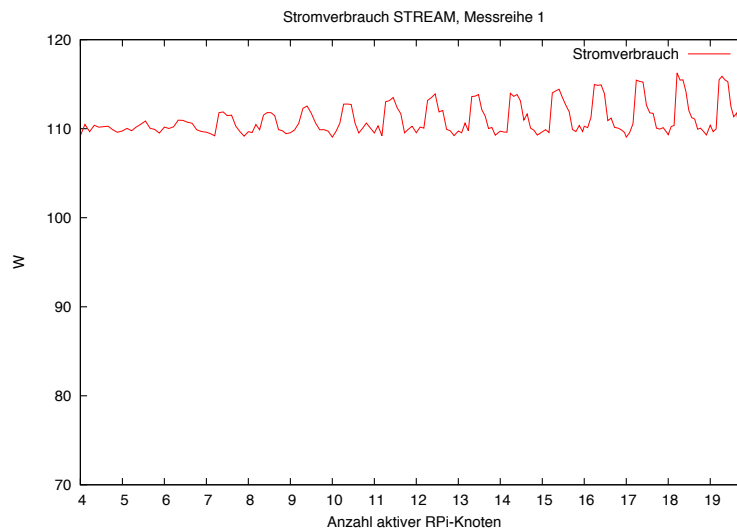


Abbildung 3.16: Stromverbrauch für STREAM, Messreihe 1.

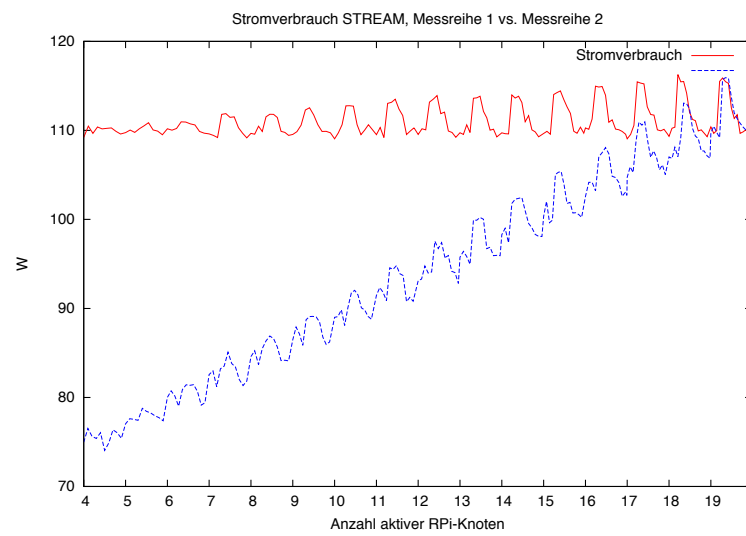


Abbildung 3.17: Stromverbrauch für STREAM, Messreihe 1 und Messreihe 2.



4 Interpretation

Das folgende Kapitel dient der Bewertung und Einordnung der Untersuchungsergebnisse. Im Fokus steht dabei das Skalierungsverhalten des Bramble.

4.1 HPL: Performance

Wie in Kapitel 3.4.1 dargestellt, besteht ein proportionales Verhältnis zwischen dem Quadrat der Problemgröße und der Größe des Gesamthauptspeichers. Somit ist bei Hinzunahme von Ressourcen in Form von RPi-Knoten ein linearer Anstieg der Ausführungsrate zu erwarten. Das gilt ebenfalls für die Ausführungsdauer. Die Diagramme 3.6 und 3.7 bestätigen die Erwartungen: Der Anstieg der Ausführungsdauer ist nahezu linear und beträgt im Mittel 0.031 GFLOPs pro zusätzlicher Ressource. Die maximale Abweichung hiervon beträgt 0.008 GFLOPs. Der Anstieg der Ausführungsdauer ist nahezu linear und beträgt im Mittel 13.28 s. Die maximale Abweichung hiervon beträgt 8.24 s.

Das Skalierungsverhalten des Bramble bei der Ausführung von HPL kann somit als erwartungsgemäß bezeichnet werden.

