technische universität
dortmund

Bachelorarbeit

Power Management in heterogenen Multiprocessing-Systemen

Rico van Endern
147941
August 2017

Gutachter:
Prof. Dr. Jian-Jia Chen
Dr-Ing. Anas Toma

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 12
http://ls12-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Strompreise unterliegen einem konstanten Wachstum und selbst, wenn der Preis nicht der problematische Faktor ist, kann es aufgrund von Batterie- bzw. Akkukapazität trotzdem ein limitiertes Gut sein. Des Weiteren ist der direkte Stromverbrauch nicht der einzige Verbrauchsfaktor, welcher sich aus der Nutzung eines Gerätes ergibt. Auch indirekter Verbrauch, wie z.B. der Stromverbrauch der Kühlung, ist zu beachten. Somit ist Energieverwaltung bzw. das Energiesparen ein wichtiges Thema.
Auffallend sind dabei unter anderem die Möglichkeiten heterogener Systeme. Dies sind Systeme, welche über Berechnungseinheiten verschiedener Ausprägungen verfügen. Im Grunde genommen, wenn in einem Gerät zwei unterschiedliche Prozessoren oder ein Prozessor und eine spezielle funktionale Einheit verbaut wurden. Solch unterschiedliche Berechnungseinheiten sind interessant, da verschiedene Berechnungseinheiten meist auch andere Merkmale, wie z.B. Energieverbrauch und Performanz, haben. Heterogene Systeme bzw. Geräte mit einer heterogenen Berechnungslösung sind nicht nur von theoretischer Relevanz, sondern auch in der heutigen Zeit durchaus verbreitet. Viele der heutigen „Smart Home“-Controller, Medien Systeme, Spielkonsolen und Netzwerk/Speicher/Server-Infrastrukturen sowie weitere Geräte folgen einer heterogenen Architektur [28]. Dies gilt für verschiedenste Bereiche, unter anderem auch für viele der heutigen Smartphones. Bei der in Smartphones befindlichen heterogenen Berechnungslösung, greift dann unter andem der bereits erwähnte Faktor der begrenzten Akkukapazität [26].
Aufgrund der schieren Masse an unterschiedlichen Geräten reicht es bereits aus, eine kleine Verbesserung im Stromverbrauch solcher Geräte zu erwirken, um im Ganzen aufgrund der Akkumulation der Veränderung eine bemerkenswerte Verbesserung herbeizuführen. Dies ist vor allem relevant, da die Prozessoren, bei Inbetrachtziehung des direkten und indirekten Stromverbrauchs, einen beachtenswerten Verbraucher darstellen.
Dabei ist jedoch immer zu beachten, dass verschiedene Anwendungsfälle unterschiedliche

Zielsetzungen, Probleme und Limitationen haben. Zum Beispiel gibt es bei einem Smartphone, so wie bei jedem anderen interaktiven Gerät mit grafischer Benutzeroberfläche, eine absolute Nutzungsuntergrenze. Darunter wäre das Gerät so langsam, dass für den Nutzer keine flüssige Bedienung mehr möglich wäre. Trotzdem muss immer mit einem gewissen Energiebudget gerechnet werden, da die Akkulaufzeit begrenzt ist. Über dieser Nutzungsuntergrenze sollen Aufgaben natürlich weiterhin in angemessener Zeit abgearbeitet werden. Daher ist es keine sinnvolle Energiesparlösung, einfach das System auf diese Nutzungsuntergrenze zu begrenzen. Auch ist die Abwärme eher schwierig zu handhaben und es gibt viele externe Faktoren, wie z.B. Kommunikationsverfahren, welche den Energieverbrauch beeinflussen. Dies führt dazu, dass es sehr wahrscheinlich keine universal optimale Lösung gibt und ein Verfahren gefunden werden muss, welches in sich flexibel ist bzw. dynamisch an jede Problemstellung angepasst werden kann.
In dieser Arbeit wird der Fokus auf der Energieverwaltung der Berechnungslösung liegen, um erstmal dieses Problem adäquat in den Griff zu kriegen.

1.2 Hintergrund

Um ein ausreichendes Grundwissen und Verständnis der Thematik bzw. eine gemeinsame Basis zu schaffen, werden in diesem Kapitel die zugrundeliegenden Konzepte sowie die spezifisch angewandte Modellierung bzw. Betrachtungsweise erklärt. Dabei wird auf die unterschiedlichen Arten heterogener Systeme als auch auf die mathematische Repräsentation eines solchen Systems eingegangen. Gleichzeitig wird auch die mathematische Repräsentation des Energieverbrauchs betrachtet. Die in diesem Kapitel eingeführten Strukturen und Modelle werden im Rest der Arbeit vorausgesetzt.

1.2.1 Heterogene Multiprocessing-Systeme

In der Informatik-Fachliteratur gibt es unterschiedliche Definitions- und Interpretationsstufen heterogener Systeme [6, 17, 18]. Dies geht von verschiedenen Metaebenen der Hardware bis hin zur Software. Dabei ist zu beachten, dass sich diese Varianten in sich nicht widersprechen. Sie behandeln lediglich unterschiedliche Sichtweisen auf Bereiche bzw. Abstraktionsebenen der Informatik. Auch ist nicht zu vernachlässigen, dass sie ineinander integriert werden können bzw. sind. In einem verteilten heterogenen Hardwaresystem kann ein zentralisiertes heterogenes Hardwaresystem sein, auf welchem ein heterogenes Softwaresystem läuft. Diese Arbeit wird aufgrund ihres Fokus nur die Sicht auf die Hardware behandeln. Dabei wird stets von heterogenen Multiprocessing-Systemen gesprochen, da die zu behandelnden Systeme auf ihren unterschiedlichen Berechnungseinheiten unterschiedliche Prozesse laufen lassen können, welche gleichzeitig abgearbeitet werden können.

1.2.1.1 Verteilte heterogene Systeme

Verteilte heterogene Systeme sind Systeme, welche sich über mehr als ein Gerät erstrecken. Daher können in der Berechnungslösung eines verteilten heterogenen Systems verschiedene Geräte an unterschiedlichen Orten eingebunden sein. Somit ist es möglich, dass Aufgaben eines Gerätes auf andere Geräte, welche möglicherweise einen Hardwarevorteil haben, ausgelagert werden. Wenn ein Smartphone eine große komplexe Berechnung zu lösen hat, kann es diese an einen externen Server auslagern, damit dieser die Berechnung löst. Dabei ist zu beachten, dass ein Overhead nicht nur beim Senden und Empfangen entsteht, sondern auch beim Codieren und Decodieren der zu übermittelnden Berechnung. Ein solcher Overhead kann, je nachdem wie der Geschwindigkeitsvorteil bzw. die Energieersparnis ausfällt, diese zunichtemachen [6].

1.2.1.2 Zentralisierte heterogene Systeme

Zentralisierte heterogene Systeme sind Systeme, welche in einem einzelnen Gerät mindestens zwei unterschiedliche Berechnungseinheiten in ihrem Berechnungssystem integriert haben. Dies können spezielle funktionale Einheiten sein, wie z.B. Grafikkarten, auf Parallelisierung optimierte Prozessoren, Prozessoren mit einer erhöhten Anzahl bestimmter funktionaler Einheiten oder sogar so etwas wie eine „Anwendungsspezifische integrierte Schaltung“ (ASIC). Jedoch ist es auch möglich, dass mehrere Berechnungseinheiten derselben Grundfunktionalität integriert sind. In einem solchen System können Aufgaben an ihre Bedürfnisse und Voraussetzungen angepasst, auf diese Berechnungseinheiten verteilt werden. Da unterschiedliche Berechnungseinheiten meist auch verschiedene Verläufe der Energiebedürfnisse bei variierenden Nutzlasten haben, kann die Verteilung der Aufgaben an diese angepasst werden, um Energie zu sparen. Auch kann die Verteilung die Ausführung beschleunigen, wenn eine Berechnungseinheit besser für eine Aufgabe oder auch nur einen Teil einer solchen geeignet ist. Ein gutes Beispiel hierbei sind Grafikkarten. Da je nach Grafikkarte diese teilweise über mehrere tausend Berechnungskerne verfügt, können auch mehrere tausend kleine Aufgaben gleichzeitig durchgeführt werden [2, 16, 18].
Im Folgenden bezieht sich heterogenes Multiprocessing-System immer auf ein zentralisiertes heterogenes Multiprocessing-System. Dabei liegt der Fokus auf unterschiedlichen Ausprägungen derselben Art von Berechnungslösungen.

1.2.2 Systemmodel

In diesem Kapitel wird eine formale Beschreibung bzw. Darstellung für ein System und für den Energieverbrauch eines solchen Systems vorgestellt. Dies dient zur mathematischen Veranschaulichung, sowie als mathematische Basisreferenz für alle mathematischen Repräsentationen in folgenden Kapiteln. Es wurde dabei ein zum Kern der Arbeit passendes Modell erstellt. Dieses Kapitel bzw. diese Repräsentation erhebt somit keinen Anspruch

auf allumfassende Anwendbarkeit, sondern lediglich darauf, eine adäquate Repräsentation für diese Arbeit zu treffen.

1.2.2.1 Formale Beschreibung des Systemmodels

Ein berechnendes System lässt sich als eine Ansammlung von Inseln I_i darstellen. Dabei stellt i_{max_S} immer den im System S höchsten repräsentierten Insel-Index dar. An einer solchen Insel liegt eine Spannung V_i an. Diese Insel beinhalten ein bis beliebig viele Kerne c. Der höchste in der Insel i repräsentierte Kern-Index wird durch c_{max_i} dargestellt. cR_{max_i} stellt den höchsten Kern-Index dar, für welchen gilt, dass dessen Frequenz unabhängig von anderen Kernen ist. Dies dient zur Modellierung von verwandten Kernen, welche auf derselben Frequenz laufen. Für die von einander unabhängigen Kerne cR gilt eine separate Indexierung. Das bedeutet, dass wenn auf einer Insel die Kerne 0 und 1 sowie die Kerne 2 und 3 verwandt sind, aus $C_{0,2}$ mit der cR-Indexierung der Kern $C_{0,1}$ bzw. $C0,1_R$ wird. Dies ist entweder direkt aus dem Kontext ersichtlich oder durch ein R im Index der Zahl zu kennzeichnen. Kerne haben eine Frequenz $F_{i,c}$, welche im Frequenzbereich $FR_{i,c}(v)$ liegt. Dieser Frequenzbereich hängt von der an der Insel i anliegenden Spannung v ab. Ein solcher Frequenzbereich hat ein Minimum $Fm_{i,c}$ und ein Maximum $FM_{i,c}$. Zu jedem Kern einer Insel gehört eine Taskliste $T_{i,c}$ an, welche aus beliebig vielen Tasks τ_t besteht. Dabei stellt $t_{max_{i,c}}$ den höchsten Task-Index dar, welcher in der Taskliste des zugehörigen Kerns vertreten ist. Ein solches System ist in Abbildung 1.1 dargestellt.

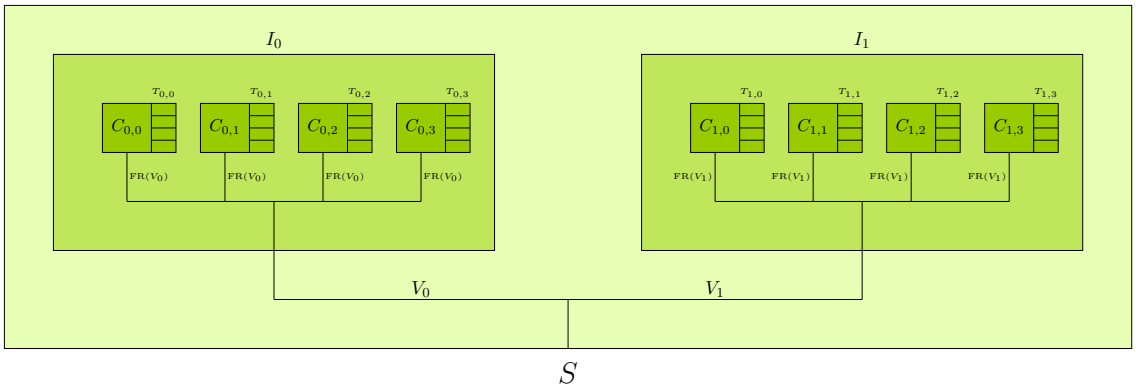


Abbildung 1.1: Formale Darstellung eines Systems (e.D.)

Legende zu Abbildung 1.1:

S	= System
I_i	= Insel i
V_i	= Anliegende Spannung der Insel i
$C_{i,c}$	= Kern c der Insel i
$F_{i,c}$	= Frequenz des Kerns c der Insel i
$Fm_{i,c}$	= Minimale Frequenz
$FM_{i,c}$	= Maximale Frequenz
$FR_{i,c}(v)$	= Frequenzbereich des Kerns c auf Insel i bei anliegender Spannung v
$P_{i,c}(f)$	= Energieverbrauch des Kerns c auf Insel i mit Frequenz f
$T_{i,c}$	= Tasks des Kerns c auf Insel i

Diese Darstellung gilt ebenso für Systeme mit nur einer Insel, wie auch für Inseln mit nur einem Kern. Durch die Darstellung der Kernfrequenz als Frequenzbereich, welcher abhängig ist von der an der Insel anliegenden Spannung, können sowohl Systeme dargestellt werden, bei denen alle Kerne in einer Insel dieselbe Frequenz haben, als auch welche, bei denen jeder Kern separat konfigurierbar ist. Somit ist diese Systemdefinition sehr flexibel in ihrer Anwendung. Zwar ist sie aufgrund der Modellierung der Tasklisten nicht vollständig realitätsgetreu, jedoch ist jede reale Struktur in diese übertragbar. Somit bietet die Systemdefinition eine angemessen präzise Repräsentation, was moderne Berechnungslösungen betrifft.

Regeln und Restriktionen

Die Formel 1.1 deklariert die benutzten Hilfsvariablen:

$$n, i, c, c_1, c_2, t, cR \in \mathbb{N}_0, f_2, v_1 \in \mathbb{R} \tag{1.1}$$

Die Formel 1.2 sagt aus, dass die Spannung V_i eine reelle Zahl ist:

$$V_i := v_0 \in \mathbb{R} \tag{1.2}$$

Die Formel 1.3 sagt aus, dass das Frequenzminimum $Fm_{i,c}$ und das Frequenzmaximum $FM_{i,c}$ voneinander unabhängige reelle Zahlen sind:

$$Fm_{i,c} := f_0 \in \mathbb{R},\ FM_{i,c} := f_1 \in \mathbb{R} \tag{1.3}$$
$$wobei\ gilt,\ dass\ Fm_{i,c} \leq FM_{i,c}$$

Die Formel 1.4 sagt aus, dass der Energievierbrauch $P_{i,c}(f_2)$ eine reelle Zahl ist:

$$P_{i,c}(f_2) := p \in \mathbb{R} \tag{1.4}$$

Die Formel 1.5 sagt aus, dass die theoretische Maximalfrequenz $FM_{i,c}(v_1)$ in Abhängigkeit zur anliegenden Spannung v_1 eine reelle Zahl ist:

$$FM_{i,c}(v_1) := f_1 \in \mathbb{R} \tag{1.5}$$

Die Formel 1.6 definiert die Funktion $R(i, c_1, c_2)$, welche angibt, ob zwei Kerne voneinander abhängig bzw. verwandt sind:

$$R(i, c_1, c_2) := b \in \{0, 1\} \tag{1.6}$$

Die Formel 1.7 sagt aus, dass die Frequenz eines Kerns $F_{i,c}$ im gültigen Frequenzbereich des Kerns $FR_{i,c}$ für die akut anliegende Spannung v_1 liegt.

$$F_{i,c} := \begin{cases} F_{i,c_1} & falls\ R(i, c, c_1) = 1 \\ f_3 \in FR_{i,c}(v_1) & sonst \end{cases} \tag{1.7}$$

Die Formel 1.8 sagt aus, dass die Frequenz eines verwandten Kern-Blocks $F_{i,cR}$ im gültigen Frequenzbereich des Kerns $FR_{i,cR}$ für die akut anliegende Spannung v_1 liegt:

$$F_{i,cR} \in FR_{i,c}(v_1) \tag{1.8}$$

Die Formel 1.9 sagt aus, dass eine Taskliste $T_{i,c}$ eine Liste aus Tasks τ_t ist:

$$T_{i,c} := \{\tau_0,\ ...,\ \tau_{t_{max_{i,c}}}\} \tag{1.9}$$

Die Formel 1.10 sagt aus, dass im Frequenzbereich $FR_{i,c}(v)$ mit „0“ die Deaktivierung eines Kerns abgebildet wird. Abgesehen vom Nullzustand liegt die Frequenz zwischen dem Frequenzminimum $Fm_{i,c}$ und dem Frequenzmaximum $FM_{i,c}$. Dabei werden nur die dem System möglichen Frequenzen abgebildet. Sollte das theoretische Frequenzmaximum in Abhängigkeit von der Spannung $FM_{i,c}(v_1)$ niedriger sein als das Frequenzmaximum des Kerns $FM_{i,c}$, wird dieser auf das theoretische Frequenzmaximum $FM_{i,c}(v_1)$ begrenzt.

$$FR_{i,c}(v_2) := \begin{cases} \{0, Fm_{i,c},\ ...,\ FM_{i,c}(v_2)\} & falls\ FM_{i,c}(v_2) < FM_{i,c} \\ \{0, Fm_{i,c},\ ...,\ FM_{i,c}\} & sonst \end{cases} \tag{1.10}$$

Die Formel 1.11 sagt aus, dass eine Insel aus Tupeln besteht. Diese Tupel wiederum bestehen jeweils aus dem Kern $C_{i,c}$, dessen aktueller Frequenz $F_{i,c}$, dem Frequenzminimum $Fm_{i,c}$ und Frequenzmaximum $FM_{i,c}$, sowie dem Energieverbrauch $P_{i,c}$ und der zugehörigen Taskliste $T_{i,c}$.

$$I_i := \{(C_{i,0}, F_{i,0}, Fm_{i,0}, FM_{i,0}, P_{i,0}, T_{i,0}),\ ..., (C_{i,\kappa}, F_{i,\kappa}, Fm_{i,\kappa}, FM_{i,\kappa}, P_{i,\kappa}, T_{i,\kappa})\} \quad mit\ \kappa = c_{max_i} \tag{1.11}$$

Die Formel 1.12 sagt aus, dass ein System S aus einer Liste an Tupeln besteht. Jedes dieser Tupel wiederum besteht aus einer Insel I_i und der dazugehörigen anliegenden Spannung V_i.

$$S := \{(I_0, V_0),\ ...,\ (I_\mu, V_\mu)\}\ mit\ \mu = i_{max_S} \tag{1.12}$$

1.2.2.2 Formale Beschreibung des Energiemodels

Der Energieverbrauch eines Systems ist aus unterschiedlichen Elementen zusammengesetzt:

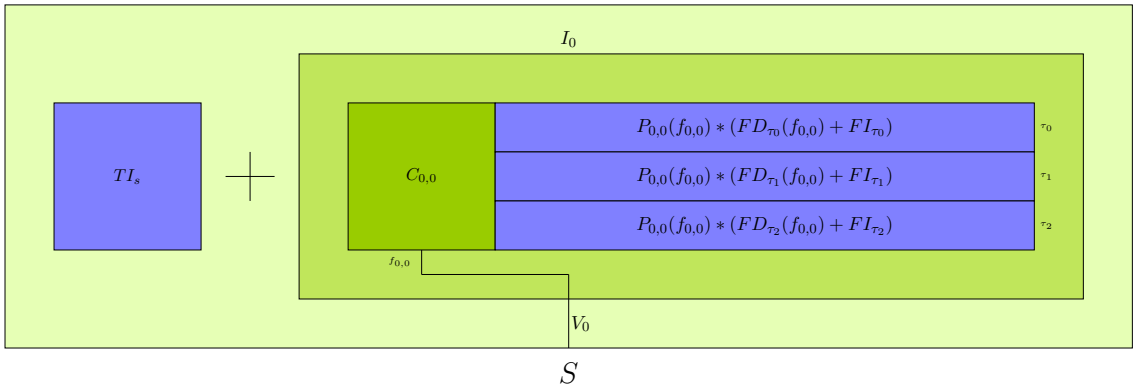


Abbildung 1.2: Formale Darstellung des Energieverbrauches eines Systems (e.D.)