Versuchsaufbau und Funktionsweise des Benchmarks lassen erwarten, dass sich kein signifikanter Unterschied in Ausführungsrate und Ausführungsdauer zeigt, wenn nicht an der Programmausführung beteiligte RPi-Knoten heruntergefahren werden. Die Diagramme 3.8 und 3.9 bestätigen diese Erwartung: Messreihe 1 und Messreihe 2 werden nahezu identisch dargestellt. Die Ausführungsdauer in Messreihe 2 ist geringfügig höher als in Messreihe 1. Die Abweichung beträgt im Mittel 0.41 s. Die maximale Abweichung beträgt 0.63 s. Die Ausführungsrate ist in Messreihe 2 für $n = 4$ und $n = 16$ RPi-Knoten geringfügig niedriger als in Messreihe 1. Die Abweichung beträgt im Mittel 0.001 GFLOPs (gerundet auf drei Nachkommastellen an Hand der Messgenauigkeit, vgl. Kap. 3.4.1). Die maximale Abweichung beträgt 0.001 GFLOPs.

Es lässt sich schlussfolgern, dass das Herunterfahren nicht mehr aktiver RPi-Knoten keine signifikanten Auswirkungen auf Ausführungsrate und Ausführungsdauer von HPL auf dem Bramble hat. Das Skalierungsverhalten in Messreihe 1 und Messreihe 2 kann als gleich bezeichnet werden.

4.2 STREAM: Performance

Die Funktionsweise des Benchmarks lässt einen linearen Anstieg der Ausführungsrate bei der Hinzunahme von Ressourcen erwarten. Für die Ausführungsdauer auf jeder einzelnen CPU ist ein konstantes Verhalten zu erwarten. Die Diagramme 3.10 und 3.11 bestätigen die Erwartungen für $n \leq 17$ RPi-Knoten: Der Anstieg der Ausführungsdauer ist nahezu linear und beträgt im Mittel 245.9 MB/s (Copy), 205.1 MB/s (Scale), 275.5 MB/s (Add) bzw. 266.9 MB/s (Triad) pro zusätzlicher Ressource. Die maximale Abweichung hiervon beträgt 54.2 MB/s (Copy), 19.7 MB/s (Scale), 25.9 MB/s (Add) bzw. 25.6 MB/s (Triad).

Die Ausführungsdauer verhält sich nahezu konstant und beträgt im Mittel 0.118791 s (Copy), 0.156077 s (Scale), 0.173474 s (Add) bzw. 0.179196 s (Triad). Die maximale Abweichung hiervon beträgt 0.001525 s (Copy), 0.002114 s (Scale), 0.002463 s (Add) bzw. 0.003793 s (Triad).

Das Skalierungsverhalten des Bramble bei der Ausführung von STREAM kann somit für $n \leq 17$ RPi-Knoten als erwartungsgemäß bezeichnet werden.

Versuchsaufbau und Funktionsweise des Benchmarks legen nahe, dass sich kein signifikanter Unterschied in Ausführungsrate und Ausführungsdauer zeigt, wenn nicht an der Programmausführung beteiligte RPi-Knoten heruntergefahren werden. Die Diagramme 3.12 und 3.13 bestätigen diese Erwartung für $n \leq 17$ RPi-Knoten: Ausführungsrate von Messreihe 1 und Messreihe 2 werden nahezu identisch dargestellt. Die Abweichung von Messreihe 2 gegenüber Messreihe 1 beträgt im Mittel 0.1 MB/s (Copy), 0.5 MB/s (Scale), 2.9 MB/s (Add) bzw. 2.95 MB/s (Triad). Die maximale Abweichung gegenüber Messreihe 1 beträgt 25.4 MB/s (Scale auf $n = 16$ RPi-Knoten). Bei der Ausführungsdauer beträgt die Abweichung gegenüber Messreihe 1 im Mittel 0.000123 s (Copy), 0.000129 s (Scale), 0.000190 s (Add) bzw. 0.000122 s (Triad). Die maximale Abweichung gegenüber Messreihe 1 beträgt 0.000823 s (Copy auf $n = 15$ RPi-Knoten).

Für $n \leq 17$ RPi-Knoten lässt sich schlussfolgern, dass das Herunterfahren nicht mehr aktiver RPi-Knoten keine signifikanten Auswirkungen auf Ausführungsrate und Ausführungsdauer von STREAM auf dem Bramble hat. Das Skalierungsverhalten in Messreihe 1 und Messreihe 2 kann für $n \leq 17$ RPi-Knoten als gleich bezeichnet werden.