Legende zu Abbildung 1.2:

E_s	= Gesamter Energieverbrauch eines Systems s
FD_τ	= Frequenzabhängige Ausführungszeit eines Tasks τ
FI_τ	= Frequenzunabhängige Ausführungszeit eines Task τ
$P_{i,c}(f)$	= Energieverbrauch des Kerns c auf Insel i mit Frequenz f
TI_s	= Taskunabhängiger Energieverbrauch eines Systems s
FC_s	= Frequenzkonfiguration der Kerne des Systems s
TM	= Taskliste

Regeln und Restriktionen

Die Formel 1.13 deklariert die benutzten Hilfsvariablen:

$$f_0, f_1 \in \mathbb{R} \tag{1.13}$$

Die Formel 1.14 sagt aus, dass die frequenzabhängige Laufzeit $FD_\tau(f_0)$ und die frequenzunabhängige Laufzeit FI_τ eines Tasks τ bei der Frequenz f_0 jeweils voneinander unabhängige Werte aus den reellen Zahlen sind:

$$FD_\tau(f_0) := t_1 \in \mathbb{R},\ FI_\tau := t_2 \in \mathbb{R} \tag{1.14}$$

Die Formel 1.15 sagt aus, dass der Energieverbrauch $P_{i,c}(f_1)$ eines Kerns c der Insel i bei der Frequenz f_1 in den reellen Zahlen liegt:

$$P_{i,c}(f_1) := p \in \mathbb{R} \tag{1.15}$$

Die Formel 1.16 sagt aus, dass der taskunabhängige Energieverbrauch eines Systems TI_s in den reellen Zahlen liegt:

$$TI_s := e \in \mathbb{R} \tag{1.16}$$

Die Formel 1.17 sagt aus, dass die Frequenzkonfiguration FC_s ein Liste ist, welche Kernen zugehörige Frequenzen $f_{i,c}$ beinhaltet:

$$FC_s := \{f_{0,0}, ..., f_{\mu,\kappa}\},\ mit\ \mu = i_{max_S},\ \kappa = c_{max_\mu} \tag{1.17}$$

Die Formel 1.18 sagt aus, dass die Taskliste $TM_{i,c}$ eine Liste aus Tasks τ_t ist:

$$TM_{i,c} := \{\tau_0\ ..\ \tau_{t_{max}}\} \tag{1.18}$$

Die Formel 1.19 stellt den Gesamtenergieverbrauch eines Systems dar. Dieser ergibt sich aus dem taskunabhängigen Energieverbrauch und der Aufsummierung des taskabhängigen Energiebedarfs. Dabei wird der Energieverbrauch jedes Tasks, jedes Kerns und jeder Insel addiert. Dieser Energieverbrauch ergibt sich aus dem frequenzabhängigen Energieverbrauch des Kerns, auf dem der Task läuft. Dabei wird als Frequenz die Frequenz aus der aktuellen Frequenzkonfigurationen herangezogen, welche mit der Laufzeit des Tasks multipliziert wird. Die Laufzeit des Tasks setzt sich aus der frequenzunabhängigen und der frequenzabhängigen Laufzeit zusammen.

$$E_{s,FC_s} = TI_s + \sum_{i=0}^{i_{max_S}} \sum_{c=0}^{c_{max_i}} \sum_{\tau \in TM_{i,c}} (P_{i,c}(f_{i,c}) * (FD_\tau(f_{i,c}) + FI_\tau))\ mit\ f_{i,c} \in FC_s \tag{1.19}$$

1.3 Zielsetzung und Aufbau der Arbeit

Die Zielsetzung dieser Arbeit ist es, sich mit Multiprocessing in heterogenen bzw. wie bereits erwähnt in zentralisierten heterogenen Systemen auseinander zu setzen. Ein großer Aspekt ist es, ein universal anwendbares Messeverfahren des Energieverbrauchs für die verschiedenen Konfigurationsmöglichkeiten der Berechnungslösungen zu erstellen. Der experimentellen Aufbau sowie die dazugehörige Hardware wird im Kapitel 2 behandelt. Dabei wird zwar auf die für diese Arbeit gewählte Hardware und die Begründung hinter dieser Auswahl eingegangen, jedoch ist die Grundstruktur auf jede Hardware anwendbar, für welche es eine geeignete Energie-Überwachungshardware gibt. Um reale Nutzungsszenarien nachzubilden und konsistent wiederholbar zu messen, werden in Kapitel 3 Benchmarks ausgewählt. Ebenfalls wird beschrieben, was die einzelnen Elemente der ausgewählten Benchmark Suites bzw. Benchmark Sammlungen repräsentieren. In Verbindung mit diesen Benchmarks wird eine Benchmark-Verwaltungssoftware geschrieben, welche sehr modular gehalten ist, sodass weitere Benchmarks problemlos hinzugefügt und Experimente wiederholt werden können. Die Entwicklung und Anwendung dieser Software wird in Kapitel 4 beschrieben. Danach wird im Kapitel 5 auf ein paar ausgewählte grundlegende Techniken zum Energiesparen eingegangen. Dabei werden die aus den Benchmarks gewonnenen Daten in Bezug gesetzt und ausgewertet. Daraufhin wird die untersuchte Haupttechnik, das DVFS, also das dynamische Anpassen von Frequenzen im Allgemeinen und anhand von ein paar Beispielen, behandelt.

Im Kontext betrachtet ist diese Arbeit eine Einführung in das Themengebiet der heterogenen Berechnungslösung mit Fokus auf deren Energieverlauf sowie dem strukturierten Erarbeiten von Messmöglichkeiten, um eben solche Energieverläufe analysieren zu können. Dabei wurde viel Wert darauf gelegt, dass die Ergebnisse wiederholbar sind und darauf folgende Arbeiten auf dieser Basis aufbauen können.

Kapitel 2

Experimentaler Aufbau

Das Ziel des Versuchsaufbaus ist es, den Energieverbrauch eines heterogenen Systems zu messen und diesen bei wechselnden Aufgaben und Auslastungen zu überwachen. Eine kombinierte Lösung, wie z.B. ein Odroid-XU3[14], ist dafür natürlich eine einfache und geeignete Lösung, da sie sowohl ein heterogenes System ist, als auch direkt die Überwachungsmöglichkeiten für den Energieverbrauch der einzelnen Komponenten mit sich bringt. Im Speziellen können somit die beiden Prozessoren, also der Cortex^{TM}-A15(big) und Cortex^{TM}-A7(LITTLE), sowie die Grafiklösung und der DRAM getrennt voneinander überwacht werden. Dadurch können die Auswirkungen von Änderungen nicht nur anhand des Gesamtenergieverbrauches ausgewertet werden, sondern auch anhand ihrer spezifischen Auswirkung auf eben diese Komponenten.

Diese Lösung hat jedoch auch einige Nachteile. Die Beschaffung von Odroid-XU3s ist eher schwierig, da diese nicht mehr hergestellt werden. Auch wäre eine für solche Hardware erzeugte Lösung nicht universal anwendbar, da nicht jedes Gerät solche internen Energiesensoren hat und es auch nicht von jeder heterogenen Berechnungslösung eine Hardwareimplementierung gibt, welche solche Sensoren beherbergt. Somit ist eine externe bzw. dedizierte Überwachungshardware die adäquatere Lösung. Der Hauptnachteil bei der Nutzung solcher externer bzw. dedizierter Überwachungshardware ist, dass mehr Daten gesammelt werden müssen. Dieser besteht, da die Daten, welche gesammelt werden können, aufgrund der fehlenden Trennung der einzelnen Komponenten wesentlich unpräziser sind. Dies führt dazu, dass jedwede Fluktuation einzelner Komponenten mit im gemessenen Ergebnis auftauchen, genau wie externe Einflüsse, wie z.B. der Stromverbrauch an den I/O Ports. Jedoch ermöglicht dieses Verfahren die Analyse von fast jeder Hardware, für welche es eine angemessene Überwachungshardware gibt. Auch ist zu bedenken, dass es häufig unkomplizierter umzusetzen ist, extern zusätzliche Überwachungshardware zu erstellen und diese anzuschließen, als die zu untersuchende Hardware selbst umzubauen oder in diese direkt einzugreifen.

2.1 Versuchshardware

Als Versuchshardware wurde ein „Odroid-XU4“ (siehe Abb. 2.1) benutzt. Dieser verwendet als CPU einen „Samsung Exynos 5 Octa“, auch bekannt als der „Samsung Exynos 5422“, welcher ein Prozessor mit ARM „big.LITTLE“-Architektur ist und somit aus unterschiedlichen Berechnungseinheiten besteht. Die ARM „big.LITTLE“-Architektur des „Odroid-XU4“ stellt eine heterogene Berechnungslösung dar. Verbaut sind ein „ARM Cortex-A7“, welcher vier Kerne hat, die von 200MHz bis zu 1400MHz taktbar sind, und ein „ARM Cortex-A15“, welcher ebenfalls vier Kerne hat, jedoch von 200MHz bis zu 2000MHz taktbar ist. Der „ARM Cortex-A7“ stellt dabei den „LITTLE“-Teil und der „ARM Cortex-A15“ den „big“-Teil der „big.LITTLE“-Architektur dar. Die Entscheidung für den „Odroid-XU4“ wurde getroffen, da er mit der „big.LITTLE“-Architektur eine der verbreitetsten heterogenen Berechnungslösungen repräsentiert [15]. Ein weiterer Faktor war es, dass die Odroid-Reihe bereits in verwandten Arbeiten über das selbe Themengebiet oder ähnliche Themengebiete als Basis benutzt wurde und somit die Vergleichbarkeit erhöht wird [5].

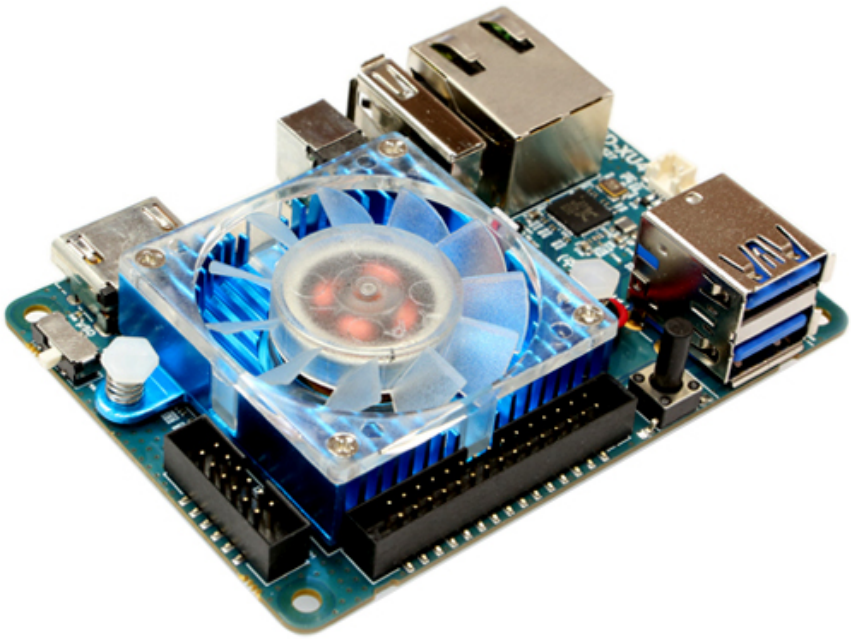


Abbildung 2.1: Odroid-XU4 [1]

2.2 Überwachsungshardware

Da der „Odroid-XU4“, wie bereits erwähnt, über keine internen Energiesensoren verfügt, muss der Energieverbrauch extern überwacht werden. Hierfür wird ein „Odroid Smart Power“ (siehe Abb. 2.2) verwendet. Dieses Gerät wird zwischen Netzteil und „Odroid-XU4“

[1] Quelle:`http://www.hardkernel.com/main/_Files/prdt/2017/201704/201704250351342687.jpg`

angeschlossen. Somit kann sowohl die Stromzufuhr gesteuert, also das zu überwachende Gerät an- und ausgeschaltet, als auch der Energieverbrauch gemessen werden. Hierbei werden die Werte für Volt, Ampere und Watt nachverfolgt, sowie beim Start eines Logging-Vorganges auch die Wattstunden. Diese Werte werden in Echtzeit über das eingebaute Display dargestellt, sind jedoch auch per USB von einem separaten Computer auslesbar. Die Wiederholungsrate beim Auslesen ist hierbei 10Hz mit einer Messungenauigkeit von ca. zwei Prozent [13]. Ein Vorteil des „Odroid Smart Power“ ist auch, dass der Ausgang nicht an ein bestimmtes Interface, wie z.B. USB, gebunden ist, da der Stromausgang über zwei Klemmverbindungen realisiert wurde. An diese kann jedes beliebige Gerät angeschlossen werden, welches mit 3-5,25 Volt bei bis zu 5 Ampere Gleichstrom arbeitet. Im Fall des „Odroid-XU4“ wird das beim „Odroid Smart Power“ mitgelieferte USB Verbindungskabel angeschlossen.

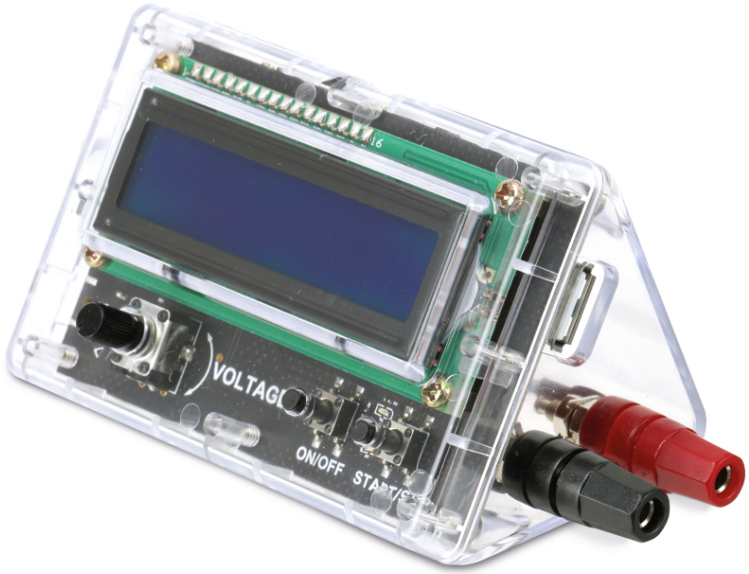


Abbildung 2.2: Odroid Smart Power [2]

2.3 Aufbau

Der Versuchsaufbau besteht aus der Versuchshardware, der Überwachungshardware und einer Verwaltungshardware. Die Überwachungshardware versorgt die Versuchshardware mit Energie. Als Überwachungshardware wurde, wie bereits erwähnt, ein „Odroid Smart Power“ gewählt und als Versuchshardware ein „Odroid-XU4“. Dabei werden alle wichtigen Energieverbrauchsmerkmale wie Volt, Ampere und Watt gemessen. Diese Werte werden von

[2] Quelle: https://www.pollin.de/media/image/e6/37/a5/I810337-1-ODROID-SMART-POWER.jpg

der Verwaltungshardware, welche jeder beliebige Computer sein kann, bei der Überwachungshardware ausgelesen. Nachdem die durchzuführenden Benchmarks beendet wurden, werden die dabei gesammelten Daten an die Verwaltungshardware übertragen. Danach werden die Energieverbrauchsmerkmale und die Benchmark-Daten auf der Verwaltungshardware zusammengeführt und ausgewertet.

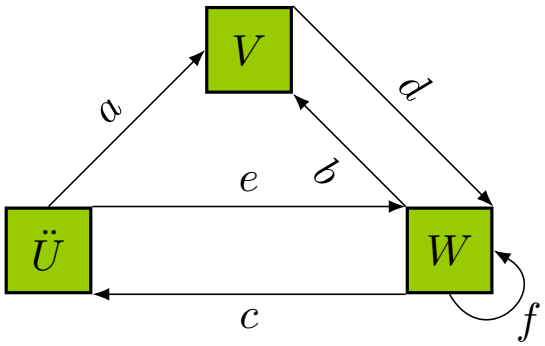


Abbildung 2.3: Versuchsaufbau (e.D.)

Legende zu Abbildung 2.3:

V = Versuchshardware
$\ddot{U}$ = Überwachungshardware
W = Verwaltungshardware
a = $\ddot{U}$ versorgt V mit Strom
b = W lässt V die Benchmarks starten
e = W fragt bei $\ddot{U}$ die Energieverbrauchsmerkmale an
e = V übermittelt die Benchmarkdaten an W
d = $\ddot{U}$ übermittelt die Energieverbrauchsmerkmale an W
f = Wwertet die gesammelten Daten aus

Ablauf

1. Start des Auslesens der Überwachungshardware durch die Verwaltungshardware.
2. Start der auszuführenden Benchmarks auf der Versuchshardware.
3. Ende der ausgeführten Benchmarks auf der Versuchshardware.
4. Ende des Auslesens der Überwachungshardware durch die Verwaltungshardware.
5. Übertragen der Benchmark-Daten von der Versuchshardware auf die Verwaltungshardware.
6. Zusammenführen und Auswerten der Daten auf der Verwaltungshardware.

Dieser Grundaufbau ist wie bereits erwähnt mit jeder Versuchshardware möglich, für welche es angemessene Überwachungshardware gibt.

Kapitel 3

Benchmarks

Damit der Energieverbrauch sowie dessen Veränderung ausgewertet und verglichen werden können, bedarf es einer festen gemeinsamen Metrik. Um eine solche einheitliche Mess- und Datengrundlage zu schaffen, werden die gleichen Benchmarks ausgeführt. Dabei wird deren Ausführungs-Dauer sowie ihre Energieverbrauchsmerkmale gemessen. Dies funktioniert nur, wenn die Benchmarks wiederholbare Ergebnisse erzielen, d.h. mit festen Abfolgen bzw. Daten arbeiten. Ein Benchmark, welcher auf Zufallszahlen basiert, ist für eine solche Anwendung somit verhältnismäßig ungeeignet. In dieser Arbeit wurden als möglichst repräsentative Benchmarks MiBench und PARSEC gewählt.

3.1 Methodik

Die angewandte Testmethodik ist weitestgehend eine Exhaustionsmethode, d.h. jede mögliche Konfiguration bzw. Variation wird somit durchgeführt. Dabei wird jeder zugrundeliegende Benchmark für jede Kern-/Frequenzkonfiguration durchgeführt und ausgewertet. Daraus ergibt sich, dass jeder Benchmark d mal durchgeführt wird, wobei d in Formel 3.1 definiert ist.

$$d := kv * fv * w \; mit \; kv, fv, w \in \mathbb{N}_0 \tag{3.1}$$

In der Formel 3.1 steht kv für die Kernvariationen, also die Anzahl der zu durchlaufenden Kern-Konfigurationen. Bei einem Durchlauf ohne „Core Deactivation“ wäre dies 1. Mit „Core Deactivation“ entspricht kv der Anzahl der möglichen Statuspermutationen. Für den Odroid-XU4 ergibt dies mit der simplen Berechnung der Variationen mit Wiederholung 2^8, also 256 Konfigurationen. Die 2^8 ergibt sich aus der Anzahl an möglichen Kernzuständen „online“ und „offline“, also 2 und der Anzahl der zu konfigurierenden Kerne, also bei unserer Versuchshardware dem „Odroid-XU4“ 8. Die Variable fv steht für die Frequenzvariationen, also die Anzahl der zu durchlaufenden Frequenzkonfigurationen. Diese ist jedoch abhängig von der Art des Durchlaufes und interner Faktoren. Viele Prozessoren erlauben nur fest definierte Frequenzen, andere sind in der Konfiguration wesentlich freier. Bei einer frei

konfigurierbaren Frequenz kann die Anzahl der Frequenzschritte fs eines Kerns mit der Formel 3.2 berechnet werden.

$$fs_{i,c} := \left\lceil \frac{FM_{i,c} - Fm_{i,c}}{s} \right\rceil + 1 \quad s \in \mathbb{N} \tag{3.2}$$

Dabei ist s die angegebene Schrittgröße. $FM_{i,c} - Fm_{i,c}$ gibt die Differenz zwischen Frequenzmaximum und dem Frequenzminimum an, also die abzudeckende Frequenzspanne. Geteilt durch s erhalten wir die Anzahl der möglichen Frequenzschritte in dieser Frequenzspanne. Dies muss aufgerundet werden, da bei nicht vollständig vollzogenen Frequenzschritten die Frequenz auf das Frequenzmaximum gesetzt wird. Die +1 wird benötigt, damit auch der erste Schritt, also das Frequenzminimum, selbst enthalten ist. Bei einem System mit fest vordefinierten Frequenzen ist fs wie in Formel 3.3 definiert.

$$fs_{i,c} = x \in \mathbb{N} \tag{3.3}$$

Da bis hierhin nur die Frequenzvariation einzelner Kerne betrachtet wurde, wird dies in Formel 3.4 noch auf alle Kerne erweitert.

$$Fs := \{fs_{0,0}, ..., fs_{i_{max_S}, c_{max_i}}\} \tag{3.4}$$

Fs stellt die Menge der Anzahlen der Frequenzvariationen aller Kerne eines Systems dar. Die Durchlaufarten sind:

- Einheitlicher Anstieg: Dabei steigen alle Kerne gleichzeitig an. Wenn die Frequenz um eine Schrittgröße erhöht wird, wird dies bei allen Kernen gemacht. Sollte ein Kern sein Frequenzmaximum bereits erreicht haben, wird dieser ignoriert. Dabei ist fv wie in Formel 3.5 definiert.

  $$fv := max(Fs) \tag{3.5}$$

  fv entspricht dem Maximum von Fs, da solange iteriert wird, bis alle ihr Frequenzmaximum erreicht haben. Dies dauert offensichtlich am längsten bei dem Kern mit der höchsten Anzahl an Frequenzvariationen.

- Alle Variationen: Alle in der Schrittgröße möglichen Frequenzvariationen werden zwischen Minimal- und Maximalfrequenz eines jeden Kerns durchlaufen. Dabei ist fv wie in Formel 3.6 definiert.

  $$fv := \prod_{i=0}^{i_{max_S}} \left(\prod_{c=0}^{cR_{max_i}} fs_{i,c} \right) \tag{3.6}$$

In der Formel 3.1 steht w für die Anzahl der Wiederholungen. Je größer also w gewählt wird, um so öfter wird der Benchmark durchgeführt und desto größer wird der erlangte Datensatz. Somit kann mit dem Parameter w die Präzision bzw. Aussagekraft des Datensatzes erhöht werden.

Odroid-XU4

Im Fall des „Odroid-XU4“ wird beim „cpufreq-dt“ Skalierungstreiber eine feste Anzahl Frequenzvariationen vorgegeben.

Für den „Cortex-A7“, also den „LITTLE“-Teil der Berechnungslösung, sind dies 200MHz, 300MHz, 400MHz, 500MHz, 600MHz, 700MHz, 800MHz, 900MHz, 1000MHz, 1200MHz, 1300MHz, 1400MHz [3]. Somit ergibt sich für die Anzahl der Frequenzschritte insgesamt $fs_{0,0} = 12$.

Für den „Cortex-A15“, also den „big“-Teil der Berechnungslösung, sind dies 200MHz, 300MHz, 400MHz, 500MHz, 600MHz, 700MHz, 800MHz, 900MHz, 1000MHz, 1100MHz, 1200MHz, 1300MHz, 1400MHz, 1500MHz, 1600MHz, 1700MHz, 1800MHz, 1900MHz, 2000MHz. Somit ergibt sich für die Anzahl der Frequenzschritte insgesamt $fs_{1,0} = 19$.

In der folgenden Rechnung wird von zehn Wiederholungen ausgegangen.

Bei einheitlichem Anstieg:

$$Fs = \{fs_{0,0}, ..., fs_{i_{max_S}, c_{max_i}}\} = \{12, 19\} \tag{3.7}$$

$$fv = max(Fs) = max(\{12, 19\}) = 19 \tag{3.8}$$

Ohne Kernvariation:

$$d = kv * fv * w = 1 * 19 * 10 = \underline{190} \tag{3.9}$$

Mit Kernvariation:

$$d = kv * fv * w = 256 * 19 * 10 = \underline{48640} \tag{3.10}$$

Und bei allen Variationen:

$$fv := \prod_{i=0}^{i_{max_S}} \left(\prod_{c=0}^{cR_{max_i}} fs_{i,c} \right) = 12 * 19 = 228 \tag{3.11}$$

$$\tag{3.12}$$

Ohne Kernvariation:

$$d = kv * fv * w = 1 * 228 * 10 = \underline{2280} \tag{3.13}$$

Mit Kernvariation:

$$d = kv * fv * w = 256 * 228 * 10 = \underline{583680} \tag{3.14}$$

Bei Benchmarks, welche eine konfigurierbare Thread-Anzahl haben, wird lediglich die vom Benchmark vorgegebene Standard Thread-Anzahl überprüft. Dies wäre ein Faktor, dessen Einfluss in späteren Arbeiten untersucht werden könnte.

[3] Hierbei ist es kein Tippfehler, dass 1100MHz fehlt, da diese Frequenz vom Treiber nicht angeboten wird.

3.2 Szenario

Das Ablaufszenario besteht aus zwei Schritten. Zuerst werden die im folgenden Abschnitt aufgelisteten Benchmarks, wie im Abschnitt 3.1 beschrieben, durchgeführt. Danach müssen die lokalen Ablaufdaten der Versuchshardware an die Verwaltungshardware übertragen werden. Diese beinhalten Zeitstempel, welche die Start- und Ende-Zeiten eines jeden Durchlaufs darstellen. Auch ist die benötigte reale Berechnungszeit eines jeden Benchmarks enthalten. Auf der Verwaltungshardware kann der erhaltene Datensatz mit den gesammelten Energiewerten der Überwachungshardware zusammengeführt und ausgewertet werden. Das angewandte Testszenario kann ohne Änderung auf jede beliebige Hardware übertragen werden, welche mit dem „Odroid Smart Power“ kompatibel ist. Das Verfahren kann auch an jede Hardware angepasst werden, für welche es auslesbare Energie-Überwachungshardware gibt.

3.3 Benchmark-Software

Bei der Wahl der Software, anhand welcher gemessen werden soll, geht es um unterschiedliche Faktoren, welche erfüllt werden müssen. Diese Arbeit sollte am Ende mit den Ergebnissen anderer Arbeiten[4] vergleichbar sein. Gemeint ist damit, dass diese nicht nur anhand der Modelle, welche als Erud der Arbeiten entstanden sind, sondern eben auch an den harten Daten und Fakten verglichen werden können. Spezifische Informationen und Ergebnisse sind vor allem entweder wesentlich einfacher oder überhaupt erst vergleichbar, wenn die gleichen Versuchsdaten zugrunde liegen. Um dies zu gewährleisten, wurde MiBench und PARSEC ausgewählt. MiBench ist ein Benchmark-Suite von der University of Michigan, welcher spezifische Anwendungsbereiche abbildet [12]. PARSEC ist ein Multithread fokussierter Benchmark, welches z.B. bei Intel, Princeton University, Cambridge University, Georgia Tech oder dem Massachusetts Institute of Technology MIT in der Forschung und somit auch für viele wissenschaftliche Arbeiten benutzt wird [1]. Ein weiterer Faktor war es nicht nur eine Problemstellung zu wählen, welche an das Ziel angepasst einen besten und schlechtesten Fall darstellt, sondern es war vielmehr wichtig, einen möglichst universellen Anwendungsfall abzubilden. Dieser sollte natürlich auch die besten und schlechtesten Fälle enthalten. Deshalb wurden im Gegensatz zu vielen anderen Arbeiten nicht nur die Paradebeispiele aus diesen Benchmark Suites gewählt, sondern alle darin enthaltenen Benchmarks. Oder um präziser zu sein: Es wurden alle mit der ausgewählten Hardware kompatiblen Benchmarks aus den Benchmark Suites ausgewählt.

[4] Wie z.B. in [2]

3.3.1 MiBench

Der Benchmark Suite MiBench [12] der Universität von Michigan ist in mehrere Workload-Gruppen unterteilt, welche verschiedene reale Anwendungsbereiche abbilden. Durch diese Abdeckung unterschiedlicher Gebiete und durch die wohl getrennte Struktur der einzelnen Benchmarks eignet sich MiBench gut zur Repräsentation dieser Gebiete und zur allgemeinen Abdeckung unterschiedlicher Workloads bzw. Workloadarten.

3.3.1.1 Automotive

Unter der Kategorie „Automotive“ sind Benchmarks zusammengefasst, welche häufig in Automobilen und deren Fahrttechnik benutzt werden.

- Basicmath: Führt einfache mathematische Berechnungen durch, welche meist keine dedizierte Hardwareunterstützung haben und somit nicht durch spezielle funktionale Einheiten beschleunigt werden. Dadurch werden viele mathematische Berechnungen abgedeckt, welche regelmäßig in diversen Anwendungen durchgeführt werden.

- Bitcount: Überprüft die Bit-Manipulationsfähigkeiten des ausführenden Prozessors anhand von sieben unterschiedlichen Algorithmen.

- QSort: Sortiert einen gegebenen Datensatz mit dem Quicksortalgorithmus, womit ein weit verbreiteter Sortieralgorithmus repräsentiert wird.

- SUSAN: „Smallest Univalue Segment Assimilating Nucleus“ (SUSAN) ist eine Bilderkennungssoftware, welche Kanten und Ecken in Bildern erkennt. Dadurch wird ein Element der Bilderkennung repräsentiert, welches z.B. in der Produktionskontrolle, sowie automatisierter räumliche Erkennung, wie z.B. bei der Parkautomatisierung, benutzt wird.

3.3.1.2 Consumer Devices

Unter der Kategorie „Consumer Devices“ sind Benchmarks zusammengefasst, welche die Nutzung moderner Multimedia-Geräte repräsentieren.

- JPEG: JPEG ist ein Bildformat, welches intern unterschiedliche Codierungsverfahren erlaubt. Dieser Benchmark berechnet die Codierung und Decodierung von Bildern im JPEG-Format. Dabei wird das Anzeigen von Bildern repräsentiert.

- LAME: „Lame Ain’t an MP3 Encoders“ (LAME) ist eine .mp3-Codierungsbibliothek, welche .wav-Dateien in .mp3-Dateien umwandelt. Dieser Benchmark repräsentiert das Umwandeln von Musik.

- MAD: „MPEG Audio Decoder“ (MAD) ist ein .mpeg-Audiodecodierer, welcher .mp3-Dateien in .mpeg-Dateien umwandelt. Dieser Benchmark repräsentiert das Umwandeln von Musik.

- TIFF: Ist eine Sammlung an Tools zur Verarbeitung und Umwandlung von Bildern im „Tagged Image File Format“ (.tiff). Dabei wird sowohl die Qualitätsanpassung als auch der Farbschemenwechsel berechnet. Dies repräsentiert Bearbeitung, Darstellung und Komprimierung von Bildern.

- Typeset: Typeset ist ein Typesetting-Tool, welches HTML als Input akzeptiert. Dieser Benchmark repräsentiert eine der Standardaufgaben eines Webbrowsers.

3.3.1.3 Network

Unter der Kategorie „Network“ sind Benchmarks zusammengefasst, welche typische Grundalgorithmen der modernen Netzwerktechnik repräsentieren.

- Dijkstra: Berechnet den kürzesten Pfad in einem Graphen, wie es z.B. bei einigen Routing basierenden Algorithmen benutzt wird.

- Patricia: Berechnet das Ablaufen der Baumstruktur Patricia-Trie. Dies ist eine besonders für Routing-Tabellen geeignete Baumstruktur.

3.3.1.4 Office

Unter der Kategorie „Office“ sind Benchmarks zusammengefasst, welche die alltäglichen Berechnungen in einer Büroumgebung repräsentieren.

- Ghostscript: „Ghostscript“ ist ein PostScript-Interpreter. Postscript wird z.B. von Druckern und einigen Grafik-Design-Programmen benutzt.

- ISpell: „International Spell“ (ISpell) ist ein Rechtschreibkontrolle-Tool, welches international, also in mehreren Sprachen, anwendbar ist und repräsentiert somit die Rechtschreibkontrolle eines alltäglichen Textbearbeitungsprogrammes.

- rsynth: Rsynth ist eine recht alte, aber dennoch verbreitete „Text to Speech“-Software. Diese repräsentiert das Vorlesen lassen von Texten.

- Sphinx[5]: Sphinx ist eine Sprach Decodierungs-, also eine „Speech to Text“-Software. Diese repräsentiert z.B. das automatisierte Abtippen von Memoranden.

[5] Dieser Benchmark wurde zwar in die Konfigurationsdatei der Verwaltungssoftware aus Kapitel 4 eingebunden, ist jedoch nicht ohne weiteres ARM-kompatibel bzw. lässt sich auf dem gewählten Odroid-XU4 nicht kompilieren und wird deshalb auch im Weiteren nicht genutzt.

- StringSearch: Dieser Benchmark sucht ohne auf Groß- und Kleinschreibung zu achten nach Wörtern. Es ist also eine Repräsentation der normalen Suchfunktion eines jeden Textbearbeitungsprogramms.

3.3.1.5 Security

Unter der Kategorie „Security“ sind Benchmarks zusammengefasst, welche sicherheitsrelevante Algorithmen und Aufgaben wie Verschlüsselung, Signierung und Hashing repräsentieren.

- Blowfish: Berechnet das Verschlüsseln und Entschlüsseln mit dem symmetrischen Blockverschlüsselungsverfahren „Blowfish“.
- PGP[5]: Berechnet die Signierung und Verifikation mittels „Pretty Good Privacy“ (PGP), welche häufig beim Schlüsselaustausch genutzt wird.
- Rijndael: Berechnet das Verschlüsseln und Entschlüsseln mit dem symmetrischen Rijndael Blockverschlüsselungsverfahren, auch „Advanced Encryption Standard“ (AES) genannt.
- SHA: Berechnet das Hashing mittels des „Secure Hash Algorithm“ (SHA-1). Es repräsentiert also eine der einfacheren Methoden zur „eindeutigen“ Identifizierung bzw. Alternativrepräsentation von Daten.

3.3.1.6 Telecomm

Unter der Kategorie „Telecomm“ sind Benchmarks zusammengefasst, welche in der Telekommunikation häufig vertretene Anwendungen repräsentieren.

- FFT/IFFT: Die „Fast Fourier Transformation“ (FFT) und dessen inverse „Inverse Fast Fourier Transformation“ (IFFT) sind Algorithmen, die bei der Signal- und Bildanalyse verwendet werden.
- GSM: „Global System for Mobile Communications“ (GSM) ist ein Verfahrensstandard zur Sprachkodierung und -dekodierung, welche in Mobilfunknetzen angewandt wird.
- ADPCM: „Adaptive Differential Pulse Code Modulation“ (ADPCM) ist eine Sprach- bzw. Audio-Komprimierungsmethode, welche z.B. bei vielen schnurlosen Telefonen Anwendung findet.
- CRC32: „Cyclic Redundancy Check 32-bit“ (CRC32) ist ein Fehler erkennendes Codierungs- bzw. Signierungsverfahren. Dieses Verfahren erkennt, ob bei einer Übertragung Daten beschädigt wurden. Jedoch hat es keine Möglichkeit diese wiederherzustellen. Dies wird bei allen möglichen Datenübertragungen sowohl in der Telekommunikation als auch in der Netzwerktechnik benutzt.

3.3.2 PARSEC

Der PARSEC Benchmark Suite besteht aus unterschiedlichen Benchmarks [3]. Diese sind alle auf verschiedenen mitgelieferten Inputdatensätzen ausführbar.
Die Inputdatensätze umfassen [7, S. 13f.]:

- test: Möglichst kleine Eingabedaten, um die bloße Funktionalität des Benchmarks zu testen.
- simdev: Möglichst kleine Eingabedaten, welche jedoch sicherstellen, dass die Ausführung realen Eingabedaten recht nahe kommt, wobei möglichst viel vom Code abgedeckt wird.
- simsmall, simmedium and simlarge: Kleine, mittlere und große Eingabedaten, welche angepasst an die geplante Simulation ausgewählt werden können, um unterschiedlich große Nutzungsszenarien abzubilden.
- native: Sehr große Eingabedaten, welche native reale Eingabedaten repräsentieren und dessen Ausführungsdauer über eine einfache Simulation hinaus geht.

Die Benchmarks selbst sind:

- Black-Scholes: Dieser Benchmark bewertet anhand des Black-Scholes-Modells unterschiedliche Finanzoptionen. Dabei werden partielle Differentialgleichungen gelöst. Dieser Benchmark ist ein Teil des „Recognition, Mining and Synthesis“ (RMS)[6] Benchmark Suite von Intel [7, S. 22f.].
- Bodytrack: Dieser Benchmark versucht einen menschlichen Körper zu erkennen und dessen Bewegung durch mehrere getrennte Videosequenzen (Aufnahmen von unterschiedlichen Kameras) nachzuverfolgen. Dabei bestehen recht hohe Ansprüche sowohl an die CPU als auch an den Speicher des berechnenden Gerätes. Dieser Benchmark ist auch ein Teil des RMS Benchmark Suite von Intel [7, S. 23 ff.].
- CAnneal: Der „Cache-aware Annealing“ Benchmark minimiert die Routing-Kosten eines Chip Designs. Dies repräsentiert speicheroptimierte Ingenieursberechnungen [7, S. 28f.].
- DeDup[5]: Ist ein „Deduplications“ Benchmark, welcher mittels lokaler und globaler Komprimierung Datenströme komprimiert. Dies repräsentiert sowohl moderne Backup-Lösungen, als auch die Komprimierung in Kommunikationssystemen. [7, S. 29 ff.]