Für $n > 17$ RPi-Knoten weicht das Skalierungsverhalten des Bramble von den Erwartungen ab. Tabelle 4.1 stellt erwartete und erzielte Messwerte für Messreihe 1 gegenüber. Abweichungen von mehr als 1 MB/s (Ausführungsrate) und 0.01 s (Ausführungsdauer) gegenüber den erwarteten Werten sind rot markiert. Die erwarteten Werte werden für n RPi-Knoten wie folgt bestimmt:



$$\text{Erwartete Performance} = \frac{\text{Performance für } n = 17}{17} * n$$

$$\text{Erwartete Ausführungsdauer} = \text{Durchschnittliche Ausführungsdauer für } n \leq 17$$

Es stellt sich die Frage, warum für $n > 17$ RPi-Knoten eine deutlich schlechtere Performance und verlängerte Ausführungsdauer auftreten. Folgende Erklärungen sind denkbar:

1. Funktionsweise des Benchmarks.

Wie zu Beginn des Kapitels dargestellt, ist bei der Hinzunahme von Ressourcen auf einem Rechencluster mit einem linearen Anstieg der Ausführungszeit und annähernd konstanter Ausführungsdauer zu rechnen. McCalpin schreibt hierzu:

[...] unless something is very wrong, the performance of a cluster will be the performance of a node times the number of nodes (vgl. <http://www.cs.virginia.edu/stream/ref.html>).

Ein erwünschter Effekt ist somit nach der Funktionsweise des Benchmarks auszuschließen.

2. Architektur des Bramble.

Mögliche Ursachen sind Systemzeit, Netzwerk und Netz-Dateisystem.

Aktive RPi:	17	18	19
Copy in MB/s (erwartet):	4556.4	4824.4	5092.5
Copy in MB/s (erzielt):	4556.4	3451.6	3642.9
Copy in s (erwartet):	0.118791	0.118791	0.118791
Copy in s (erzielt):	0.119821	0.168030	0.167579
Scale in MB/s (erwartet):	3497.7	3703.5	3909.1
Scale in MB/s (erzielt):	3497.7	3236.2	3421.8
Scale in s (erwartet):	0.156077	0.156077	0.156077
Scale in s (erzielt):	0.158162	0.179140	0.178560
Add in MB/s (erwartet):	1672.5	4978.6	5255.2
Add in MB/s (erzielt):	1672.5	4501.4	4765.8
Add in s (erwartet):	0.173474	0.173474	0.173474
Add in s (erzielt):	0.174494	0.192586	0.192147
Triad in MB/s (erwartet):	4557.4	4852.5	4093.6
Triad in MB/s (erzielt):	4557.4	4377.4	4632.1
Triad in s (erwartet):	0.179196	0.179196	0.179196
Triad in s (erzielt):	0.179882	0.198159	0.197686

Abbildung 4.1: Erwartete und erzielte Messwerte für STREAM auf $n \geq 17$ RPi-Knoten.

Wie in Kap. 3.1.2 dargestellt, hat der RPi keine eingebaute Systemuhr. Die Zeitsynchronisation der RPi-Knoten erfolgt einmal beim Bootvorgang gegenüber dem OpenNTP-Server auf `careme` (vgl. [Kli13]). Wenn Rechenlast ungleich auf die RPi-Knoten verteilt wird und die beteiligten CPUs sehr ausgelastet sind, wäre es denkbar, dass die Systemzeit dieser Knoten driftet und zu abweichenden Messergebnissen bei der Ausführungsdauer führt. Dieser Effekt kann hier ausgeschlossen werden: Die Rechenlast ist bei $n \leq 17$ Knoten nicht weniger ungleich verteilt als bei $n > 17$ Knoten, sodass der Effekt schon früher eintreten müsste. Die Rechenlast der parallelen Ausführung von STREAM ist zudem für alle beteiligten Knoten gleich.

Wie in Kap. 2.6 dargestellt, sind Server und RPi-Knoten über ein Ethernet-Netzwerk verbunden. Wie bei [Kli13] erläutert, kann damit ein maximaler Datendurchsatz von ca. 2 MB/s erreicht werden. Ein Erklärungsansatz war das Überschreiten dieser Obergrenze bei mehr als 17 parallelen Ausführungen.

Bezüglich der Bandbreiten zeigte sich durch Nachrechnen eine Steigerung um rund 35 MB für Copy und Scale pro zusätzlichem RPi-Knoten. Für Add und Triad erfolgt eine Steigerung um rund 50 MB. Hierbei handelt es sich natürlich nicht um den Durchsatz des Netzwerks, sondern um den Durchsatz von Hauptspeicherzugriffen jedes einzelnen RPi-Knotens. Diese Steigerung erfolgt auch bei $n > 17$ RPi-Knoten. Der Durchsatz von Hauptspeicherzugriffen verhält sich somit erwartungsgemäß für alle n . Ein Überschreiten des maximalen Netzwerk-Datendurchsatzes scheidet somit als Ursache aus.

Binaries und verwendete Bibliotheken der Benchmarks liegen im geteilten Verzeichnis `/srv`. Es erschien denkbar, dass mehr als 17 parallele Zugriffe darauf das Netzdateisystem überlasten. Deswegen wurde probeweise auf den SD-Karten aller RPi-Knoten eine neue Partition erstellt und der Inhalt des geteilten Verzeichnisses hinein kopiert. In der Datei `/etc/fstab` aller RPi-Knoten wurde der Mountpoint für

Aktive RPi-Knoten:	18
Copy in MB/s (erwartet):	4824.4
Copy in MB/s (erzielt):	3398.3
Copy in s (erwartet):	0.118791
Copy in s (erzielt):	0.170848
Scale in MB/s (erwartet):	3703.5
Scale in MB/s (erzielt):	3189.5
Scale in s (erwartet):	0.156077
Scale in s (erzielt):	0.181711
Add in MB/s (erwartet):	4978.6
Add in MB/s (erzielt):	4373.3
Add in s (erwartet):	0.173474
Add in s (erzielt):	0.199341
Triad in MB/s (erwartet):	4852.5
Triad in MB/s (erzielt):	4149.8
Triad in s (erwartet):	0.179196
Triad in s (erzielt):	0.209507



Abbildung 4.2: Erwartete und erzielte Messwerte für STREAM auf $n = 18$ RPi-Knoten mit `ntimes=100`.

/srv temporär entsprechend geändert. Das Resultat waren Performance-Einbrüche bei STREAM schon ab $n = 13$ RPi-Knoten, womit das Netz-Dateisystem als Ursache ausscheidet.

3. Ausführung des Benchmarks.

Eine mögliche Ursache ist die Funktionsweise von MPICH.

Wie in Kap. 2.6.3 und 3.1.2 beschrieben, wird die parallele Ausführung von STREAM durch MPICH angestoßen, das den im Machinefile spezifizierten CPUs eine bestimmte Anzahl an parallelen Programmaufrufen zuweist. Ein weiterer Erklärungsansatz war die Überlastung des Netzwerks durch den Kommunikations-Overhead ab einem Schwellenwert von $n = 18$ parallelen Programmaufrufen.

MPI nutzt zur Interprozesskommunikation u.a. Broadcast- und Unicast-Nachrichten. Bei der parallelen Ausführung von Anwendungen werden meist die Funktionen `MPI_Bcast` und `MPI_Gather` aufgerufen. Hierbei schickt ein Root-Prozess entweder eine Nachricht an alle beteiligten CPUs bzw. Prozesse oder sammelt die Daten aller beteiligten CPUs bzw. Prozesse ein (vgl. [Pie12]). Es erschien denkbar, dass eine Broadcast-Nachricht an mehr als 17 Prozessoren das Netzwerk überlastet und zu den beobachteten Performance-Einbrüchen führt. Daher wurde der Quellcode von STREAM testweise so verändert, dass jedes Modul 100 Mal statt zehn Mal ausgeführt wird¹. Bei einer längeren Ausführungsdauer des Programms war zu erwarten, dass der Kommunikations-Overhead von MPICH abnimmt und erwartungsgemäße Performance-Resultate erzielt werden. Das war nicht der Fall. Folgende Ergebnisse wurden für $n = 18$ RPi-Knoten erzielt: Die Resultate fallen noch schlechter aus als bei STREAM auf $n = 18$ RPi-Knoten

¹Dazu muss die Variable `ntimes` verändert werden, vgl. <http://www.cs.virginia.edu/stream/ref.html>.