[6] RMS ist ein technologischer Ansatz von Intel, welcher dazu gedacht ist, große Datenmengen zu verarbeiten und zu analysieren [10]

- FaceSim: Die physikalische Simulation einen Gesichtes, wobei Muskeln und deren Restriktionen mit simuliert werden. Dabei werden viele parallele Berechnungen durchgeführt und ein Berechnungsmodel benutzt, welches viele zustandsbasierende Zwischenergebnisse erzeugt. Dieser Benchmark ist auch ein Teil des RMS Benchmark Suite von Intel [7, S. 32 ff.].

- Ferret [24]: Ein auf dem Ferret Toolkit basierender Benchmark, welcher eine inhaltsbasierende Gemeinsamkeitssuche bzw. -analyse durchführt. Das Ferret Toolkit selbst weist eine hohe Parallelität auf und ist nicht nur für das Arbeiten mit Text sondern für inhaltsbasierendes Arbeiten, mit z.B. Multimedia Daten, ausgelegt [7, S. 35 ff.].

- FluidAnimate [20]: Dieser Benchmark berechnet eine sehr speicherintensive dynamische Flüssigkeitsanimation anhand einer erweiterten Version der „Smoothed Particle Hydrodynamics“/„Geglättete Teilchen-Hydrodynamik“ (SPM) Methodik. Dieser Benchmark ist auch ein Teil des RMS Benchmark Suite von Intel [7, S. 38 ff.].

- FreqMine: Dieser Benchmark betreibt „Frequent Itemset Mining“ (FIMI) mit dem Array basierenden Ansatz des gut skalierbaren „Frequent Pattern-growth“ (FP-growth) [7, S. 42 ff.].

- RayTrace[5]: Dieser Benchmark berechnet im Raytrace Verfahren die Belichtung in einer 3D-Szene. Dies repräsentiert einen Standardfall von 3D-Berechnungen [7, S. 44 ff.].

- Streamcluster: Streamcluster ist auch ein Data-Mining Benchmark, welcher jedoch nach Clustern, also metrikbezogenen Ansammlungen von ähnlichen Daten, sucht. Dieser Benchmark ist auch ein Teil des RMS Benchmark Suite von Intel [7, S. 48 ff.].

- Swaptions[5]: Dieser Benchmark berechnet Preise für Swaption-Portfolios und repräsentiert damit Trader- bzw. Börsen-Berechnungen [7, S. 50].

- VIPS: Dieser Benchmark basiert auf dem „VESARI Image Processing System“ (VIPS) und berechnet diverse Bildtransformationen und -manipulationen. Somit repräsentiert dieser Benchmark Bildbearbeitung und allgemeine Computergrafik [7, S. 51 ff.].

- x264: Dieser Benchmark codiert Videos im H.264 Standard. Dieser, auch „Advanced Video Coding“ (AVC) genannte Standard, ist weit verbreitet und repräsentiert somit Multimediaanwendungen [7, S. 54 ff.].

3.3.3 Zusammenfassung

Im Allgemeinen repräsentiert MiBench eine große Anzahl an Alltagsanwendungen bzw. Workloads und PARSEC die besonders anspruchsvollen Algorithmen und Abläufe, wie z.B.

Datamining und Simulation. Somit sollte, da sowohl die Standard- als auch die Extremfälle abgedeckt sind, diese Software-Auswahl eine verhältnismäßig repräsentative Abdeckung eines realen Anwendungsszenarios darstellen. Neben der Vergleichbarkeit mit anderen Arbeiten war dies einer der Hauptfaktoren. Auch ist bei beiden Benchmarks gegeben, dass diese auf fest vorgegebenen Eingabedaten arbeiten, womit alle gewünschten Hauptfaktoren erfüllt sind.

Kapitel 4

Software

In diesem Kapitel geht es um die Entwicklung der Verwaltungssoftware zum Durchführen der Benchmarks. Dabei geht es nicht nur um die genauen Ansprüche an diese Software, sondern auch um die Problemstellungen, welche sich bei der Entwicklung dieser stellten. Außerdem werden die unterschiedlichen Entwicklungsstufen, deren Begründungen und das Endergebnis behandelt.

4.1 Zielsetzung

Eines der Ziele dieser Arbeit ist es eine Verwaltungssoftware zu schreiben, welche im Nachhinein ohne viel Aufwand mit weiteren Benchmarks als den zwei gewählten MiBench und PARSEC zu erweitern ist. Dabei muss es möglich sein, unterschiedliche Testszenarien wiederholbar durchzuführen. Die Installation dieser Benchmarks soll soweit realisierbar automatisierbar sein. Vorzugsweise sollte dies auch außerhalb der Verwaltungssoftware möglich sein. Zum Durchführen der Benchmarks müssen diese ansteuerbar sein. Um sinnvolle Rahmenbedingungen für die Benchmarks zu schaffen, müssen die Frequenzen der Kerne sowie deren Aktivierungszustand anpassbar sein. Als Ergebnis müssen sowohl die genauen Laufzeiten der Benchmarks als auch verwertbare Start- und Ende-Zeiten für die Vereinigung mit den Daten der Überwachungshardware ausgegeben werden.

4.2 CPUFreq

„CPUFreq“ ist ein „sysfs“-Interface zum Verwalten der CPU. Dabei ist es sowohl möglich viele Hardware-Daten auszulesen, wie z.B. die Maximal- und Minimalfrequenz, als auch auf viele Werte, welche die CPU betreffen, Einfluss zu nehmen. Dies geschieht über ein „sysfs“-Interface [23]. Das heißt es gibt im Dateisystem unter „/sys/devices/system/cpu“ mehrere Unterordner, welche jeweils einen Kern repräsentieren. Die dortigen Dateien sind

bereits Teil des Interfaces. In diesem Kapitel wird sich auf den für die Umsetzung der Software relevanten Teil des Interfaces begrenzt.

- „present“: Gibt an, welche CPUs theoretisch insgesamt ansteuerbar sind.
- „isolated“: Gibt an, welche CPUs vom Linux Scheduler isoliert wurden.
- „possible“: Gibt alle nicht vom Linux Scheduler isolierten CPUs an.
- „kernel_max“: Gibt an, bis zu welcher Zahl die CPUs durchnummeriert wurden.
- „online“: Gibt alle CPUs an, welche sich im Online-Status befinden.
- „offline“: Gibt alle CPUs an, welche sich im Offline-Status befinden.

Auch sind Ordner enthalten, welche dem Namensschema „cpuX“ folgen. Dabei sind an der Stelle X die CPUs durchnummeriert. Diese enthalten mit der „online“-Datei einen Zugriffspunkt, in welchem in Erfahrung gebracht werden kann, in welchem Aktivierungszustand sich der CPU befindet, sowie einen Unterordner namens „CPUFreq“. In diesem Unterordner befindet sich der Rest des für diese Arbeit relevanten Interfaces. Die Interface-Punkte, welche nicht nur Informationen liefern, sondern auch angepasst werden können, wurden mit einem Sternchen markiert [8].

- „affected_cpus“: Gibt eine Liste der CPUs an, welche online sind und durch eine Anpassung der Frequenz beeinflusst wären. Zu beachten ist, dass dies nur für Frequenzanpassungen gilt, nicht jedoch für das Ändern des Online/Offline-Status.
- „related_cpus“: Gibt eine Liste der CPUs an, welche theoretisch durch eine Anpassung der Frequenz beeinflusst wären. Also sowohl solche, die online als auch offline sind. Zu beachten ist, dass dies nur für Frequenzanpassungen gilt, nicht jedoch für das Ändern des Online/Offline-Status.
- „cpuinfo_max_freq“: Gibt die Maximalfrequenz der CPU an, welche von der Hardware vorgesehen ist.
- „cpuinfo_min_freq“: Gibt die Minimalfrequenz der CPU an, welche von der Hardware vorgesehen ist.
- „cpuinfo_cur_freq“: Gibt die aktuelle Frequenz an, auf welcher die CPU wirklich läuft.
- „scaling_governor“ *: Gibt an, welcher Skalierungs-Governor für die aktuelle CPU ausgewählt ist.
- „scaling_available_governors“: Gibt eine Liste aller Governors aus, welche für die aktuelle CPU möglich sind.

- „scaling_max_freq“ *: Gibt die Maximalfrequenz, bis zu welcher der aktuelle Governor skalieren darf, an.
- „scaling_min_freq“ *: Gibt die Minimalfrequenz, bis zu welcher der aktuelle Governor skalieren darf, an.
- „scaling_cur_freq“: Gibt die Frequenz an, welche theoretisch aktuell vom Governor gesetzt seien sollte.
- „scaling_setspeed“ *: Gibt die aktuell vom Governor gesetzte Frequenz an und kann bei Benutzung des „userspace“-Governors genutzt werden, um die Frequenz direkt zu setzen.
- „scaling_available_frequencies“: Gibt an, welche Frequenzen vom gegebenen Skalierungs-Treiber unterstützt werden.
- „cpuinfo_transition_latency“: Gibt in Nanosekunden an, wie lange ein Frequenzwechsel dauert. Dies entspricht beim Odroid-XU4 z.B. 154000 Nanosekunden bzw. 0,154 Millisekunden.

4.3 Generelle Problemstellungen

Bei der Entwicklung einer angemessenen Verwaltungssoftware für das Durchführen der Benchmarks boten sich einige Grundprobleme. Diese Probleme und ihre Lösungsansätze waren bspw.:

- Das bereits erwähnte „CPUFreq“-„Dateisystem“ bzw. Interface verhält sich nicht wie eine Konfigurationsdatei und lässt sich somit auch nicht auf jede beliebige Art beschreiben. Die dabei erhaltenen Fehlermeldungen, wie z.B. anscheinend fehlende Berechtigungen, sind im Allgemeinen sehr irreführend und wenig hilfreich. Die Tatsache, dass dieses Verhalten nicht deterministisch ist und ein und die selbe Lösung zum Schreiben der Datei im ersten Versuch funktioniert und im zweiten nicht mehr, war beim Debugging dieses Problems auch ein erschwerender Faktor. Der erfolgversprechendste Weg war am Ende das Benutzen der systemeigenen Befehle [23].
- Es musste eine angemessene Verwaltungsstruktur gefunden werden, in welche sich alle Benchmarks abbilden lassen. Eine einfache Konfig-Datei ermöglicht nicht das universale, voll automatische Installieren neuer Benchmarks. Auch wäre mit einer solchen Datei die Zustandskontrolle doch eher schwierig. Es stellte sich im Entwicklungsprozess heraus, dass ein Bash-Skript, welches sich an ein festes Template hält, all diese Voraussetzungen erfüllt. Dies ließ sich später auch sehr einfach in ein Python-Skript überführen.

- Die Implementierung kann nicht ohne entstehende Wartezeit während des Durchführens der Benchmarks Daten mit der Verwaltungshardware austauschen, da der Energieverbrauch des Netzwerkinterfaces die Energieverbrauchsmerksmale verfälschen würde. Somit muss vor einem Benchmarkdurchlauf soviel Hardware und natürlich auch Software wie möglich ausgeschaltet werden. Um die Ergebnisdaten übertragen zu können, wird das Netzwerkinterface nach dem Benchmarkdurchlauf wieder aktiviert.

4.4 Entwicklung

Die Entwicklung hatte mehrere Zwischenstufen, bei denen die Programmiersprache gewechselt wurde. Dies geschah aufgrund von Restriktionen der Programmiersprache oder unvorhergesehenen Fehlern bei Designentscheidungen. Dabei konnten mehrfach Elemente vorheriger Iterationen wiederverwendet werden. Jedoch ergab sich aus jeder Iteration auch ein größerer Funktionsumfang, was den jeweiligen Reimplementierungsaufwand erhöhte.

4.4.1 Bash

Zuerst wurde die Verwaltungssoftware als Bash-Skript entwickelt. Dabei wurde sich für eine einfache Struktur zur Verwaltung der Benchmarks entschieden. Jeder Benchmark Suite wird in ein separates Skript ausgelagert, wobei die Verwaltungsstruktur durch einen „include“ eingebunden wird. Ein Benchmark Suite ist unterteilt in „Moduls“„Submoduls“ und „Cases“. Dabei kann ein „Modul“ beliebig viele „Submoduls“ haben und ein „Submodul“ beliebig viele „Cases“. Ein „Case“ ist ein Befehl, welcher zum Durchführen des Benchmarks bzw. des spezifischen Benchmark-„Cases“ ausgeführt werden muss. Dabei bot sich das Problem, dass die organisatorischen und strukturellen Eigenschaften von Bash stark begrenzt sind. Dies zeigte sich darin, dass das Verwalten und Ansteuern von Arrays, welche dynamisch anhand ihres Namens verwaltet wurden, nur durch mehrere „evals“[7] möglich war. Dabei entstehen Konstrukte wie in Listing 4.1 zu sehen. Bei diesen finden mehrere verkomplizierende Zwischenevaluationen statt. Diese Struktur ist weder besonders lesbar, noch ist dieser Code besonders gut wartbar. Die Zwischenevaluationen sind nötig, da sämtliche Funktionen und die Arrays, in welchen die Informationen gespeichert sind, anhand ihrer Namen codiert sind. Auch stieg die Codekomplexität beim Hinzufügen weiterer Informationen stark an, da Bash bei Verschachtlungen von Strukturen in Arrays sehr umständlich und fehleranfällig ist. Durch diese Verschachtlung wurde das Einbinden zusätzlicher Benchmarks so komplex, dass diese Struktur im Nachhinein betrachtet nicht mit der Zielsetzung vereinbar war.

[7] Evaluiert den Eingabestring und handhabt ihn, als wäre er in die Shell direkt eingegeben worden.

Listing 4.1: Das Ausführen jedes Testcases eines Benchmark Suites in Bash

```
1  for i in ${TESTSUITES[@]} #Choose Testsuite
2  do
3    echo "Running Testsuite \"${i}\""
4    TMPI="${i}TESTS" #Choose Test
5    for j in $(eval echo \${${TMPI}[@]})
6    do
7      echo "Running Test \"${j}\""
8      TMPJ="${j}_CASES" #Choose Testdatacases
9      for k in $(eval echo \${${TMPJ}[@]})
10     do
11       echo "Running TestCase \"${k}\""
12       TMPK="${j}_OPT"
13       if [ -z $(eval echo \${${TMPK}}) ]
14       then
15         runTest ${j} ${k} ${TIMES}
16       else
17         for l in $(eval echo \${${TMPK}[@]}) #Choose Opt
18         do
19           runTest ${j} ${k} ${TIMES} ${l}
20         done
21       fi
22     done
23   done
24 done
```

4.4.2 C++

Nach der ersten Fehlentscheidung wurde die Programmiersprachentscheidung für C++ gefällt. Diese war jedoch vollkommen ungeeignet, um dynamisch mehr Benchmarks einzubinden, was dazu führte, dass für die „Konfigurationsdateien“, welche auch die Installation der Benchmarks durchführten, weiterhin Bash benutzt wurde. Die Struktur des Ausführens externer Skripte zum Durchführen von Benchmarks war durchaus von Vorteil. Das damit verbundene Problem war jedoch, dass zwei separate Programmiersprachen zu Redundanzen und somit zu mehr Aufwand führten. Auch wurde es immer klarer, dass Bash zum Erzielen des gewünschten Ergebnisses eine ungeeignete Struktur hat. Da auch nach einer gewissen Einarbeitungsphase keine sinnvolle, zufriedenstellende Lösung für dieses Problem gefunden wurde, wurde C++ verworfen. Einer der Hauptfaktoren war das eher schwierige Debugging. Dies war der Fall, da während dieses Entwicklungsschrittes noch nicht

klar war, dass die Dateiinteraktionsprobleme durch das „CPUFreq“ eigene „sysfs“-Interface verursacht wurden.

4.4.3 Python

Nach den ersten Fehlentscheidungen ist die Entscheidung der Programmiersprache auf Python gefallen. Der große Vorteil von Python ist die Art der Programmiersprache. Denn da Python eine Interpretorsprache ist, konnten die bereits in Bash umgesetzten Installationsskripte ohne viel Aufwand in die neue Programmiersprache überführt werden. Dabei mussten alle Teile, welche im Bash Skript in „pre_install“, „post_install“ und „install“ gekapselt waren, in „os.system()“ Befehle als Parameter übergeben werden. Die Organisationsstruktur kann in Python dank Objektorientierung wesentlich einfacher umgesetzt werden. Die einzelnen Installationsskripte bleiben jedoch weiterhin vom Verwaltungsprogramm abgekapselt. Diese dienen zur Installation, Deinstallation sowie beim Durchführen der Benchmarks als eine Art Datenbank, welche per „JavaScript Object Notation“ (JSON)[8] die zum Benchmark Suite gehörigen „Module“, „Submodule“ sowie deren Installationsstatus und die ausführbaren „Cases“ liefert. Die Verwaltungssoftware muss die meisten Funktionen dann an die Konfigurationsscripts weiterleiten. Für das Ausführen von bereits installierten Benchmarks muss bei dieser Struktur das Konfigurationsscript nur einmal angesprochen werden.

4.5 Ergebnis

Das Ergebnis ist eine Benchmark Verwaltungssoftware. Diese besteht primär aus zwei Teilen. Der erste Teil ist das Benchmark-Konfigurationsskript, welches einer bestimmten Struktur folgt und mit dem beliebig viele Benchmarks eingebunden werden können. Der zweite Teil ist die Verwaltungssoftware selbst, welche Benchmarks ausführt und verwaltet, die in den Benchmark-Konfigurationsskripten definiert wurden.

4.5.1 Benchmark-Konfigurationsskripte

Die Benchmark-Konfigurationsskripte liegen im Unterordner „Suites“. In diesem Ordner dürfen nur die Konfigurationsdateien, welche vom Namensschema her „<SuiteName>.py“ einhalten, sowie die „Template.py“-Datei, welche als Basis für die Konfigurationsdateien dient, liegen. Zusätzlich ist dort noch ein Ordner, welcher alle für das Konfigurationsskript benötigten Dateien enthalten kann. Jede dieser Konfigurationsdateien bietet die Möglichkeit den Benchmark zu installieren bzw. deinstallieren und sich den Zustand des Benchmark

[8] JSON ist eine einfache standardisierte, strukturierte Möglichkeit der Text basierenden Datenrepräsentation.

Suites ausgeben zu lassen. Diese Konfigurationsdateien werden von der Verwaltungssoftware angesprochen. Im Folgenden sind mit „<>“ optionale Parameter markiert. In allen Fällen gilt, dass es nicht möglich ist, einen Case anzugeben ohne Submodul und auch nicht möglich ist, ein Submodul ohne Modul anzugeben.

Interface/API der Konfigurationsskripte:

- ./<SUITE>.py status <MODUL> <SUBMODUL> <CASE> <-H 0/1>:
  Gibt den Status des Benchmarks bzw. je nach Parametern den Status eines Moduls, Submoduls oder Cases an. Den Status gibt es in zwei Ausprägungen:

  -H 0 : Das Ausführen ohne diesen Parameter gibt den Status im JSON Format aus.

  -H 1 : Gibt den Status in einem für Menschen leichter lesbaren Listenformat aus.

  Der Status beinhaltet den Namen des angefragten Elements sowie eine Liste aller Unterelemente. Dies gilt für Suite, Modul und Submodul. Bei einem Case gibt es keinen Installationszustand, jedoch beinhaltet dieser zusätzlich den für den Case auszuführenden Befehl.

- ./<SUITE>.py install <MODUL> <SUBMODUL>:
  Installiert je nach Parameter die Basis des Suites, ein Modul oder ein Submodul.

- ./<SUITE>.py installAll <MODUL>:
  Installiert je nach Parameter die Basis des Suites oder ein Modul. Dabei werden jeweils alle Unterelemente mit installiert.

- ./<SUITE>.py uninstall <MODUL> <SUBMODUL>:
  Deinstalliert je nach Parameter die Basis des Suites, ein Modul oder ein Submodul.

- ./<SUITE>.py isInstalled <MODUL> <SUBMODUL>:
  Gibt je nach Parametern aus, ob die Basis des Suites, ein Modul oder ein Submodul installiert ist.

Jedes Python-Skript, welches sich an diese API hält, kann als Konfigurationsdatei agieren, solange sich auch die Ausgabe des Status-Befehls an das JSON-Format hält. Mithilfe dieser Struktur kann jeder beliebige Benchmark nachträglich eingebunden werden.

JSON-Format:
Der vom Status-Befehl zurückgegebene Datensatz pro Benchmark Suite folgt im JSON-Format dem Schema aus Listing 4.2. Sollten mehrere Benchmarks Suites auf einmal ausgegeben werden, wird dies zusätzlich noch in einem Array gekapselt.

Listing 4.2: Status JSON-Format

```
{
  "name": <SuiteName>,
  "isInstalled": true/false,
  "moduls":[
    {
      "name": <ModulName>,
      "isInstalled": true/false,
      "submoduls":[
        {
          "name": <SubmodulName>,
          "isInstalled": true/false,
          "cases":[
            {
              "name": <CaseName>,
              "cmd": <CaseCMD>
            }, ...
          ]
        }, ...
      ]
    }, ...
  ]
}
```

4.5.2 Verwaltungssoftware

Die Verwaltungssoftware wird über die „main.py“ im Hauptordner angesteuert. Bei der Ansteuerung gilt, dass optionale Parameter mit „<>“ markiert sind und es nicht möglich ist, einen Case anzugeben ohne Submodul und auch nicht möglich ist ein Submodul ohne Modul anzugeben. Da diesmal jedoch auch zwischen Suites unterschieden werden muss, gilt auch noch, dass es nicht möglich ist, ein Modul anzugeben ohne eine Suite anzugeben. Für alle Befehle gilt auch, dass sollte keine Suite angegeben worden sein, der jeweilige Befehl für alle vorhandenen Benchmark Suites ausführt wird.

Interface/API der Verwaltungssoftware:

- ./main.py status <SUITE> <MODUL> <SUBMODUL> <CASE> <-H 0/1>:
  Leitet die „Status“-Anfrage an die angegebene Suite weiter.

- ./main.py isInstalled <SUITE> <MODUL> <SUBMODUL>:
  Leitet die „isInstalled“-Anfrage an die angegebene Suite weiter.

- ./main.py install <SUITE> <MODUL> <SUBMODUL>:
  Leitet die „install“-Anfrage an die angegebene Suite weiter.

- ./main.py uninstall <SUITE> <MODUL> <SUBMODUL>:
  Leitet die „uninstall“-Anfrage an die angegebene Suite weiter.

- ./main.py installAll <SUITE> <MODUL> <SUBMODUL>:
  Leitet die „installAll“-Anfrage an die angegebene Suite weiter.

- ./main.py list <SUITE> <MODUL> <SUBMODUL>:
  Listet je nach Parametern die für den nächsten freien Parameter möglichen Optionen auf.

- ./main.py run <SUITE> <MODUL> <SUBMODUL> <-a *amount* $\in \mathbb{N}$>:
  Der „run“-Befehl ist das Kernstück der Software. Es wird je nach Parameter jede Suite, eine Suite, ein Modul, ein Submodul oder ein Case ausgeführt. Dabei können über optionale Parameter interne Funktionalitäten gesteuert werden. Die nicht positionalen Parameter, also solche, die mit einem „-“ beginnen, funktionieren wie folgt:

  ▶ <-a amount>: „amount“ gibt an, wie oft ein Test ausgeführt bzw. mit Perf analysiert werden soll. Der Standardwert ist ein Mal.

  ▶ <-pM perfMode>: „perfMode“ gibt an, welche Testarten durchgeführt werden sollen:

  1 : Nur Perf-Tests werden durchgeführt.

  -1 : Nur normale Benchmark-Tests werden durchgeführt.

  0 : Beide Testarten werden durchgeführt.

  ▶ <-tM testMode>: „testMode“ gibt an, in welchem Modus das Ausführen der Testanzahl geschehen soll.

  0 : Batch-Mode, welcher alle Tests gleichzeitig startet und danach wartet, bis alle durchgelaufen sind.

  1 : Single-Run-Mode, in welchem jeder Test einzeln ausgeführt wird. Es wird dann so lange gewartet, bis dieser durchgelaufen ist, bevor der nächste gestartet wird.

  Dies gilt nur für die durch „amount“ gesteuerten Durchläufe. Cases werden in jedem Fall getrennt abgearbeitet.

- <-f freq>: „freq“ ist zu setzen, wenn der Benchmark nur auf dieser einen Frequenz ausgeführt werden soll. „freq“ kann nur auf die für die Hardware mögliche Frequenzen gesetzt werden.
- <-fm fMin>: „fMin“ gibt an, welche Frequenz als Minimalfrequenz bei der Frequenzvariation gewählt wird. Der Standard hierbei ist die Minimalfrequenz der Hardware. „fMin“ greift nur, wenn „freq“ nicht gesetzt ist. „fMin“ kann nur auf die für die Hardware mögliche Frequenzen gesetzt werden.
- <-fM fMax>: „fMax“ gibt an, welche Frequenz als Maximalfrequenz bei der Frequenzvariation gewählt wird. Der Standard hierbei ist die Minimalfrequenz der Hardware. „fMax“ greift nur, wenn „freq“ nicht gesetzt ist. „fMax“ kann nur auf die für die Hardware mögliche Frequenzen gesetzt werden.
- <-vM variationMode>: „variationMode“ gibt an, welche Art von Frequenzvariationen beim Durchlaufen durchgeführt werden.

  -2 : Software unterlässt sämtliche Frequenzanpassungen, sodass dies extern geregelt werden kann.

  -1 : Frequenzvariation wird deaktiviert. Jedoch wird die Frequenz vor der Durchführung auf die Minimalfrequenz gesetzt.

  0 : Frequenzen werden im einheitlichen Ablauf durchlaufen. Das bedeutet, dass alle Kerne jeweils einen Frequenzschritt machen.

  1 : Alle möglichen Frequenzvariationen werden durchlaufen.
- <-fOC filterOutCases>: „filterOutCases“ stellt hierbei eine mit Leerzeichen („ “) getrennte Liste an Cases dar, welche bei der Ausführung ausgelassen werden sollen.
- <-fIC filterInCases>: „filterInCases“ gibt eine mit Leerzeichen („ “) getrennte Liste von Cases an, welche ausgeführt werden sollen. Bei Ausführung dieses Parameters wird „filterOutCases“ ignoriert und insgesamt werden nur die in „filterInCases“ gelisteten Cases durchgeführt.
- <-fOSM filterOutSubmoduls>: „filterOutSubmoduls“ stellt hierbei eine mit Leerzeichen („ “) getrennte Liste an Submoduls dar, welche bei der Ausführung ausgelassen werden sollen.
- <-fISM filterInSubmoduls>: „filterInSubmoduls“ gibt eine mit Leerzeichen („ “) getrennte Liste von Submoduls an, welche ausgeführt werden sollen. Bei Ausführung dieses Parameters wird „filterOutSubmoduls“ ignoriert und insgesamt werden nur die in „filterInSubmoduls“ gelisteten Submoduls durchgeführt.
- <-fOM filterOutModuls>: „filterOutModuls“ stellt hierbei eine mit Leerzeichen („ “) getrennte Liste an Moduls dar, welche bei der Ausführung ausgelassen werden sollen.

- <-fIM filterInModuls>: „filterInModuls“ gibt eine mit Leerzeichen („ “) getrennte Liste von Moduls an, welche ausgeführt werden sollen. Bei Ausführung dieses Parameters wird „filterOutModuls“ ignoriert und insgesamt werden nur die in „filterInModuls“ gelisteten Moduls durchgeführt.
- <-fOS filterOutSuites>: „filterOutSuites“ stellt hierbei eine mit Leerzeichen („ “) getrennte Liste an Suites dar, welche bei der Ausführung ausgelassen werden sollen.
- <-fIS filterInSuites>: „filterInSuites“ gibt eine mit Leerzeichen („ “) getrennte Liste von Suites an, welche ausgeführt werden sollen. Bei Ausführung dieses Parameters wird „filterOutSuites“ ignoriert und insgesamt werden nur die in „filterInSuites“ gelisteten Suites durchgeführt.
- <-H Human>: Schaltet den Output des Status-Befehls im Standardwert 0 auf JSON- und beim Wert 1 auf ein menschlich lesbares Listen-Format um.
- <-R Result>: Schreibt die Ergebnisse des Durchlaufs mit in die Dateistruktur, identifiziert mit dem Ordnernamen „Result“ und ermöglicht somit einen abgebrochenen Benchmark wiederaufzunehmen.

4.5.3 Daten und Datenverarbeitung

In diesem Abschnitt wird auf die Struktur der Daten eingegangen, welche beim Benchmark-Prozess erzeugt werden. Es werden drei Arten von Daten erzeugt:

- Power-Daten: Werden von der Überwachungshardware an die Verwaltungshardware übertragen.
- Ergebnis-Daten: Werden nach Vollendung der Benchmarks von der Versuchshardware auf die Verwaltungshardware übertragen.
- Verarbeiteten-Daten: Werden auf der Versuchshardware aus den Power- und den Ergebnis-Daten erzeugt.

Power-Daten

Die Power-Daten bestehen aus einer CSV-Datei, welche pro Zeile folgende Informationen enthält:

- timestamp: Der Messzeitpunkt als Timestamp zum Datenabgleich mit den Ergebnis-Daten.
- volt: Gemessene Volt während <Timestamp>.
- ampere: Gemessene Ampere während <Timestamp>.

- watt: Gemessene Watt während <Timestamp>.
- watthour: Verbrauchte Wattstunden während des Testdurchlaufs bis zu <Timestamp>.

Der genaue Header lautet: „timestamp,volt,ampere,watt,watthour“. Die Power-Daten müssen vor dem Verarbeiten in den Unterordner „Power“ vom Ordner der Ergebnis-Daten übertragen werden.

Ergebnis-Daten

Die Ergebnis-Daten liegen auf der Versuchshardware im Ordner „Results“. Dort sind die unterschiedlichen Durchläufe in Unterordner sortiert, welche mit dem Start Timestamp des jeweiligen Skriptstarts benannten sind. Dort gibt es zwei Unterarten: die Normalen-Ergebnis-Daten und die Perf-Ergebnis-Daten, welche anhand der Prefixe „run_“ und „run_perf“ unterschieden werden können. Jeder Ergebnis-Datensatz ist mehrfach repräsentiert. Es wird pro Case, Submodul, Modul und Suite jeweils eine separate Datei angelegt, in welcher sich alle in die Rubrik passenden Datensätze befinden. Auch gibt es pro Art eine Datei, welche alle Datensätze enthält. Diese heißen „run.csv“ bzw. „run_perf.csv“. Die Normalen-Ergebnis-Daten und Perf-Ergebnis-Daten enthalten beide folgende Informationen:

- stateX: Eine durchnummerierte Aufzählung des Kern Online bzw. Offline-Zustandes.
- freqY: Eine durchnummerierte Aufzählung der Frequenz des Kerns.
- suite: Die Suite, zu welchem der Datensatz gehört.
- modul: Das Modul, zu welchem der Datensatz gehört.
- submodul: Das Submodul, zu welchem der Datensatz gehört.
- case: Der Case, zu welchem der Datensatz gehört.

Die Normalen-Ergebnis-Daten enthalten zusätzlich die Informationen start und end, welche den Start- und Endpunkt des jeweiligen Cases angeben. Die Perf-Ergebnis-Daten enthalten zusätzlich die Information perf, welche die mittels „perf“ ermittelte Ablaufzeit eines Durchlaufs des jeweiligen Cases angibt. Die Ergebnis-Datensätze sind vor der Verarbeitung headerfrei, um eine externe Weiterverarbeitung nicht zu behindern. Nach der Verarbeitung erhalten die Normalen-Ergebnis-Daten den Header „stateX, freqY, suite, modul, submodul, case, start, end“ und die Perf-Ergebnis-Daten den Header „stateX, freqY, suite, modul, submodul, case, perf“. Dabei stehen stateX und freqY für so viele Einträge wie das untersuchte System Kerne hat.

Verarbeiteten-Daten

Die Verarbeiteten-Daten, welche beim Durchführen des „processing.py“-Skriptes entstehen, werden im Ordner des jeweiligen Durchlaufs separat verwaltet. Hierfür wird im, mit einem Timestamp benannten, Ergebnis-Daten-Order ein Unterordner Namens „Joined“ angelegt. Diese fallen je nach Konfiguration des Skripts anders aus, können jedoch folgende Informationen enthalten:

- stateX: Eine durchnummerierte Aufzählung des Kern Online bzw. Offline-Zustandes.
- freqY: Eine durchnummerierte Aufzählung der Frequenz des Kerns.
- suite: Die Suite, zu welchem der Datensatz gehört.
- modul: Das Modul, zu welchem der Datensatz gehört.
- submodul: Das Submodul, zu welchem der Datensatz gehört.
- case: Der Case, zu welchem der Datensatz gehört.
- minWatt/avgWatt/maxWatt: Während dieses Cases gemessene minimale, maximale sowie durchschnittliche Watt.
- minAmpere/avgAmpere/maxAmpere: Während dieses Cases gemessenes minimales, maximales sowie durchschnittliches Ampere.
- minVolt/avgVolt/maxVolt: Während dieses Cases gemessene minimale, maximale sowie durchschnittliche Volt.
- perf: Die mittels „perf“ ermittelte Ablaufzeit eines Durchlaufs des jeweiligen Cases.
- fX: Eine Auflistung der Frequenzgruppen, also der verwandten Kerne.
- iteration: Ein Index, welcher jede Frequenziteration einzigartig identifiziert.

Diese umfassende Informationsstruktur bietet die Möglichkeit, die Daten in unterschiedlichen Arten weiter zu verwenden bzw. zu verarbeiten.

4.5.4 Zusammenfassung

Durch die entkoppelte Struktur wird die Flexibilität gewährleistet. Die Software bietet dabei alles, was nötig ist, um auf unterschiedlicher Hardware mit minimalem Aufwand wiederholbare Ergebnisse zu erzielen. Dabei können beliebig viele Benchmarks eingebunden werden und auch für spezifische Tests nur Teile dieser angesprochen werden. Außerdem ist es möglich, Benchmark übergreifend zu filtern. Somit ist die Zielsetzung der Verwaltungssoftware erfüllt.

Kapitel 5

Power Management

In diesem Kapitel wird auf den Kern der Arbeit, das Power Management, eingegangen. Dabei werden zuerst einige Grundtechniken eingeführt, welche dazu dienen Basisstrukturen und Untertechniken darzustellen. Außerdem sollen die Grundtechniken verdeutlichen, welche Problemstellungen in einer Power Management Lösung entstehen können. Des Weiteren bilden sie jeweils auch eine repräsentative Basis für die in Abschnitt 5.4 vorgestellten Umsetzungen.

5.1 Techniken

Da in dieser Arbeit nur zentralisierte heterogene Multiprocessing-Systeme behandelt werden, werden auch nur in solchen Systemen anwendbare Power Management-Techniken betrachtet. Bevor die Haupttechniken behandelt werden, mit welchen sich diese Arbeit auseinandersetzt, werden an dieser Stelle zuerst ein paar grundlegende Methodiken erklärt:

5.1.1 Peak Power Management

Peak Power Management steht für zwei unterschiedliche Arten des Power Managements.

- Verlaufsanalyse: Dies ist eine Technik, bei der anhand der Nutzungsdaten vorhergesehen wird, ob ein Anstieg im Energieverbrauch bzw. in der Prozessorauslastung lediglich ein Peak[9] ist oder ob die Last wirklich „dauerhaft“ gestiegen bzw. gesunken ist. Für einen Peak würde es sich meistens nicht lohnen, die Intensität anzupassen, da eine solche Anpassung immer einen Overhead mit sich bringt, welcher den Anpassungsvorteil möglicherweise überwiegt. Intensitätsanpassung steht hierbei für eine beliebige Anpassung, welche den Energieverbrauch erhöht bzw. senkt, indem z.B. Kerne ein- bzw. ausgeschaltet werden oder die Frequenz angepasst wird.

[9] Ein Peak ist ein kurzer Anstieg, welcher auch direkt nach dem Anstieg wieder auf das ursprüngliche oder ein nur leicht erhöhtes Niveau absinkt.

- Verbrauchsanalyse: Bei dieser Art des Peak Power Managements handelt es sich um das Handhaben der maximalen Auslastung einer Berechnungslösung. Dies ist nötig, da viele Systeme, sollten sie zu lange auf der maximalen Auslastung laufen, mit „Thermal Throttling“[10] beginnen. Dabei können die Peaks auf unterschiedliche Kerne verteilt oder sogar verhindert werden. Somit kann das „Thermal Throttling“ präemptiv abgewendet bzw. verzögert werden [19].

5.1.2 User Profiling

Beim User Profiling werden Ablaufs- bzw. Handlungsmuster eines spezifischen Nutzers analysiert. Dabei steht der Nutzer nicht zwanghaft für einen Menschen, sondern kann auch als Hardware Nutzer, also Programm bzw. Applikation, interpretiert werden. Dies geschieht sowohl in einer zeitlichen Sichtweise, also von wann bis wann liegt dem Nutzer welches Nutzungsprofil zugrunde, als auch im Untersuchen der Abfolge von Aktivitäten. Eine solche relevante Abfolge wäre z.B. wenn der Nutzer eine Aktion durchführt oder ein bestimmtes Handlungsmuster X zeigt, dass danach davon ausgegangen werden kann, dass mit einer hohen Wahrscheinlichkeit darauf Muster Y folgen wird. Anhand einer solchen Analyse kann die Systemauslastung bis zu einem gewissen Grad vorhergesagt werden und somit, bevor das Bedürfnis nach mehr Leistung auftritt, die Intensität präemptiv angepasst werden [9]. Intensitätsanpassung ist hierbei äquivalent zur Intensitätsanpassung im Peak Power Management.

5.1.3 Sensor-Optimierung

Viele Power Management Methoden basieren auf Sensordaten. Jedoch beherbergt dies das Problem, dass beim Auslesen von Sensoren auch Energie benötigt wird. Um diesen Energiebedarf zu senken, kann die Nutzung der Sensoren optimiert werden. Dies kann auf unterschiedliche Arten geschehen. Es können alternative Daten, wie z.B. Performance Counter, welche das System unabhängig verwalten, benutzt werden. Auch kann das Muster, in welchem die Sensoren gelesen werden, an die anfragende Power Management Technik angepasst werden, was je nach Anpassung entweder zu weniger oder zu spezifischeren Sensoranfragen führt. Alternativ kann auch eine Trendanalyse durchgeführt werden, wobei seltener reale Sensordaten angefragt werden und in der Zwischenzeit auf geschätzte Werte zurückgegriffen werden kann. Die geschätzten Werte basieren dabei auf dem Verlauf der Daten der letzten realen Sensoranfragen. Somit kann der Energiebedarf der Sensor basierenden Informationsbeschaffung verringert werden. Unter diese Rubrik fällt auch „Adaptiv Sensing“, also das Anpassen der Ausleseart bzw. -struktur und „Adaptiv Sampling“, also das Anpassen der Ausleserate an die aktuellen Energiereserven [4, S. 165-172].