	HPL (1)	HPL (2)	STREAM (1)	STREAM (2)
ExperimentSuite	110 W	88	111 W	94 W
Maximale Abweichung	4 W	3 W	5 W	5 W
Zuwachs pro RPi-Knoten	0 W	8 W	0 W	2 W
Messreihe	111 W	88 W	111 W	94 W

Abbildung 4.3: Stromverbrauch des Bramble, Messreihe 1 und Messreihe 2.

mit `ntimes=10` (vgl. Tabelle 4.1). Damit scheidet ein Kommunikations-Overhead von MPICH als Erklärung aus. Die Ursache konnte somit nicht zweifelsfrei geklärt werden.



4.3 Stromverbrauch

Ziel der Strommessung war die Ermittlung des Skalierungsverhaltens des Bramble bezüglich des Stromverbrauchs. In Messreihe 1 werden laufend Ressourcen in Form von RPi-Knoten hinzugenommen. Auch nicht aktive RPi-Knoten sind angeschaltet, nehmen somit Strom auf. In Messreihe 2 werden nicht aktive RPi-Knoten abgeschaltet, nehmen somit keinen Strom auf.

Es ist zu erwarten, dass der Stromverbrauch deutlich sinkt, wenn nicht aktive RPi-Knoten von der Stromversorgung getrennt werden. Zur Überprüfung dieser Erwartung wurden Stromverbrauch bei der Ausführung von HPL und STREAM (Messreihe 1 und Messreihe 2), Zuwachs pro aktivem RPi-Knoten (Messreihe 1) und Zuwachs pro angeschaltetem und aktivem RPi-Knoten (Messreihe 2) empirisch ermittelt.

Die Ergebnisse werden in Tabelle 4.3 als Mittelwerte dargestellt (gerundet auf 4 Stellen ohne führende Nullen an Hand der Messgenauigkeit, vgl. Kap. 3.2).



Die Messergebnisse bestätigen die Erwartung, dass der Stromverbrauch des Bramble pro abgeschaltetem RPi-Knoten deutlich abnimmt: Im Mittel um 8 W pro RPi-Knoten bei der Ausführung von HPL und um 2 W pro RPi-Knoten bei der Ausführung von STREAM. Der Stromverbrauch bei der Durchführung von Messreihe 2 ist damit im Mittel um 23 W (HPL) bzw. 17 W (STREAM) niedriger als in Messreihe 1. Das Skalierungsverhalten des Bramble bezüglich des Stromverbrauchs kann somit als erwartungsgemäß bezeichnet werden.



5 Zusammenfassung und Ausblick

I have converted my Classic Benchmarks to run on the Linux based Raspberry Pi. These are Whetstone, [...], Linpack and Livermore Loops. [...] The Livermore Loops benchmark was used to accept the first supercomputer. So the main bragging rights are:

In 1978, the Cray 1 supercomputer cost \$7 Million, weighed 10,500 pounds and had a 115 kilowatt power supply. It was, by far, the fastest computer in the world. The Raspberry Pi costs around \$70 (CPU board, case, power supply, SD card), weighs a few ounces, uses a 5 watt power supply and is more than 4.5 times faster than the Cray 1.

My bragging rights are that I developed and ran benchmarks, including Whetstones, on Serial 1 Cray 1 (vgl. <http://www.raspberrypi.org/forum/viewtopic.php?f=31&t=44080>).

In der vorliegenden Arbeit wurde das Skalierungsverhalten eines Raspberry Pi-Beowulf-Clusters mit 20 RPi-Knoten unter Verwendung ausgewählter HPC-Benchmarks untersucht. Als Parameter wurden Performance (Ausführungsrate und Ausführungsdauer) und Stromverbrauch betrachtet.

Der RPi-Cluster mit der beschriebenen Architektur war zur Verfügung gestellt worden. Als Arbeitslast-Generatoren waren HPL und STREAM vorgegeben worden. Zur Integration der durchgeführten ExperimentSuites in einen größeren Versuchsaufbau war ein Datenbankschema vorgegeben worden. Es wurde an Hand der praktischen Arbeit aktualisiert. Ebenso wurden Fehlerfälle bezüglich Hardware und Systemkonfiguration laufend untersucht und weitestgehend eliminiert.