[10] Beim „Thermal Throttling“ wird die Hardware aufgrund der Überschreitung eines Temperaturlimits abgebremst, um Überhitzung zu vermeiden.

5.1.4 Zusammenfassung

All diese Techniken sind Verfahren, welche in und auf Anpassungs- bzw. Analyseverfahren für Power Management benutzt werden können. Es wird dabei klar, dass beim Power Management viele Faktoren zusammenlaufen und auch wenn die meisten Techniken sich nur auf einen bzw. wenige dieser Faktoren beziehen, sie alle in Betracht gezogen werden müssen. Auch kristallisiert sich heraus, dass unterschiedliche Techniken unter bestimmten Bedingungen ineinandergreifen bzw. eine Technik intern eine andere verwendet.

5.2 Dynamic Voltage and Frequency Scaling

Das dynamische Anpassen der Prozessor- bzw. Kernfrequenz nennt sich „Dynamic Voltage and Frequency Scaling“ (DVFS). Dabei wird die Frequenz und somit der Energieverbrauch an die aktuelle Systemauslastung bzw. kommende Systembedürfnisse angepasst. Dies kann entweder auf Basis vordefinierter Ereignisse bzw. Codestellen passieren oder anhand aktueller Systemdaten dynamisch ausgelöst werden. Natürlich kann auch hier die Methodik der Sensor-Optimierung angewandt werden und anhand von Leistungsmerkmalen entschieden werden. Die Frequenzanpassungen geschehen jedoch meist nicht nach einem simplen Min bzw. Max-Prinzip, d.h. sie werden nicht einfach nur zwischen maximaler und minimaler Leistung hin und her geschaltet. Bei der Adaption wird sich häufig eher an einem Energy- bzw. Power-Model orientiert. Anhand eines solchen Modells kann die bestmögliche Frequenz in Bezug auf den Energieverbrauch zu der aktuellen Gegebenheit ausgewählt werden. Es gibt jedoch unterschiedliche DVFS-Varianten:

5.2.1 Offline DVFS

Bei Offline-DVFS werden während des Kompilierens eines Programms Adaptionspunkte vordefiniert. Diese können entweder fest einprogrammiert werden oder es wird während des Kompilierens im Voraus ein Ausführungsplan anhand einer Ablaufanalyse erzeugt. Demnach wird an bestimmten Punkten des Programms die Frequenz des Prozessors angepasst. Dabei wird entweder statisch oder relativ gearbeitet. Statisch heißt in diesem Fall, dass bei den Adaptionspunkten vordefinierte Frequenzwerte gesetzt werden. Relativ bedeutet, dass die Frequenzwerte basierend auf Input-Art bzw. -Größe angepasst werden. Das Problem an Offline-DVFS ist, dass aufgrund der vorab definierten Anpassungsvarianten diese nicht auf aktuellen Daten basieren und somit externe Faktoren, wie z.B. Wärmeentwicklung oder andere unabhängige Workloads, nicht in Betracht gezogen werden können. Somit ist dies nur wirklich in Systemen anwendbar, bei welchen sämtliche Workloads selbst kontrolliert werden. Beachtenswert ist hierbei, dass der primäre Overhead, der Overhead während des Kompilierens ist, da er bei der Ausführung selbst nicht ins Gewicht fällt [4, S. 164 f.]. Es stellt sich heraus, dass sich dieses Verfahren besonders für Systeme eignet, welche einmal

konfiguriert werden und dann ohne direkter Nutzerbeeinflussung der Software ihre Aufgabe erfüllen.

5.2.2 Online DVFS

Bei Online-DVFS werden zur Laufzeit in Intervall oder Checkpoint basierender Form die Frequenzen dynamisch angepasst.

- Intervall basierend :
  Bei dieser Form wird in regelmäßigen Zeitabständen der aktuelle Systemzustand analysiert. Dabei werden sowohl reale Sensordaten in Betracht gezogen, als auch, wie in der „Sensor-Optimierung“ üblich, „Leistungsmerkmale“ und andere Hilfszähler bzw. Hilfsdaten benutzt. Auch werden über diese Intervalle hinweg die Verläufe der Anforderungen und die daraus resultierenden Frequenzen analysiert. Das Ziel dieser Intervall übergreifenden Analyse ist es, einen Trend zu erkennen und somit den zukünftigen Verlauf der Daten vorherzusagen. Der Vorteil des Intervall basierenden DVFS ist, dass dies vollständig unabhängig von der laufenden Software geschehen kann. Es müssen in die Software weder vordefinierte Adaptionspunkte einprogrammiert werden, noch muss die Software vorher analysiert werden, um solche Punkte zu setzen. Zusätzlich dazu, dass die Software vorher nicht bekannt sein muss, gilt bei diesem Verfahren auch, dass zur Laufzeit nicht nach bestimmten Aktivitäten, wie „waits“ oder anderen relevanten Funktionen, zusätzlicher Code ausgeführt werden muss und somit diese Form des Laufzeitoverheads ebenfalls umgangen werden kann [4, S. 162 f.].

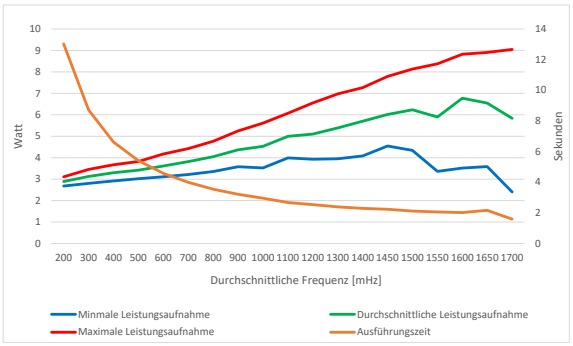
- Checkpoint basierend:
  Bei dieser Form wird an im Programmcode vordefinierten oder zur Laufzeit gehookten speziellen Punkten im Programmcode der Systemzustand analysiert oder entsprechend der Art des erreichten Punktes dieser angepasst. Dabei gibt es wieder unterschiedliche Herangehensweisen. Zum einen wird wie beim Intervall basierenden DVFS eine Analyse durchgeführt und anhand dieser die Frequenz angepasst. Jedoch gibt es auch Methoden, bei welchen diese vordefinierten Punkte die Art der Anpassung selbst definieren, dies geschieht z.B. anhand von Punktsystemen, welche dann wiederum als Grundlage zur Frequenzanpassung herangezogen werden. Eine zu betrachtende Sonderform wäre noch die Nutzung von Checkpoint basierenden DVFS in Echtzeitsystemen. Bei diesem Sonderfall wird der Zustandswechsel des Scheduling bzw. Taskmanagementsystems auch häufig als ein solcher Checkpoint angesehen. Da die Systemrestriktionen bzw. die Grundbedingungen eben dieses Scheduling bzw. Taskmanagementsystems bekannt sind, kann dies auch in die Anpassungsmethodik einfließen [4, S. 163 f.].

5.3 Datenanalyse

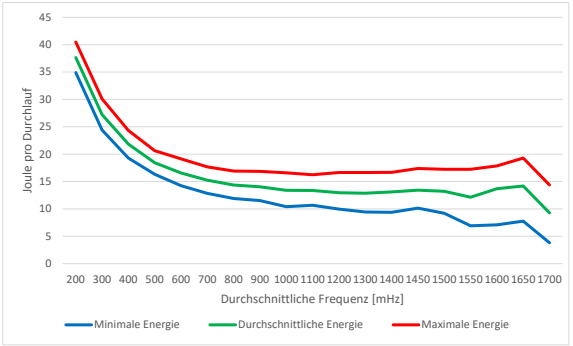
In diesem Abschnitt wird nur auf die Ergebnisse der Benchmarkdaten sowie deren Bedeutung eingegangen. Zuerst wird dabei der Verlauf betrachtet, wie er entsteht, wenn man einen, wie in Abschnitt 3.1 beschrieben, Durchlauf mit einem einheitlichen Anstieg durchführt. Danach wird ein Durchlauf mit allen Variationen betrachtet. Dabei wird auf reale Daten eingegangen und diese werden interpretiert. Die Ergebnisse unterscheiden sich von Verläufen anderer Arbeiten[11] primär aufgrund der Betrachtung der Nutzungsfälle und der unterschiedlichen Frequenzstrukturen der Prozessoren.
Alle in diesem Abschnitt betrachteten Frequenzen sind in mHz angegeben.

5.3.1 Einheitlicher Anstieg

Bei einem einheitlichen Anstieg werden die Frequenzen aller Kerne bei jedem Frequenzschritt gleich angehoben. Da die „big.LITTLE“-Architektur des „Odroid-XU4“ dazu führt, dass der „big“-Teil höher taktbar ist als der „LITTLE“-Teil, muss ab dieser Schwelle, welche im Fall des „Odroid-XU4“ bei 1400mHz liegt, mit der durchschnittlichen Frequenz gerechnet werden. Dementsprechend steht z.B. 1450mHz für 1400mHz im „LITTLE“-Teil und 1500mHz im „big“-Teil der Berechnungslösung. Eine getrennte Ansicht der Graphen befindet sich im Anhang A.



(a) Verlauf der Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)



(b) Verlauf des Energieverbrauchs pro Durchlauf (e.D.)

Abbildung 5.1: Betrachtung eines einheitlichen Durchlaufs

Abbildung 5.2a stellt den Verlauf der maximalen, durchschnittlichen und minimalen Leistungsaufnahme in Relation zur Ablaufzeit eines Benchmarks dar. Dabei stellt dies einen vollständigen Durchlauf mit PARSEC und MiBench dar. Daneben ist in Abbildung 5.2b der daraus resultierende Energieverbrauch dargestellt. Auch dieser ist für die drei Stufen maximal, durchschnittlich und minimal abgebildet.

[11] Wie in [21] u. [22]

Bei der Betrachtung der minimalen Leistungsaufnahme bzw. der minimalen Energieverbrauchswerte ist der relative glatte Verlauf der Leistungsaufnahme am bemerkenswertesten. Der Anstieg ist nur sehr gering für die Größe der entsprechenden Frequenzanpassung. Dies hängt vermutlich mit den I/O bzw. Speicherzugriffen zusammen, welche nicht von den Frequenzvariationen beeinflusst werden, sowie sämtlichen Task unabhängigen Energieverbrauchsfaktoren.

Bei einer Betrachtung des durchschnittlichen Verbrauchs entspricht dieser schon eher dem erwarteten Ergebnis. Auffallend ist jedoch erneut der Abfall zwischen einer durchschnittlichen Frequenz von 1600 und 1700 mHz, welcher vermutlich aufgrund der Task Migrierung beim globalen Scheduling auftritt. Die den „big“-Kernen zugewiesenen Aufgaben wurden von diesen bereits erledigt, da diese höher getaktet sind als die „LITTLE“-Kerne. Infolgedessen kann der Scheduler die Tasks auch auf die nun frei gewordenen „big“-Kerne verteilen und diese sogar von den „LITTLE“-Kernen zu diesen migrieren.

Die Betrachtung der Maximalwerte zeigt, dass obwohl die Leistungsaufnahme durchgehend steigt, der Abfall der Ausführungszeit ausreicht, dass selbst in dieser Betrachtungsweise der Energieverbrauch im Gesamten sinkt.

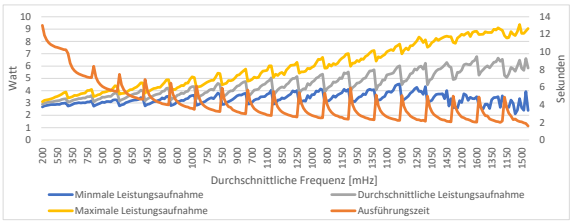
Dass das Konzept einer heterogenen Berechnungslösung funktioniert, ist am Knick eines jeden Energieverlaufsgraphen erkennbar, da dieser auftritt ohne dass die Variationsart geändert wurde.

5.3.2 Alle Variationen

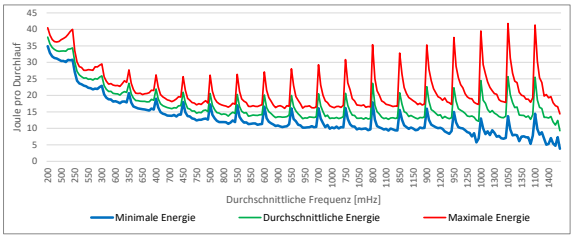
Bei der Ablaufart mit allen Variationen wird systematisch durch die Frequenzvariationen iteriert. Die Tabelle 5.1 dient als Beispiel für ein besseres Verständnis der Systematik hinter den Frequenzvariationen aus Abb. 5.2, welche den Frequenzverlauf in mHz darstellt.

„LITTLE“-Frequenz	„big“-Frequenz
200	200
300	200
...	...
1400	200
200	300
300	300
...	...

Tabelle 5.1: Alle Variationen - Frequenzverlauf



(a) Verlauf der Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)

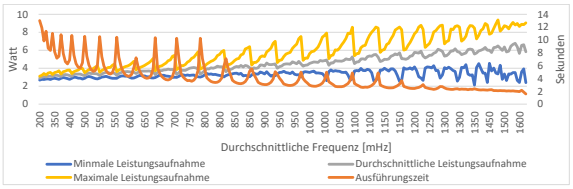


(b) Verlauf des Energieverbrauchs pro Durchlauf (e.D.)

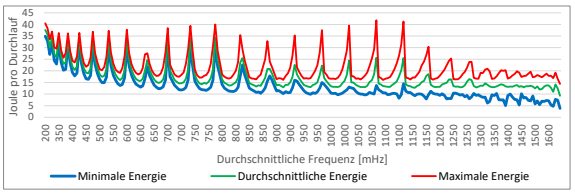
Abbildung 5.2: Betrachtung eines Durchlaufs mit allen Variationen in Iterationsreihenfolge

Da in diese Betrachtung mehr Faktoren einfließen als in die eines einheitlichen Anstiegs, ist hierbei mehr Interpretation nötig. Bei der Betrachtung der Ausführungszeit fällt auf, dass die einzelnen Spitzen durch den Überlauf von „LITTLE“ zu „big“ entstehen, da bei jedem dieser Überläufe nur um 100mHz im „big“-Teil erhöht, jedoch um 1200mHz im „LITTLE“-Teil verringert wird. Ein ähnliches Verhalten ist bei der Leistungsaufnahme zu beobachten, wobei jedoch das Minimum und Maximum immer weiter ab ebben.

Dabei wird klar, dass diese Art der Frequenzerhöhung zur glatten Abbildung eines energiesparenden Frequenzverlaufs ungeeignet ist. Somit folgt die Betrachtung derselben Daten in nach durchschnittlicher Frequenz sortierter Form in Abb. 5.3. Dabei beginnen gleiche Frequenzbereiche immer mit der Betrachtung der kleinsten Frequenz des „big“-Teils.



(a) Verlauf der Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)



(b) Verlauf des Energieverbrauchs pro Durchlauf (e.D.)

Abbildung 5.3: Betrachtung eines Durchlaufs mit allen Variationen in sortierter Form

Bei dieser Betrachtungsart wird die Glättung bei höheren Frequenzen deutlich. Jedoch zeigt sich auch der wohl unintuitivste Faktor eines heterogenen Multiprocessing-Systems. Die Tatsache, dass es nicht ausreicht, sich nur an der Frequenz zu orientieren, da diese von Berechnungseinheiten unterschiedlicher Ausprägung stammen. Die Tiefpunkte in der Ausführungszeit sind stets die Repräsentation der Hochpunkte der Frequenz des „big“-Teils, welche interessanter Weise nicht mit den Hochpunkten der Leistungsaufnahme übereinstimmen. Infolgedessen stellt sich die Frage nach einer Betrachtungsweise, aus welcher das Verhältnis ersichtlich wird.

Als eine solche Betrachtungsweise stellte sich die Repräsentation sortiert anhand des durchschnittlichen Energieverbrauchs, welche in Abb. 5.4 dargestellt wurde, als am geeignetsten heraus.

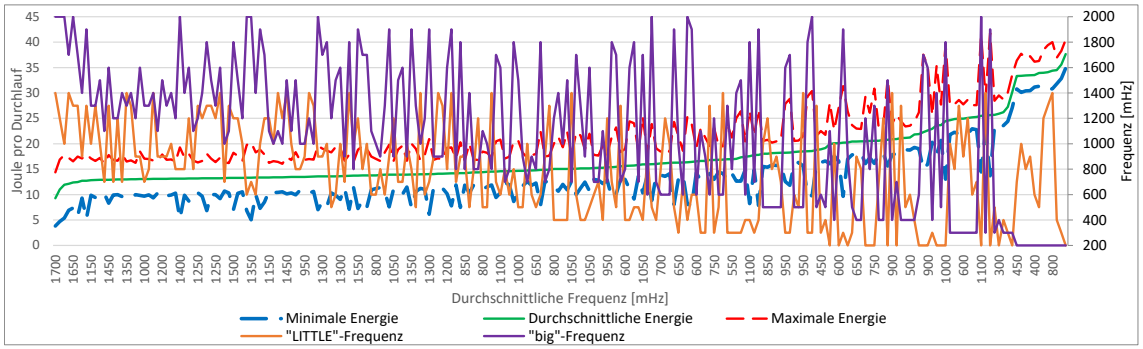


Abbildung 5.4: Verlauf des Energieverbrauchs pro Durchlauf sortiert anhand des durchschnittlichen Energieverbrauchs (e.D.)

Die Abb. 5.4 zeigt primär das Verhältnis der Frequenz der beiden Teile der heterogenen Berechnungslösung auf. Dabei ist auffallend, dass der Verlauf in keinster Weise einer glatten Funktion oder Struktur folgt und sich somit als Power-Model eine simple Funktion nicht wirklich eignet. Als Struktur sollte somit entweder ein Mapping oder eine verhältnismäßig grobe Approximation gewählt werden.

5.4 Anwendung

In diesem Kapitel werden DVFS basierende Techniken angewandt und miteinander verglichen.

5.4.1 OnDemand - Linux Nativ

Der „ondemand“-Governor passt die Frequenz abhängig von der CPU-Auslastung an. Dieser ist seit der Kernel-Version 2.6.9 Bestandteil des Linux Kernel. Somit ist dies ein „inKernel“-Governer, welcher nicht im Userspace agiert. Das Ziel des „ondemand“-Governor ist es, die Frequenz nur bei Bedarf anzuheben und somit sowohl Energie zu sparen, als auch mit minimalem Overhead eine konfigurierbare DVFS-Lösung zu bieten. Trotzdem sollte bei dieser Energiesparmethode die Performance soweit möglich nicht beeinflusst werden. Dabei wird der Algorithmus 5.1 zugrunde gelegt. Hierbei ist zu beachten, dass die Grundeinstellung für die Abtastraten X und Y auf der CPU-Transition-Latency (siehe Abschnitt 4.2) und der plattformabhängigen Idle-Konstante „HZ“ basiert.

```
1: every X milliseconds do
2:     for all CPU in the system do
3:         get utilization since last check
4:         if utilization > UP_THRESHOLD then
5:             increase frequency to MAX
6: every Y milliseconds do
7:     for all CPU in the system do
8:         get utilization since last check
9:         if utilization < DOWN_THRESHOLD then
10:            decrease frequncy by 20%
```

Algorithmus 5.1: Der ursprüngliche „ondemand"-Algorithmus [23, S. 220 bzw. S. 6]

Diese Grundstruktur bietet einige Konfigurationsmöglichkeiten:

- sampling_rate_min: Vordefinierter, minimaler, akzeptabler Wert für die sampling_rate, welcher nicht verändert werden kann.