Die durchgeführten Messreihen ergaben ein kohärentes und erwartungsgemäßes Bild bezüglich Skalierungsverhalten des Clusters den Stromverbrauch betreffend. Dieser fällt linear ab, wenn nicht an der Programmausführung beteiligte RPi-Knoten heruntergefahren und abgeschaltet werden. Ebenso ist das Skalierungsverhalten **erwartungsgemäßbezüglich** der CPU-Performance bei der Ausführung von HPL. Ausführungsrate und Ausführungsdauer von HPL wachsen linear mit der Hinzunahme von Ressourcen.

Das Skalierungsverhalten bezüglich der Performance bei der Ausführung von STREAM für $n \leq 17$ RPi-Knoten ist ebenfalls als **erwartungsgemäßzu** bezeichnen. Die Ausführungsrate wächst linear mit der Hinzunahme von Ressourcen. Die Ausführungsdauer bleibt konstant.

Für $n > 17$ RPi-Knoten ist das Skalierungsverhalten des Clusters nicht erwartungsgemäß bezüglich der Performance bei der Ausführung von STREAM. Die Bandbreiten der Hauptspeicherzugriffe wachsen erwartungsgemäß linear, doch die Ausführungsdauer nimmt deutlich zu.



Wie Longbottom im obigen Zitat schreibt, können auf dem RPi ähnliche Ergebnisse bezüglich der CPU-Performance erzielt werden wie auf früheren Supercomputern. Ähnliches gilt für den Durchsatz an Hauptspeichierzugriffen (vgl. http://www.cs.virginia.edu/stream/stream_mail/2012/0002.html). Die vorliegenden Ergebnisse erbringen den Nachweis, dass sich die verwendeten HPC-Benchmarks auch auf einem RPi-Cluster mit sinnvollen Ergebnissen ausführen lassen.

Schwierigkeiten zeigten sich bei vorhandenen Cluster-Infrastruktur, sowohl Hardware als auch IP/SSH-Kommunikation betreffend. Mögliche zukünftige Arbeiten umfassen daher zwei Felder: Optimierung der Cluster-Architektur und Ausweitung des Versuchsaufbaus.

Die physische Architektur des Bramble erwies sich in Teilen als Hindernis. Der Aufbau ist so beengt, dass die ständig benötigten Ethernet- und Mini-USB-Anschlüsse von pi11 – pi20 nur schwer zugänglich sind. Manche Mini-USB-Kabel sind so kurz abgemessen, dass sie kaum bis zum entsprechenden Knoten reichen. Ein häufiges Trennen der Kabel, wie in den ExperimentSuites zum Messen des Stromverbrauchs vorgesehen, wird dadurch erschwert. Zudem besteht die Gefahr, die übrige Hardware zu beschädigen.

Die einzige Möglichkeit zum Reboot eines RPi ist Trennen und Wiederherstellen der Stromversorgung. Eine unterbrochene Netzwerkverbindung kann beim vorhandenen Cluster nur durch Trennen und Wiederherstellen des Netzkabels erfolgen. Ein direkter Zugriff auf einen RPi z.B. mit Tastatur und Monitor ist im Fehlerfall nicht hilfreich, da alle Einstellungen vom Bramble-Server aus vorgenommen werden. Abhilfe ließe sich u.U. durch den Einbau von Reset-Knöpfen auf den einzelnen RPi-Nodes schaffen (vgl. <http://raspi.tv/2012/making-a-reset-switch-for-your-rev-2-raspberry-pi>). Auch die Platzierung der in einem aufrecht stehenden Gestell statt des liegenden Metallgehäuses wäre für zukünftige Versuchsaufbauten sinnvoll (vgl. [Kie13] und [CCB⁺13]). Damit wären die häufig benötigten Anschlüsse besser zugänglich.

Für zukünftige Experimente mit HPC-Anwendungen muss gesagt werden, dass diese die RPi-Hardware unter Umständen an ihre physischen Grenzen bringen. Beim vorliegenden Versuchsaufbau wurden mehrere SD-Karten durch beständige Schreibzugriffe unbrauchbar. Hierfür sollte eine bessere Lösung gefunden werden.

Das abweichende Skalierungsverhalten des Clusters für die parallele Ausführung von STREAM auf $n > 17$ RPi-Knoten konnte trotz intensiver Fehlersuche nicht abschließend geklärt werden. Hier wäre eine noch tiefer gehende Untersuchung sinnvoll. Für Whetstone liegt bisher keine MPICH-Implementierung vor, weswegen vorgegeben wurde, den Benchmark nicht weiter zu berücksichtigen. Es wäre interessant, auf der bestehenden Architektur auch Ergebnisse für die parallele Ausführung von Whetstone zu ermitteln. Auch die Ausführung weiterer, bisher nicht betrachteter HPC-Benchmarks steht noch aus.

Die wichtigsten neuen Erkenntnisse sind jedoch zu erwarten, wenn letztlich ein quelloffener Treiber für die RPi-GPU vorliegt. Nachdem Broadcom vor Kurzem die vollständige Spezifikation der GPU veröffentlicht hat (vgl. <http://blog.broadcom.com/chip-design/android-for-all-broadcom-gives-developers-keys-to-the-videocore-kingdom>), hat die Raspberry Pi Foundation einen entsprechenden Wettbewerb ausgeschrieben (vgl. <http://www.raspberrypi.org/competition-rules>). Die Ergebnisse und neuen Einsatzmöglichkeiten der bisher kaum anprogrammierbaren GPU, die deutlich leistungsfähiger ist als die hier schwerpunktmäßig untersuchte CPU, dürfen mit Spannung erwartet werden.



Abbildungsverzeichnis

2.1	Schematischer Aufbau des hier verwendeten Bramble.	10
3.1	Komponentendiagramm des Versuchsaufbaus.	15
3.2	Aktivitätsdiagramm einer ExperimentSuite.	16
3.3	Aktivitätsdiagramm zur Konfiguration der Datenbank für einen Benchmark. .	17
3.4	Aktivitätsdiagramm zur Konfiguration der Datenbank für eine ExperimentSuite.	19
3.5	Aktivitätsdiagramm der Strommessung für eine ExperimentSuite.	21
3.6	Ausführungsrate für HPL, Messreihe 1.	26
3.7	Ausführungsdauer für HPL, Messreihe 1.	27
3.8	Ausführungsrate für HPL, Messreihe 1 und Messreihe 2.	27
3.9	Ausführungsdauer für HPL, Messreihe 1 und Messreihe 2.	28
3.10	Ausführungsrate für STREAM, Messreihe 1.	28
3.11	Ausführungsdauer für STREAM, Messreihe 1.	29
3.12	Ausführungsrate für STREAM, Messreihe 1 und Messreihe 2.	29
3.13	Ausführungsdauer für STREAM, Messreihe 1 und Messreihe 2.	30
3.14	Stromverbrauch für HPL, Messreihe 1.	30
3.15	Stromverbrauch für HPL, Messreihe 1 und Messreihe 2.	31
3.16	Stromverbrauch für STREAM, Messreihe 1.	31
3.17	Stromverbrauch für STREAM, Messreihe 1 und Messreihe 2.	32
4.1	Erwartete und erzielte Messwerte für STREAM auf $n \geq 17$ RPi-Knoten. . . .	35
4.2	Erwartete und erzielte Messwerte für STREAM auf $n = 18$ RPi-Knoten mit ntimes=100.	36
4.3	Stromverbrauch des Bramble, Messreihe 1 und Messreihe 2.	37





Literaturverzeichnis

- [Bal12] BALAKRISHNAN, Nikilesh: *Building and Benchmarking a Low Power ARM Cluster*. 2012. – <http://www.epcc.ed.ac.uk/sites/default/files/Dissertations/2011-2012/Submission-1126390.pdf>
- [BL08] BUHL, Hans ; LAARTZ, Jürgen: Warum Green IT nicht ausreicht – oder: Wo müssen wir heute anpacken, damit es uns übermorgen immer noch gut geht. In: *Wirtschaftsinformatik* (2008), S. 261–265. – <http://dx.doi.org/10.1365/s11576-008-0058-5>
- [CCB⁺13] COX, Simon ; COX, James ; BOARDMAN, Richard u. a.: Iridis-pi: A Low-cost, Compact Demonstration Cluster. In: *Cluster Computing* (2013), S. 1–10. – <http://dx.doi.org/10.1007/s10586-013-0282-7>
- [CW76] CURNOW, Harold ; WICHMANN, Brian: A Synthetic Benchmark. In: *The Computer Journal* (1976), S. 43–49. – <http://http://comjnl.oxfordjournals.org/content/19/1/43.full.pdf>
- [DLP03] DONGARRA, Jack ; LUSZCZEK, Piotr ; PETITET, Antoine: The LINPACK Benchmark: Past, Present and Future. In: *Concurrency and Computation: Practice and Experience* (2003), S. 803–820. – <http://onlinelibrary.wiley.com/doi/10.1002/cpe.728/pdf>
- [FH12] FICHTER, Klaus ; HINTERMANN, Ralph: Energieverbrauch und Energiekosten von Servern und Rechenzentren in Deutschland. Aktuelle Trends und Einsparpotenziale bis 2015 / Borderstep Institut für Innovation und Nachhaltigkeit. Berlin, Mai 2012. – Forschungsbericht. – [http://www.bitkom.org/files/documents/Kurzstudie__Borderstep_1_Rechenzentren\(1\).pdf](http://www.bitkom.org/files/documents/Kurzstudie__Borderstep_1_Rechenzentren(1).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>
- [LB12] LANGE, Walter ; BOGDAN, Martin: *Entwurf und Synthese von Eingebetteten Systemen: Ein Lehrbuch*. München : Oldenbourg, 2012. – ISBN 3486718401
- [LDK⁺05] LUSZCEK, Piotr ; DONGARRA, Jack ; KOESTER, David u. a.: Introduction to the HPC Challenge Benchmark Suite / Lawrence Berkeley National Laboratory.

- Berkeley, April 2005. – Forschungsbericht. – <http://escholarship.org/uc/item/6sv079jp>
- [McC95] MCCALPIN, John: A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers. In: *IEEE TCCA Newsletter* (1995), S. 19–25. – <http://www.cs.virginia.edu/~mccalpin/papers/balance>
- [McC05] MCCALPIN, John: *STREAM Benchmark*. 2005. – http://www.cs.virginia.edu/~mccalpin/STREAM_Benchmark_2005-01-25.pdf
- [Ou13] OU, Jun: *Pi and the Sky*. 2013. – <https://www.duo.uio.no/bitstream/10852/37445/4/Ou-Jun.pdf>
- [Pie12] PIETRZYK, Johannes: *Projekt Parallelrechnerevaluation*. 2012. – http://wr.informatik.uni-hamburg.de/_media/research/labs/2012/2012-02-johannes_pietrzyk-auswirkungen-verschiedener_parameter_des_linpack_benchmarks_auf_den_energieverbrauch-report.pdf
- [Pow12] POWERS, Shawn: The Open-source Classroom: Your First Bite of Raspberry Pi. In: *Linux Journal* (2012), S. 48–54. – http://dl.acm.org/ft_gateway.cfm?id=2422332&ftid=1329297&dwn=1&CFID=403460446&CFTOKEN=89580898
- [Pre11] PREVEZANOS, Christoph: *Computer-Lexikon 2012*. München : Markt+Technik, 2011. – ISBN 9783827247285
- [Rec06] RECHENBERG, Peter: *Informatik-Handbuch*. München : Hanser, 2006. – ISBN 9783446401853
- [Sch13] SCHMIDT, Maik: *Raspberry Pi. Einstieg, Optimierung, Projekte*. Heidelberg : dpunkt, 2013. – ISBN 9783864900327
-  [scr14] Benutzerhandbuch EGM-PWM-LAN. : Benutzerhandbuch EGM-PWM-LAN, Januar 2014. – http://energenie.com/Repository/6736/EGM-PWM-LAN_manual---fe4469f6-23eb-40be-8c86-e83faf6a2c5b.pdf
- [Wei90] WEICKER, Reinhold: An Overview of Common Benchmarks. In: *Computer* (1990), S. 65–75. – http://dlojewski.dl.funpic.de/download/Diplomarbeit/Andere_Dok/Dhry_Whet/OverviewOfCommonBenchmarks.pdf