- sampling_rate_max: Vordefinierter, maximaler, akzeptabler Wert für die sampling_rate, welcher nicht verändert werden kann.

- sampling_rate: Die aktuelle Abtastrate, welche auch gesetzt werden kann, solange dabei die Begrenzungen gegeben durch sampling_rate_min und sampling_rate_max eingehalten werden.

- up_threshold: Bietet die Möglichkeit die Variable UP_THRESHOLD aus dem Algorithmus 5.1 zu setzen und somit die Möglichkeit den Threshold, ab welchem die Frequenz direkt auf die Maximalfrequenz gesetzt wird, zu beeinflussen.

- ignore_nice_load: Bietet die Einstellungsmöglichkeit, dass abgestorbene Tasks als Idle-Zeit behandelt werden.

Die Grundfunktionalität ist die regelmäßige Überprüfung, ob die Auslastungsobergrenze UP_THRESHOLD überschritten oder unterschritten wurde. Sollte sie überschritten worden sein, wird die Frequenz direkt auf das Frequenzmaximum gesetzt. Sollte sie unterschritten worden sein, wird die Frequenz um 20% verringert.

Diese Grundstruktur hat sich bis heute nicht viel verändert, wurde jedoch optimiert (siehe Algorithmus 5.2).

```
1: for all CPU in the system parallel do
2:     every X milliseconds do
3:         get utilization since last check
4:         if utilization > UP_THRESHOLD then
5:             increase frequency to MAX
6:         else
7:             decrease frequncy by 20%
```

Algorithmus 5.2: Der aktuelle „ondemand“-Algorithmus [23, S. 220 bzw. S. 6]

Dabei wurden die vorher entkoppelten Abtastraten für das Hoch- und Runtersetzen der Frequenz zusammengefasst, sowie das Überprüfen auf die einzelnen Kerne parallelisiert [23, S. 221 ff. bzw. S. 7 ff.].

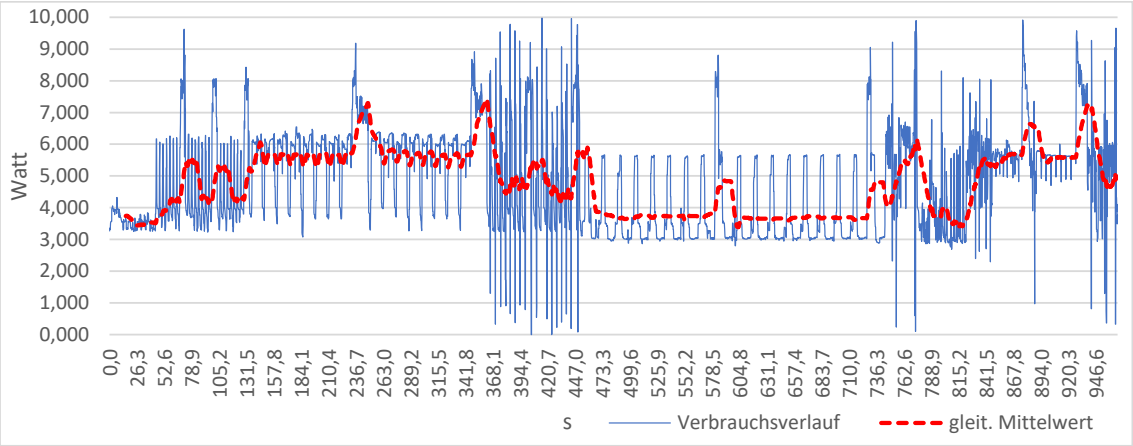


Abbildung 5.5: Verbrauchsverlauf eines Benchmark Durchlaufs mit „ondemand“ (e.D.)

In Abbildung 5.5 ist der schnelle Frequenzwechsel des „ondemand“-Governors und dessen Auswirkung auf den Verbrauch sehr gut zu erkennen. Die unterdurchschnittlichen Werte sind durch Restbestände begründet. Dabei wird ein vollständiger Durchlauf abgebildet. Die horizontale Achse stellt die vergangenen Sekunden seit Start des Durchlaufs und die vertikale den zum jeweiligen Zeitpunkt gemessenen Verbrauch dar. Die rot gestrichelte Linie stellt den gleitenden Mittelwert zur 75 Periode dar, um eine Approximation der realen Leistung zu bieten. Die einzelnen Elemente eines Durchlaufes sind anhand des jeweiligen Strukturwechsels zu erkennen. Dies wird deutlich, wenn man die Abbildung 5.5 in Relation zu den Benchmark-Ablaufdaten aus der Tabelle 5.2 betrachtet.

Suite	Modul	Energie	Laufzeit
PARSEC	Black-Scholes	67, 0	135.0
	CAnneal	231, 9	361, 2
	Ferret	399, 6	452, 7
	FreqMine	580, 0	743, 0
MiBench	Automotive	749, 0	756, 0
	Consumer Devices	758, 0	835, 0
	Network	835, 8	840, 8
	Office	841, 0	845, 0
	Security	874, 8	888, 2
	Telecomm	927, 0	965, 8

Tabelle 5.2: Benchmark-Ablaufdaten - „ondemand“

Der reale Energieverbrauch des Durchlaufs, welcher sich im Ganzen auf ca. 237 Joule beläuft, sowie dessen Aufteilung ist in Abbildung 5.6 dargestellt. Die dazugehörigen Werte können der Tabelle 5.3 entnommen werden.

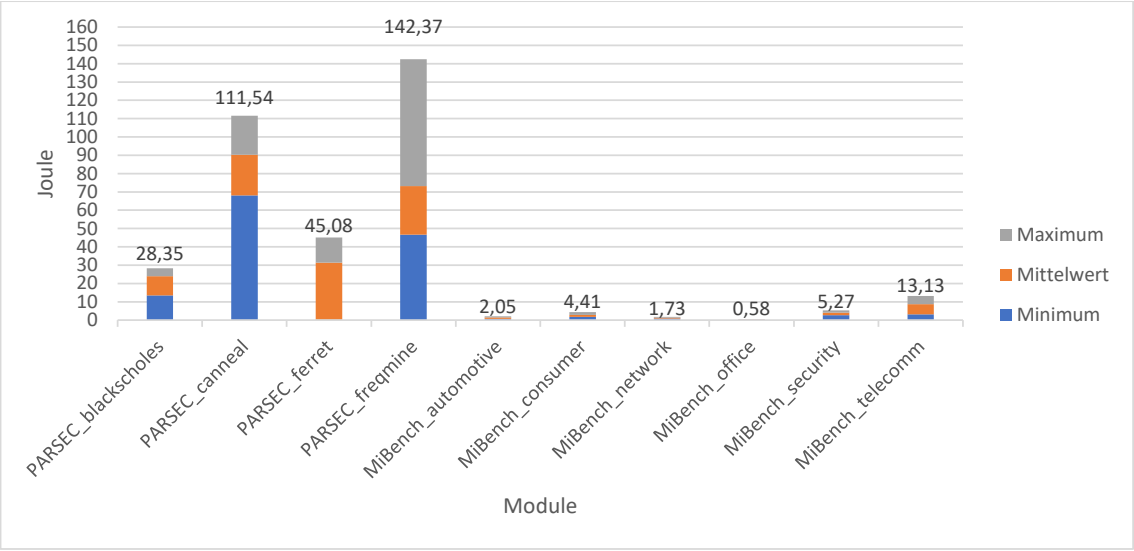


Abbildung 5.6: Verbrauchswerte eines Benchmark Durchlaufs mit „ondemand“ (e.D.)

Suite	Modul	Minimum	Mittelwert	Maximum
PARSEC	Black-Scholes	13, 49	23, 95	28, 35
	CAnneal	68, 03	90, 31	111, 54
	Ferret	0, 20	31, 41	45, 08
	FreqMine	46, 58	73, 13	142, 37
MiBench	Automotive	0, 68	1, 44	2, 05
	Consumer Devices	1, 81	2, 85	4, 41
	Network	0, 89	1, 28	1, 73
	Office	0, 29	0, 41	0, 58
	Security	2, 64	4, 05	5, 27
	Telecomm	3, 23	8, 63	13, 13

Tabelle 5.3: Benchmark-Verbrauchswerte - „ondemand“

Dabei ist deutlich die Spektrumsabdeckung zu erkennen, welche durch die Wahl unterschiedlicher Workloads erreicht wurde. Ein weiterer bemerkenswerter Punkt ist die Fluktuation im Energieverbrauch einzelner Workloads, welche durch das schnelle Wechseln und Maximieren der Frequenz im „ondemand“-Governor entsteht.

5.4.2 Fuzzy

Ein alternativer Ansatz wäre es nicht nach einer Lösung zu suchen, welche universal zum aktuellen Task bzw. dem Belastungsszenario passt, sondern sich am Nutzer und dessen gewünschten Verhalten zu orientieren. Um ein solches Verhalten zu ermöglichen, bedarf es einer Priorisierung durch den Nutzer. Die Entscheidung ist: Steht aktuell Energie oder Performance im Vordergrund? Dieser Kerngedanke ist in verhältnismäßig binärer Form bereits durch die Linux internen Governor implementiert. Um maximale Performance zu erhalten, wird der „performance“-Governor aktiviert und wenn die Energieersparnis an erster Stelle steht, der „powersave“-Governor gesetzt. Für die mehr oder weniger strukturierte Anpassung an das aktuelle Belastungsszenario gibt es dann noch den „conservative“-Governor, welcher sich schrittweise an die Bedürfnisse angleicht und den „ondemand“-Governor, welcher bereits ausführlich in Abschnitt 5.4.1 eingeführt wurde. All diese Governors bieten dennoch nur sehr grob granulierte Kontrolle an und können somit zwar an das Belastungsszenario angepasst die Frequenz setzen, treffen damit jedoch nicht zwangsläufig die realen Bedürfnisse des Nutzers.
Um dieses Nutzerbedürfnis abzubilden und auf eine Frequenzkonfiguration zu übertragen, wählte Anas Toma[12] ein Fuzzy Model, auf welchem der theoretische Teil dieses Abschnittes basiert. Dabei wurden zwei Eingaben betrachtet, welche anhand der Regeln zu einem

[12]https://ls12-www.cs.tu-dortmund.de/daes/de/daes/mitarbeiter/dr-ing-anas-toma.html

Faktor verrechnet werden, um die Frequenzkonfiguration anhand eines Power Models zu bestimmen. Die erste Eingabe ist die gewünschte Geschwindigkeit, durch welche angegeben wird, wie hoch der Nutzer die Performance priorisiert. Als zweite Eingabe wird die gewünschte Energieersparnis erwartet.
Das Modell ist im Fuzzy-Controller Stil nach Mamdani gehalten, welches der wohl verbreiteste Fuzzy Stil ist [27].
Dies geschieht anhand folgender Regeln:

Listing 5.1: Fuzzy Regeln

```
1 if (DesiredSpeed is Fast) and (EnergySaving is Min(DC)) then
    (FrequencyFactor is VHigh)(1)
2 if (DesiredSpeed is Fast) and (EnergySaving is Middle) then
    (FrequencyFactor is High)(1)
3 if (DesiredSpeed is Fast) and (EnergySaving is Max) then
    (FrequencyFactor is Middle)(1)
4 if (DesiredSpeed is Slow(DC)) and (EnergySaving is Max) then
    (FrequencyFactor is VLow)(1)
5 if (DesiredSpeed is Normal) and (EnergySaving is Max) then
    (FrequencyFactor is Low)(1)
6 if (DesiredSpeed is VFast) then
    (FrequencyFactor is Max)(1)
7 if (EnergySaving is Extreme) then
    (FrequencyFactor is Min)(1)
```

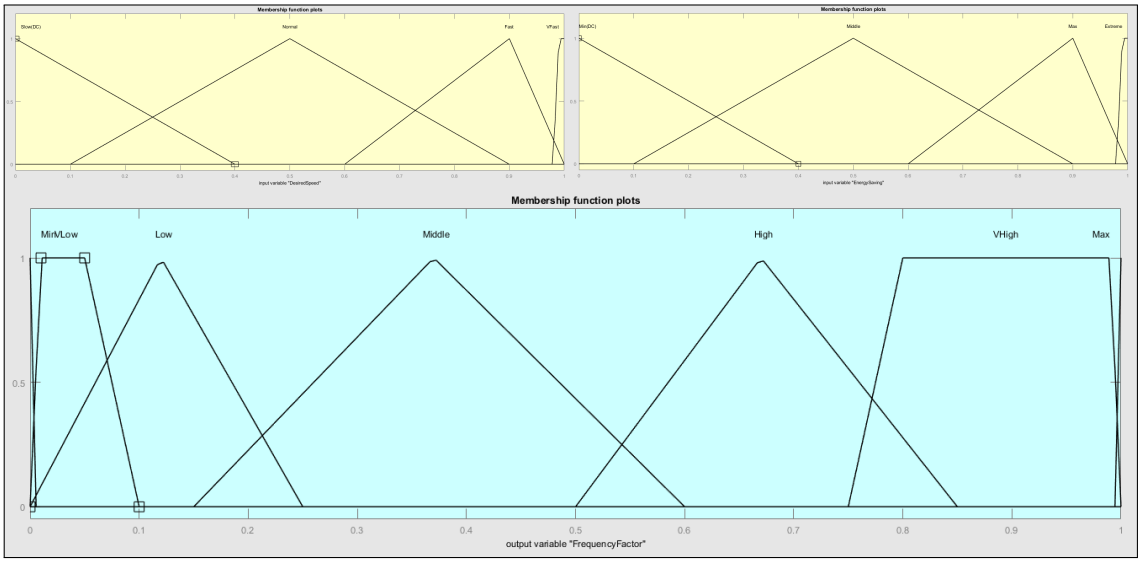


Abbildung 5.7: Zugehörigkeitsfunktionsgraphen der Eingaben DesiredSpeed und EnergySaving sowie der Ausgabe des FrequencyFactor (e.D.)

Diese Zugehörigkeitsfunktionen bilden ab, wie die flächenbasierende Informationsbasis für die Regeln gegeben ist. Zu beachten sind die zwei Spikes, also kurzzeitige Anstiege an den Rändern des Output-Graphen. Sie stellen die absolute Priorisierung dar, um einen sauberen, jedoch direkten Übergang zwischen Flexibilität und der fixen Extremwert Entscheidung zu schaffen.

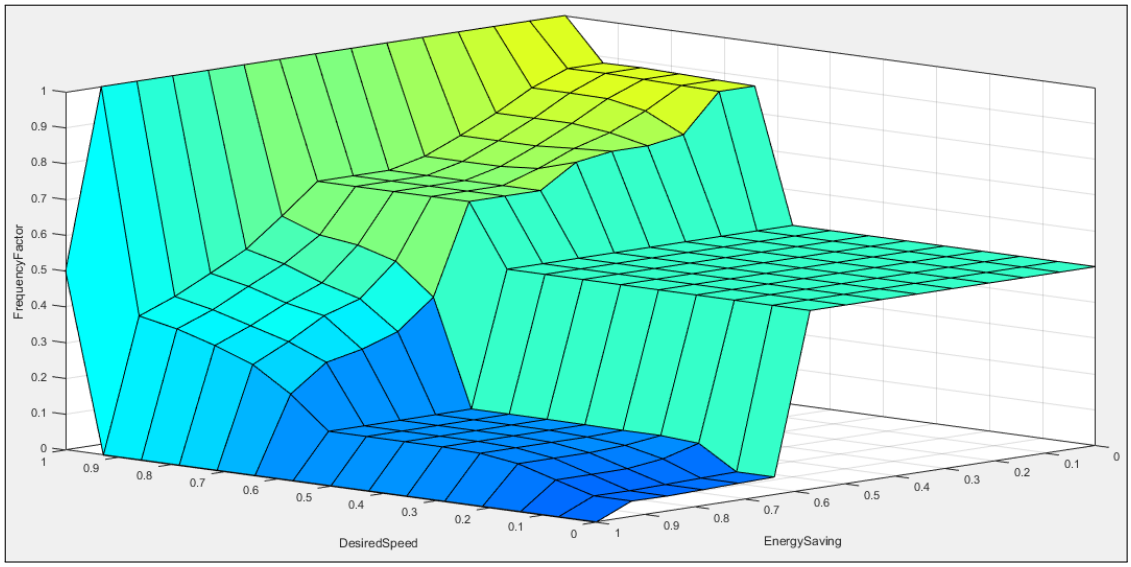


Abbildung 5.8: Abhängigkeit zwischen den Eingaben und der Ausgabe (e.D.)

In Abbildung 5.8 ist relativ gut nachvollziehbar, welche Bereiche zu vollständig dynamischen Anpassungen führen und welche zu Ebenen bzw. festen Werten. Durch ein solches Model ist es für den Nutzer möglich, anhand zwei getrennter Regler bzw. Werte seine aktuellen Bedürfnisse dem System mitzuteilen. Auch können diese Regler zusätzlich nur systemeigenen Gegebenheiten beeinflusst werden, wie z.B. das ab 5% Akku und ohne direkter Nutzerinteraktion der Energiesparfaktor maximiert und dem Nutzer bei der ersten Interaktion dies mitgeteilt wird. Dies führt dazu, dass eine solche Modellierung ein sehr flexibles Verhalten bietet, welches auch mit Fremdsystemen sehr gut interagiert. Ein Mappen des Outputfaktors geschieht anhand der Daten aus Abschnitt 5.3.

5.5 Zusammenfassung

Es wurden Power Management Techniken eingeführt und teilweise evaluiert. Dabei stellte sich nicht immer nur die Komplexität der Energieverläufe, sondern auch die unintuitive Wechselwirkung in einer heterogenen Berechnungslösung, in den Vordergrund.
Final lässt sich jedoch sagen, dass eine heterogene Berechnungslösung funktioniert und eine flexible Energieanpassung bietet. Dies lässt sich zwar per externer Energiesparmaßnahmen optimieren, jedoch führt bereits die Grundstruktur des globalen Schedulings zur Behebung vieler Probleme, welche in herkömmlichen Berechnungslösungen auftreten.

Kapitel 6

Fazit und Ausblick

In diesem Kapitel wird ein zusammenfassender Rückblick auf die Arbeit gegeben. Außerdem werden die Ergebnisse noch einmal konkretisiert. Dabei wird auf die Zusammenhänge der einzelnen Abschnitte eingegangen.

6.1 Fazit

Zusammenfassend lässt sich sagen, dass Power Management, egal auf welchem Level es betrieben wird, ein komplexes Thema mit vielen Facetten ist. Es stellt immer eine Gratwanderung zwischen einer Verbesserung und einem Overhead dar, welches die Energieersparnis wieder negiert. Während der Arbeit hat sich immer wieder herausgestellt, dass kaum eine Entwicklung beim Power Management linear ist. So ziemlich jede Veränderung führt entweder zu einem kurzzeitigem exponentiellen oder sogar diskreten Verhalten. Die verbrauchte Energie [13] unterliegt diesem Verhalten auch. Es ergab sich, dass bei jeder Struktur Grenzen bzw. Wechselgrenzen zu beachten sind. Jede Verbesserung funktioniert nur bis zu einer bestimmten Obergrenze und führt danach entweder zu einer Verschlechterung oder stagniert. Die Analyse der Laufzeit relativ zur Frequenz und dem daraus resultierenden Energieverbrauch verdeutlichte erneut, dass durch die Verringerung der Frequenz aufgrund der daraus resultierenden Verlängerung der Ausführungszeit, der Energieverbrauch sein relatives Verhalten bei bestimmten Grenzen ändert [5]. Der in anderen Arbeiten behandelte $s_{critical}$ genannte Punkt, war bei dieser Analyse nicht ersichtlich. Zwar verhielt sich das Kurvenverhalten weitestgehend gleich, jedoch gab es den, bei anderen Arbeiten gemessenen, erneuten Anstieg nie [21, 22]. Diese resultiert vermutlich aus der Mischung von „big.LITTLE MP“ mit realen Anwendungsdaten. Dadurch wird die Betrachtung theoretischer Extremfälle zwar nicht obsolet, jedoch zeigt es, dass diese Fälle nicht direkt in Bezug gesetzt werden können.

[13] In dieser Arbeit wurde diese Energie immer durch die von der Überwachungshardware gemessene Leistung multipliziert, mit der mittels Perf gemessenen Laufzeit berechnet.

Auch stellte sich immer wieder der große Nachteil von „Userspace“ Power Management heraus. Die Interaktion mit dem „CPUFreq“ eigenen „sysfs“-Interface führt zu einem Overhead, welcher bei „InKernel“-Verfahren in vielen Fällen vermieden werden kann. Dies wurde bereits bei der Entwicklung des „ondemand“-Governors in Betracht gezogen [23, S. 219 bzw. S. 5]. Jedoch führt diese Überlegung nicht dazu, dass Lösungen, welche im „Userspace“ umgesetzt wurden, per se falsch oder überholt sind, sondern eher, dass diese, wenn sie im Kernel umgesetzt werden würden, noch besser funktionieren würden. Es ist sowohl möglich, dass Verfahren, welche im „Userspace“ zu einer Verschlechterung führen, im Kernel plötzlich doch funktionieren, als auch, dass Verfahren, welche ursprünglich schon einen Vorteil brachten, sich noch weiter verbessern. Das Problem, dass im „Userspace“ viele Mechaniken ineinander laufen und somit jedwede Messung beeinflussen, lässt sich nur mittels langfristiger Analyse in den Griff kriegen. Allein der Faktor des thermalen Abbremsens kann schnell zu fehlerhaften Daten führen, welche, sollte auf der Basis dieser Daten gearbeitet werden, zu unvorhersehbaren Ergebnissen führen.
Final haben sich heterogene Systeme jedoch als nützliche Stuktur herausgestellt, welche das Power Management selbst zwar verkomplizieren, jedoch durch ihre eigene Struktur bereits zu einer Verbesserung führen.

6.2 Ausblick

In dieser Arbeit wurde bisher immer nur globales Scheduling betrachtet, welches bei ARM big.LITTLE Prozessoren auch „big.LITTLE MP“ genannt wird [11]. Dabei werden im Resource Mapping alle Kerne und somit auch die unterschiedlichen Kern-Arten/Ausprägungen (vgl. Abb. 6.1) auf derselben Ebene behandelt. Es findet auch keine Gruppierung nach Kernel Art bzw. Insel statt und es werden intern Tasks zwischen Kernen migriert. Durch diese Migration können Task bei freien Ressourcen auf die performanteren Berechnungslösungen verlagert werden.

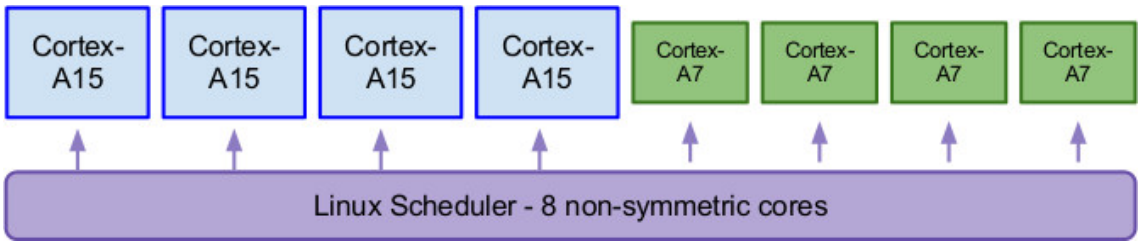


Abbildung 6.1: Globales Task-Scheduling[14]

Unterschiedliche, an heterogene Berechnungslösungen angepasste, Verfahren könnten mit den in dieser Arbeit vorgestellten Methodiken unterschiedliche Ergebnisse erzielen. Dabei wird dies voraussichtlich mit unterschiedlichen Verfahren zu verschiedenen Ergebnissen

[14] Quelle: `https://www.linaro.org/blog/hardware-update/big-little-software-update/`

führen. Nicht in allen Fällen wird eine solche Anpassung zu einer Verbesserung des gesamten Zustandes führen [28]. In einer weiteren Arbeit könnte die Eignung unterschiedlicher Resource Mapping Techniken, wie EIM[15], STM[16], HTM[17] etc. oder auch diesen übergeordnete Techniken, wie „In Kernel Switching“, dessen Aufbau in Abbildung 6.2 beschrieben ist, analysiert werden.

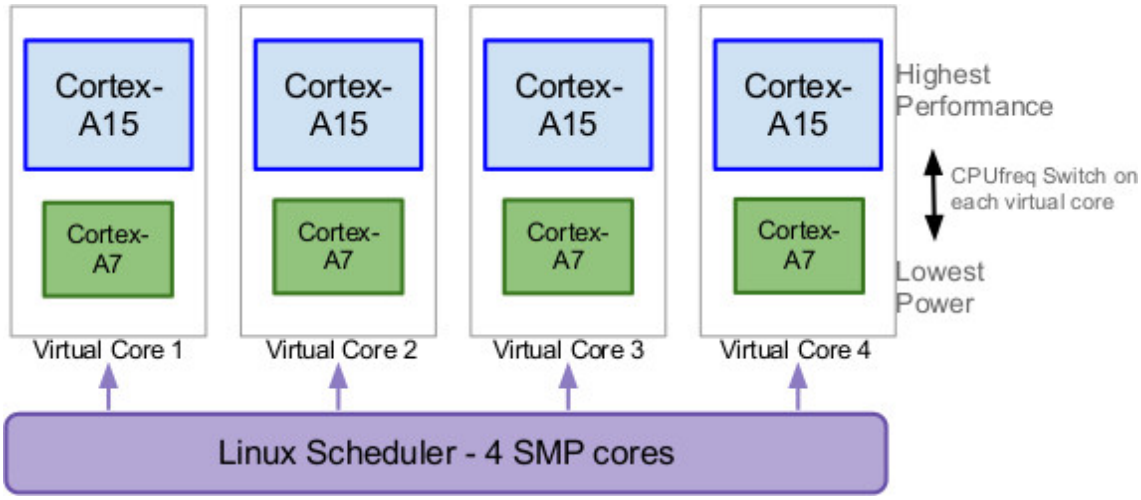


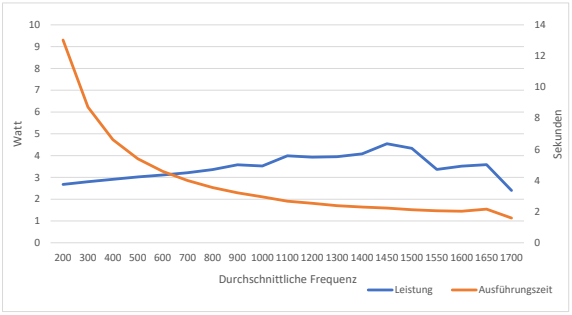
Abbildung 6.2: In Kernel Switcher[18]

Auch wäre es ein interessanter Ansatz, die problematischsten Komponenten aller Benchmarks zu analysieren und diese durch spezifisch angepasste heterogene Berechnungslösungen abzudecken. Welche zusätzlichen spezifischen Berechnungseinheiten könnten im Alltag wirklich einen Geschwindigkeits- bzw. Energieverbrauchsvorteil bringen und dies nicht nur in Sonderfällen, sondern bei einer Vielzahl von Anwendungen?
Ein weiterer interessanter Aspekt wäre die Untersuchung der Unterschiede zwischen „Userspace“ und „InKernel“ Implementierung des selben Algorithmus bzw. derselben Grundtechnik. Wie groß fällt dieser Unterschied wirklich aus und gäbe es Möglichkeiten dies durch Funktionen im Kernel zu verbessern und so „Userspace“ Implementierungen generell noch konkurrenzfähiger zu machen?

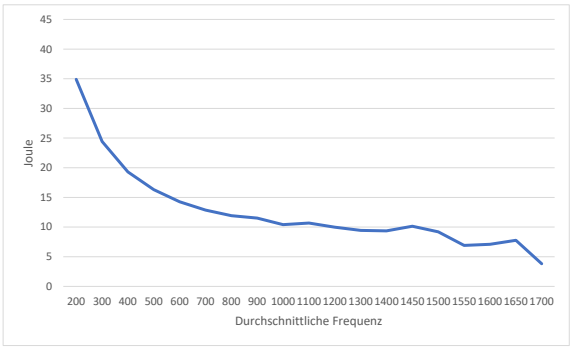
[15] EIM steht für „Energy-aware Iterative multi-application Mapping“. Dies ist ein Energiebudget basierendes Resource-Mapping Verfahren, welches auf dem aktuellen Verbrauch basiert, wobei Restbudget sowie die maximale Restnutzung in Betracht gezogen werden [25, S. 9 ff.].
[16] STM steht für „Scenario-based runtime Task Mapping“. Es ist ein Resource Mapping Verfahren, welches die Task-Map nur anhand einer Anwendung anpasst. Dabei wird die Anwendung mit der höchsten Abweichung gewählt [25, S. 13 ff.].
[17] HTM steht für „Hybrid Task Mapping Algorith“. Dies ist ein Algorithmus, welcher eine optimierte Kombination des EIM und STM Verfahrens ist [25, S. 16 ff.].
[18] Quelle: `https://www.linaro.org/blog/hardware-update/big-little-software-update/`

Anhang A

Weitere Informationen

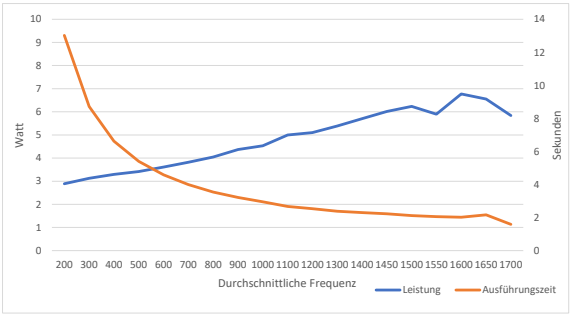


(a) Verlauf der minimalen Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)

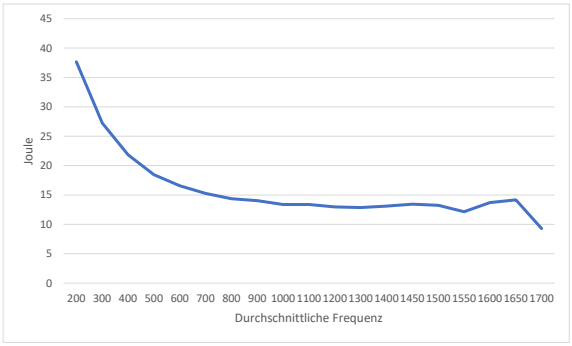


(b) Verlauf des minimalen Energieverbrauchs pro Durchlauf (e.D.)

Abbildung A.1: Betrachtung der minimalen Werte eines einheitlichen Durchlaufs

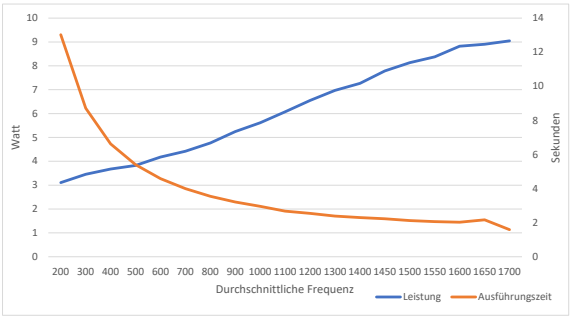


(a) Verlauf der durchschnittlichen Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)

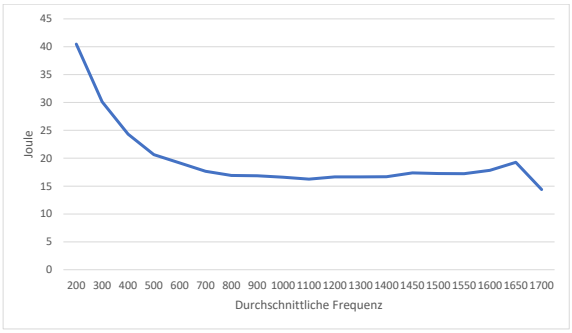


(b) Verlauf des durchschnittlichen Energieverbrauchs pro Durchlauf (e.D.)

Abbildung A.2: Betrachtung der durchschnittlichen Werte eines einheitlichen Durchlaufs



(a) Verlauf der maximalen Leistungsaufnahme und Ausführungszeit während eines Durchlaufs (e.D.)



(b) Verlauf des maximalen Energieverbrauchs pro Durchlauf (e.D.)

Abbildung A.3: Betrachtung der maximalen Werte eines einheitlichen Durchlaufs

Abbildungsverzeichnis

Abkürzungsverzeichnis

Abb. Abbildung

ADPCM „Adaptive Differential Pulse Code Modulation“

AES „Advanced Encryption Standard“

API „Application Programming Interface“

ASIC „Anwendungsspezifische integrierte Schaltung“

AVC „Advanced Video Coding“

bspw. beispielsweise

bzw. beziehungsweise

ca. circa

CPU Central Processing Unit

CRC32 „Cyclic Redundancy Check 32-bit“

d.h. das heißt

DRAM Dynamic Random Access Memory

DVFS „Dynamic Voltage and Frequency Scaling“

e.D. eigene Darstellung

etc. et cetera

f. folgende

ff. fortfolgende

FFT „Fast Fourier Transformation“

FIMI „Frequent Itemset Mining“

FP-growth „Frequent Pattern-growth“

GSM „Global System for Mobile Communications“

IFFT „Inverse Fast Fourier Transformation“

I/O Ports Input/Output Ports

ISpell „International Spell“

JSON „JavaScript Object Notation“

LAME „Lame Ain’t an MP3 Encoders“

MAD „MPEG Audio Decoder“

PGP „Pretty Good Privacy“

RMS „Recognition, Mining and Synthesis“

S. Seite

SHA-1 „Secure Hash Algorithm“

SPM „Smoothed Particle Hydrodynamics“/„Geglättete Teilchen-Hydrodynamik“

SUSAN „Smallest Univalue Segment Assimilating Nucleus“

.tiff „Tagged Image File Format“

USB Universal Serial Bus

VIPS „VESARI Image Processing System“

z.B. zum Beispiel

Algorithmenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[1] *PARSEC-Publications.* http://parsec.cs.princeton.edu/publications.htm, 7 2017.

[2] Aalsaud, Ali, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang und Alex Yakovlev: *Power-Aware Performance Adaptation of Concurrent Applications in Heterogeneous Many-Core Systems.* In: *International Symposium on Low Power Electronics and Design*, 2016.

[3] (Alias), Cbienia: *PARSEC.* http://wiki.cs.princeton.edu/index.php/PARSEC, 8 2010.

[4] Alippi, Cesare: *Intelligence for Embedded Systems: A Methodological Approach.* Springer International Publishing Switzerland, 2014.

[5] Arega, Frehiwot Melak, Markus Haehnel und Waltenegus Dargie: *Dynamic Power Management in a Heterogeneous Processor Architecture.* In: *ARCS 2017 - 30th International Conference on Architecture of Computing Systems*, 2017.

[6] Bhagwat, Ashwini: *Methodologies and tools for computation offloading in heterogenous multicores.* Master Thesis, 2015.

[7] Bienia, Christian: *Benchmarking Modern Multiprocessors*, 01 2011.

[8] Brodowski, Dominik: *CPUFreq.* https://www.kernel.org/doc/Documentation/cpufreq/user-guide.txt, 7 2017.

[9] Do, Thanh, Suhib Rawshdeh und Weisong Shi: *pTop: A Process-level Power Profiling Tool*, 2013.

[10] Dubey, Pradeep: *Recognition, Mining and Synthesis Moves Computers to the Era of Tera.* 2005.

[11] Grey, George: *big.LITTLE Software Update.* https://www.linaro.org/blog/hardware-update/big-little-software-update/, 7 2013.

[12] Guthaus, Matthew, Jeff Ringenberg, Todd Austin, Trevor Mudge und Richard Brown: *MiBench.* http://vhosts.eecs.umich.edu/mibench/, 7 2017.

[13] Hardkernel co., Ltd.: *Odroid-SmartPower*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137361754360, 05 2017.

[14] Hardkernel co., Ltd.: *Odroid-XU3*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127, 05 2017.

[15] Hardkernel co., Ltd.: *Odroid-XU4*. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825, 05 2017.

[16] Ma, Jun, Guihai Yan, Yinhe Han und Xiaowei Li: *An Analytical Framework for Estimating Scale- Out and Scale-Up Power Efficiency of Heterogeneous Manycores*. IEEE Transactions on Computers, 2015.

[17] Mittal, Sparsh und Jeffrey Vetter: *A Survey of CPU-GPU Heterogeneous Computing Techniques*. ACM Computing Surveys, 2015.

[18] Muhammad Rehman Zafar, Muhammad Asfand-e-Yar: *Scheduling on Heterogeneous Multi-core Processors Using Stable Matching Algorithm*. International Journal of Advanced Computer Science and Applications, 2016.

[19] Munawar, Waqaas, Heba Khdr, Santiago Pagani, Muhammad Shafique, Jian-Jia Chen und Jorg Henkel: *Peak Power Management for Scheduling Real-time Tasks on Heterogeneous Many-Core Systems*. 2014.

[20] Owen, J. Michael, Jens V. Villumsen Paul R. Shapiro und Hugo Martel: *Adaptive smoothed particle Hydrodynamics: Methodology II*. 1998.

[21] Pagani, Santiago und Jian-Jia Chen: *Energy Efficient Task Partitioning based on the Single Frequency Approximation Scheme*. Real-Time Systems Symposium, 2013.

[22] Pagani, Santiago und Jian-Jia Chen: *Energy Efficiency Analysis for the Single Frequency Approximation (SFA) Scheme*. ACM Transactions on Embedded Computing Systems, 2014.

[23] Pallipadi, Venkatesh und Alexey Starikovskiy: *The Ondemand Governor*. Intel Open Source Technology Center, 2006.

[24] Qin Lv, William Josephson, Zhe Wang, Moses Charikar und Kai Li: *Ferret: A Toolkit for Content-Based Similarity Search of Feature-Rich Data*. 2006.

[25] Quan, Wei und Andy D. Pimentel: *A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs*. ACM Transactions on Embedded Computing Systems, 14, 2015.

[26] Sauter, Marc: *SMARTPHONE-PROZESSOREN: Krieg der Kerne.* https://www.golem.de/news/smartphone-prozessoren-krieg-der-kerne-1410-109202-5.html, 10 2014.

[27] Toma, Anas: *Fuzzy Logic.* Presentation, 6 2012.

[28] Yang, Liang, Pavel Ghosh, Devesh Sinha, Arunabha Sen und Andrea Richa: *Resource Mapping and Scheduling for Heterogeneous Network Processor Systems.* In: *ACM symposium on Architecture for networking and communications systems*, 2005.

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift