Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 326

Energieeffiziente Ausführung von qualitätsbewussten Algorithmen für Mobile Simulationen

Dominik Schreiber

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer/in:	Dipl.-Inf. Christoph Dibak
Beginn am:	12. April 2016
Beendet am:	12. Oktober 2016
CR-Nummer:	I.6.3

Kurzfassung

Der Energieverbrauch mobiler Geräte ist ein wichtiger Aspekt des modernen technologischen Alltags, da Mobilgeräte nur begrenzte Energie zur Verfügung haben, jedoch häufig im Dauereinsatz sind und dabei auf Abruf anspruchsvolle Berechnungen durchführen. Ein möglicher Ansatz für bessere Energieeffizienz ist hierbei der Einsatz qualitätsbewusster Algorithmen, um den Ressourcenverbrauch unter Verringerung der Qualität in einem akzeptablen Bereich zu halten.

Die vorliegende Arbeit befasst sich damit, wie numerische Berechnungen in einer qualitätsbewussten Implementierung derart ausgeführt werden können, sodass sie eine dynamische, zur Zeit der Berechnung bekannte Energieschranke einhalten und die Qualität dabei anpassen. Zunächst werden qualitätsbewusste Algorithmen und deren wichtigste Eigenschaften als Grundlage erörtert. Verwandte Arbeiten, welche sich ähnlicher Problemstellungen annehmen, werden thematisiert und von der vorliegenden Arbeit abgegrenzt.

Anhand einer Voruntersuchung zu Simulationsberechnungen wird festgestellt, dass bei einfachen Berechnungen ohne weitere Schnittstellen in guter Näherung ein proportionaler Zusammenhang zwischen Laufzeit und Energieverbrauch besteht. Als Konsequenz wird ein Verfahren vorgeschlagen, das sich diesen Zusammenhang zunutze macht und mit nur wenigen Energiemessungen die Berechnung eines akkuraten Energiemodells erlaubt, das einfache Vorhersagen über den Energieverbrauch bestimmter Ausführungen ermöglicht.
Auf Grundlage des initialen Modells wird anschließend ein erweitertes, auf Energiezuständen basierendes Modell erarbeitet, das für mehrstufige Berechnungen verschiedener Art eingesetzt werden kann.

Die Evaluation des initialen Modells ergab abhängig von der Parameterwahl im relevanten Bereich mittlerer bis hoher Qualität Abweichungen zwischen 0% und 20%, teilweise bedingt durch die Implementierung der Messungen. Das erweiterte Modell wurde einzeln sowie im Vergleich mit dem initialen Modell evaluiert und ergab akkurate Ergebnisse mit durchschnittlicher Abweichung von 4,1% und Einzelabweichungen zwischen 0% und 12%. Der Vorteil des erweiterten Modells gegenüber dem initialen konnte identifiziert und begründet werden.

Abschließend werden mögliche Ansätze für zukünftige Forschungsarbeiten beschrieben, für welche die erarbeiteten Modelle verwendet und erweitert werden können.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation: Energieeffizienz von Mobilgeräten

Durch den Einzug mobiler digitaler Geräte, die hohe Leistungsfähigkeit mit umfassender und allgegenwärtiger Vernetzung verbinden, hat der Alltag in der globalisierten Welt eine Revolution erfahren. Ein allgegenwärtiger und ständiger Zugang zu den vielfältigen Anwendungen der Mobilgeräte (darunter persönliche Organisation, Kommunikation, Information und Unterhaltung) wird heute von den meisten Anwendern als selbstverständlich wahrgenommen. Umso mehr fällt es daher auf, wenn das Gerät einmal nicht zur Verfügung steht.

Der Unterschied des Energieverbrauchs mobiler Geräte gegenüber herkömmlicher, stationärer Rechner besteht darin, dass der Einsatz ohne externe Stromversorgung den hauptsächlichen Anwendungsfall des Geräts darstellt. Das bedeutet, dass der Energieverbrauch nicht allein aus Gründen der Wirtschaftlichkeit und Umwelt ein Kriterium darstellt, sondern zusätzlich die Benutzbarkeit des Geräts einschränken kann; nämlich dann, wenn die gewünschte Nutzungsdauer die durch den vorhandenen Akku mögliche Nutzungsdauer übersteigt.

Hersteller müssen bei der Wahl des Akkus Faktoren wie Gewicht, Kosten, Lebensdauer und Ladezeit berücksichtigen; eine hohe Kapazität und Qualität des verbauten Akku ist damit nicht zwangsläufig die höchste Priorität. Zudem führte eine ungleiche Entwicklung zwischen Smartphone-Features und Akku-Technologie zu schwindenden Laufzeiten der Geräte: auf Seiten der Möglichkeiten von Smartphones war eine „Revolution“ zu verzeichnen, während die Entwicklung von Akkus von eher langsamer „Evolution“ geprägt ist [Ked12]. Die mittlerweile große Produktreihe der externen Akkus für Mobilgeräte [Hil16] bestätigt das dringende Bedürfnis der Nutzer nach zusätzlicher Akku-Kapazität.

Auch Entwickler von mobilen Anwendungen müssen eine Vielzahl von Kriterien berücksichtigen, um ein effektiv und effizient nutzbares Produkt zu entwickeln. Als eines der wichtigsten Kriterien muss dabei neben Aspekten wie der Bedienbarkeit und der Schnelligkeit die Energieeffizienz berücksichtigt werden. Dies zeigt sich unter anderem darin, dass sich in im Google Play Store mehrere Apps mit über einer Million Downloads befinden, deren bekundeter Hauptzweck in der Überwachung und der Verringerung des Batterieverbrauchs liegt[1]. Indes

[1] Abgerufen am 22.05.2016: *DU Battery Saver & Batteriespar* von DU APP STUDIO mit 10 565 611 Downloads, *Battery Doctor (Battery Saver)* von Cheetah Mobile Inc. mit 7 776 500 Downloads, und *Akku - Battery* von MacroPinch Tools mit 1 047 220 Downloads.

kann eine zusätzlich installierte App auf einem Mobilgerät zwar gewisse Prozesse überwachen, jedoch kaum den inhärenten Energieverbrauch einzelner Programme reduzieren, ohne diese in ihrer Ausführung einzuschränken. Gerade bei aufwendigen Anwendungen, deren Betrieb fortlaufende Berechnungen bedingt, kann der Batterieverbrauch hinreichend groß sein, sodass das Mobilgerät nicht genügend Energie zur Verfügung hat, um einen einzelnen Arbeitstag im Sinne des Nutzers zu arbeiten.

Entwickler stehen also vor der Herausforderung, mobile Applikationen derart zu entwickeln, dass ihre Ausführung den verbleibenden Akku-Ladestand im Rahmen ihrer Möglichkeiten schont. Eine Anwendung, die sich der aktuellen Energiesituation bewusst ist, kann durch Nutzung eines bestehenden Energiemodells und/oder vorheriger Erfahrungen sogar ihre Berechnungen derart anpassen, dass die praktisch vorgegebene Energieschranke eingehalten wird und somit die Funktion des Mobilgerätes über den gewünschten Zeitraum hinweg (etwa einen Tag lang) gewährleistet werden kann.

Die sogenannte *Augmented Reality*-Technologie hat Einzug in die Nutzung mobiler Geräte gehalten, wie unter anderem durch die erfolgreiche App „Pokémon Go“. Hier werden Objekte in die Realität des Nutzers eingeblendet und dazu weitere energieaufwendige Module wie GPS und Netzwerkdaten verwendet. Und es zeigt sich ebenso, dass der Akku häufig sehr schnell geleert wird, was sich negativ auf das Nutzererlebnis auswirkt [Eik16]. Auch abseits der Unterhaltungsbranche finden sich sinnvolle Anwendungen für *Augmented Reality*, wie in folgender Problemstellung skizziert wird.

Ein Mobilgerät, das dem Träger *Augmented Reality* ermöglicht (also in die dreidimensionale visuelle Realität zusätzliche, virtuelle Elemente einblendet), wird dazu verwendet, zukünftige Gebäude am Bauort im Voraus zu sehen, um ihre Planung und Entstehung zu unterstützen. Eine Anwendung auf dem Mobilgerät, die dies leistet, muss entweder andauernd numerische Berechnungen durchführen, um das Objekt in korrektem Abstand, Drehwinkel und räumlicher Zerrung darzustellen, oder aber einen ständigen Kommunikationskanal mit einem externen Rechner aufrechterhalten, durch den das Mobilgerät Positionsdaten sendet und resultierende Bilddaten empfängt. In beiden Fällen handelt es sich um ein energetisch aufwendiges Programm, das bei naiver Implementierung innerhalb kurzer Nutzungsdauer zu einem leeren Akku führen kann.
Ein möglicher Ansatz wäre nun hier den Energieverbrauch des Geräts zu berücksichtigen, sodass die Anwendung davon ausgehend Abstriche bei der Generierung der einzublendenden Bilder macht, um die vorhandene Energieschranke nicht zu überschreiten. So kann – im Fall eines Kommunikationskanals zu einem externen Rechner – die Framerate und Auflösung reduziert werden, um die Bandbreite zu reduzieren, oder – im Fall der lokalen Berechnung – der Diskretisierungsgrad vergröbert werden, damit der Aufwand der nötigen Berechnungen pro Zeitschritt absinkt.

Das Beispiel zeigt, dass ein intelligenter Energiehaushalt von Mobilgeräten von großem Nutzen ist, falls deren vielfältiges Potenzial tatsächlich voll ausgeschöpft werden soll. Die Problemstel-

Abbildung 1.1: *Augmented Reality* Anwendungen bieten Nutzern eine Integration verschiedenster Informationen in die Realität. Dazu sind fortwährende, aufwendige Numerische Berechnungen notwendig[2]

lung kann im Kontext verschiedener neuartiger Technologien gesehen werden, etwa im Bereich des *Internet Of Things* und der *Wearables* (Smartwatches, Fitnessarmbänder und Weiteres).

Die vorliegende Arbeit steht im Kontext eines Forschungsansatzes hin zu qualitätsbewusster Ausführung verteilter Simulationen [DK14]. Berechnungen auf Mobilgeräten sollen zeit- und ressourceneffizient durchgeführt werden, was entweder durch eine Vernetzung mit externen Servern oder durch intelligente Berechnung innerhalb des Geräts geschehen kann. Die vorliegende Arbeit untersucht lokale Berechnungen, während das Paradigma verteilter Berechnung in anderen Arbeiten untersucht wird [DDR15].

[2]https://upload.wikimedia.org/wikipedia/commons/7/7b/MediatedReality_on_iPhone2009_07_13_21_33_39.jpg

1.2 Herangehensweise

Im Verlauf der Arbeit wird, initial ausgehend von Messwerten verschiedener qualitätsbewusster Algorithmen, ein Energiemodell aufgestellt werden, das den Energieverbrauch bestimmter Berechnungen vorhersagen kann. Darüber hinaus soll aus dem Energiemodell auch ableitbar sein, wie genau eine Berechnung durchgeführt werden sollte, damit sie eine vorgegebene Energieschranke einhält und unter dieser Bedingung das bestmögliche Ergebnis erzielt. Zudem wird anschließend an die Erarbeitung eines initialen Energiemodells untersucht, ob und wie sich die Energieverhältnisse bei etwas komplexeren Anwendungen verändern, die beispielsweise den internen Speicher nutzen.

1.2.1 Zielsetzung

Ziel der Arbeit ist, es ein Verfahren zu finden, um eine gegebene, qualitätsbewusste Berechnung derart einzubinden, sodass die Qualität für die qualitätsbewusste Berechnung gemäß einer gesetzten Energieschranke gewählt werden kann. Durch die Verwendung eines Vorhersagemodells soll die Qualität derart festgesetzt werden können, sodass die derzeit zur Verfügung stehende Energie berücksichtigt wird.

1.2.2 Ansatz

Um einen Eindruck vom Energieverbrauch von Mobilgeräten zu erhalten, werden zunächst Messungen von mobilen Simulationsberechnungen durchgeführt. Diese werden auf Besonderheiten überprüft und analysiert, insbesondere auf die Frage hin, ob anhand einiger weniger Messungen mit möglichst geringem Aufwand ein mathematisches Modell für den qualitätsabhängigen Energieverbrauch einer Berechnung gefunden werden kann. Eine der zentralen Fragen soll dabei sein, wie gut der Energieverbrauch einer Berechnung mit dessen Laufzeit korreliert – denn Zeitmessungen sind verglichen mit Energiemessungen sehr einfach durchzuführen. Ausgehend von der Art und Stärke einer solchen Korrelation soll ein Verfahren bestimmt werden, das durch intelligente Nutzung des Laufzeitprofils mit nur wenigen Energiemessungen das geforderte Vorhersagemodell errechnet.

1.3 Aufbau der Ausarbeitung

Im folgenden Kapitel 2 werden die zugrundeliegenden Techniken und Theorien in Form der qualitätsbewussten Algorithmen vorgestellt und verwandte Arbeiten thematisiert.

Anschließend folgt in Kapitel 3 das verwendete Systemmodell und die Anforderungen an die erbrachte Lösung.

Im Kapitel 4 werden empirische Voruntersuchungen in Form von Energiemessungen eines Android-Smartphones präsentiert.

Der Entwurf (Kapitel 5) beschreibt das in zwei Schritten entwickelte Energiemodell und die daraus resultierenden Möglichkeiten der Energieeffizienz qualitätsbewusster Algorithmen.

Details über die entwickelte Anwendung, die verwendeten Algorithmen sowie die Aufbereitung der Messungen folgen in Kapitel 6.

In Kapitel 7 folgt die Evaluation des entwickelten Modells, erkannte Herausforderungen und die Eignung der erarbeiteten Methodik für verschiedene Anwendungsfälle.

Kapitel 8 schließt die Arbeit mit einer Zusammenfassung der ermittelten Ergebnisse, deren Konsequenzen, sowie mit dem Blick auf mögliche weitere Arbeiten in Bezug auf die Thematik ab.

2 Grundlagen und Verwandte Arbeiten

Im Folgenden werden zentrale Grundlagen der Thematik vermittelt und verwandte Arbeiten zu den bearbeiteten Themen vorgestellt.

2.1 Qualitätsbewusste Algorithmen

In vielen Anwendungsbereichen werden keine exakten Resultate für bestimmte Problemstellungen benötigt. Ein wichtigeres Ziel ist die Wahl eines guten Mittelwegs zwischen der Qualität des Ergebnisses und des Aufwands der Berechnung, sodass ressourcensparende Berechnungen durchgeführt werden können, die den Anforderungen an die Qualität dennoch genügen. Hierbei spricht man von qualitätsbewussten Algorithmen – Algorithmen, die ein bestimmtes Ergebnis approximieren und dabei in der Lage sind von außen vorgegebene Schranken auf Kosten der Güte des Ergebnisses einzuhalten [Zil96]. Diese Güte bezeichnet man dabei als *Qualität*.

Zu beachten ist, dass der Begriff im Rahmen dieser Arbeit nicht ausschließlich im Sinne von Zeitschranken, sondern auch anderen Schranken definiert wird. Allgemein wird im Folgenden von *Ressourcenschranken* ausgegangen. Im Rahmen der Arbeit handelt es sich dabei konkret um Zeit- oder um Energiebedarf. Je nach Art der Ressource kann der bisherige Verbrauch während der Berechnung „kostenfrei“ (also ohne nennenswerten Aufwand) betrachtet werden. Falls es sich bei der relevanten Ressource um Zeitbedarf handelt, ist dies für isolierte, sequentielle Berechnungen gut möglich; bei dem Energieaufwand (in Joule) einer Berechnung auf einem handelsüblichen Smartphone ist dies jedoch nicht zu erwarten.

Ein „Qualitätsbewusstsein“ eines Algorithmus lässt sich mit vorgegebener Ressourcenschranke auf verschiedene Arten realisieren. Ein recht geradliniges Verfahren besteht darin, die Schranke zu Beginn festzulegen: Ein *Contract-Algorithmus* ist ein qualitätsbewusster Algorithmus, der zu Beginn seiner Berechnung eine Ressourcenschranke erhält und das Ergebnis nur zu einer Qualität berechnet wie es die Schranke zulässt. [Zil96]

Ein iteratives Näherungsverfahren kann auf einfache Weise in einen Contract-Algorithmus umgeformt werden, falls der Ressourcenbedarf der Berechnung in Echtzeit beobachtet werden kann; es kann am Ende jeder Iteration überprüft werden, ob die Schranke in der nächsten Iteration überschritten würde und dementsprechend das bisher vorhandene Ergebnis zurückgegeben werden. Sollte der Ressourcenbedarf der einzelnen Iterationen nicht (effizient) vorhersagbar

sein, kann auch inmitten einer Iteration abgebrochen werden und das Ergebnis der zuletzt abgeschlossenen Iteration verwendet werden. Mit diesem Verfahren gehen jedoch bei Berechnungen mit wenigen, großen Iterationsschritten potentiell mehr Ergebnisse verloren als bei Berechnungen mit vielen kleinen Schritten.

2.1.1 Performance-Profile

Eine Herausforderung bei Contract-Algorithmen besteht darin, bei Verfahren, die nicht auf Iteration beruhen und keine Echtzeit-Beobachtung des Ressourcenbedarfs ermöglichen, vorherzusagen, welche Ressourcen ein Verfahren mit bestimmten Parametern verbrauchen wird. Denn nur so kann der Algorithmus den „Vertrag“ mit dessen Aufrufer einhalten.

Somit ist es von großer Bedeutung, welche Qualität durch einen bestimmten Ressourcenaufwand erreicht wird. Dies gelingt durch das Aufstellen eines *Performance-Profils*: Für einen gegebenen Qualitätsbewussten Algorithmus lässt sich eine solche Funktion

$$r \mapsto Q(r)$$

definieren, das die aufgewendeten Ressourcen r für die Berechnung auf eine Qualität der Ausgabe (in einer geeigneten Metrik) abbildet. [Zil96]

Sowohl für die Laufzeit als auch die Energie als Ressourcenform werden im Lauf der Arbeit Performance-Profile empirisch durch Messungen identifiziert und approximiert.

Abbildung 2.1 stellt ein beispielhaftes Energie-Performance-Profil dar, gemeinsam mit den Operationen, die darauf ausgeführt werden sollen. Ist das Performance-Profil bekannt, so können vor allem die beiden folgenden Operationen darauf ausgeführt werden:

- *Vorhersage:* Aus einem gegebenen Qualitätsparameter kann auf das Maß der Energie geschlossen werden, das durch Ausführung des Algorithmus mit eben jenem Qualitätsparameter verbraucht wird.
- *Optimierung:* Gegeben eine bestimmte Energieschranke kann der Qualitätsparameter ausgerechnet werden, für welchen eine Berechnung genau diese Energie verbraucht.

Liegt das Performance-Profil (beziehungsweise eine Approximation desselben) als kontinuierliche Funktion $Q(w)$ vor, so entspricht die Optimierung für den Energieverbrauch $\hat{w}$ einer einfachen Funktionsauswertung $Q(\hat{w})$. Die Vorhersage für eine Qualität q dagegen entspricht einer Nullstellensuche $Q(w) - q = 0$.

Im Verlauf der Arbeit wird es aufgrund der verwendeten Methodik eine natürlichere Vorgehensweise darstellen, eine Funktion aufzustellen, die von einem Qualitätsparameter auf den entsprechenden Energieverbrauch abbildet und nicht andersherum. Dabei handelt es sich um die Umkehrfunktion von $Q(w)$; im Folgenden wird dennoch auch diese als Performance-Profil bezeichnet, da sie denselben Zusammenhang beschreibt wie $Q(w)$ selbst.

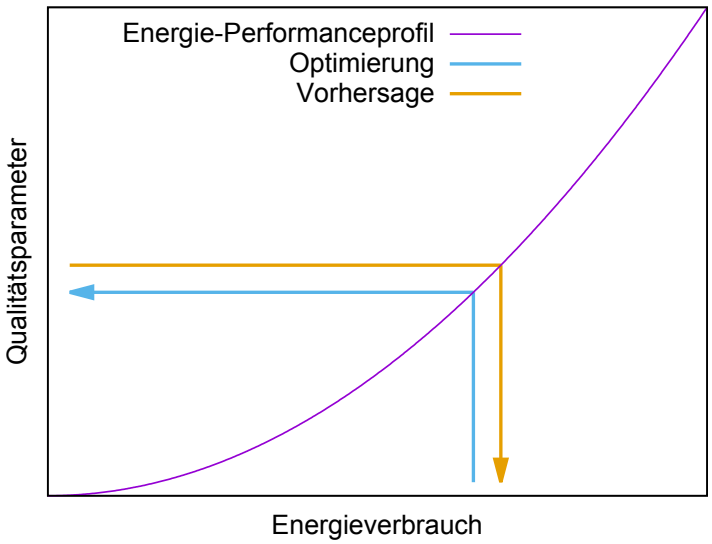

Abbildung 2.1: Vereinfachte Darstellung der Nutzung eines Performance-Profils zur Vorhersage und Optimierung des Energieverbrauchs

Monotonie

Im Rahmen der Arbeit wird angenommen, dass die Performance-Profile der behandelten Berechnungen in Abhängigkeit der Qualität monoton ansteigen. Eine höhere Qualität führt also nicht zu einem geringeren Ressourcenaufwand.

Würde diese Annahme nicht gelten, so ist ein Performance-Profil gemäß Abbildung 2.1 nicht mehr definiert, da sonst für einen bestimmten Ressourcenbedarf mehrere Qualitätsparameter existieren können. Anders ausgedrückt besitzt das soeben beschriebene „inverse“ Performance-Profil (das somit eine nicht-monotone Funktion darstellt) keine Umkehrfunktion mehr. Aus diesen Gründen kann dadurch im Allgemeinen kein eindeutiger Ressourcenbedarf ermittelt werden, der für eine gegebene Qualität benötigt wird (stattdessen kann es mehrere oder auch keinen solcher Punkte geben). Zum anderen existiert das zusätzliche Problem, dass eine Berechnung eine gegebene Ressourcenschranke nicht mehr notwendigerweise treffen sollte, um die optimale Qualität zu erzielen, sondern dafür unter Umständen weniger ausgeben sollte, um ein noch besseres Ergebnis zu berechnen.

2.1.2 Qualitätsbewusstsein herkömmlicher Algorithmen

Viele Algorithmen, die zunächst nicht inhärent approximativ sind, können in eine qualitätsbewusste Form überführt werden, wie im Folgenden beschrieben.

Iterative Algorithmen können qualitätsbewusst arbeiten, indem ein Qualitätsparameter gemäß der maximalen Anzahl an Iterationen übergeben wird. Ein einfaches Beispiel für einen sol-

chen Algorithmus ist die numerische Nullstellenbestimmung nach Newton. Da die exakte (also entweder analytische oder bis in den Bereich der Maschinengenauigkeit genaue) Lösung möglicherweise zu teuer ist, verwendet man ein Verfahren, das in jedem Schritt eine genauere Lösung findet und bei guter Konditionierung mit quadratischer Geschwindigkeit zur exakten Lösung konvergiert. Durch Angabe der maximalen Anzahl von Iterationsschritten kann die Berechnung nach Erreichen dieser Schranke abgebrochen und das aktuelle Ergebnis zurückgegeben werden. Auch Algorithmen wie die *Monte Carlo Tree Search* besitzen qualitätsbewusste Eigenschaften. Im Rahmen mehrstufiger Entscheidungsprozesse wird eine große Anzahl möglicher Optionen analysiert und nach Erreichen einer Ressourcenschranke das bisher beste Ergebnis zurückgegeben.

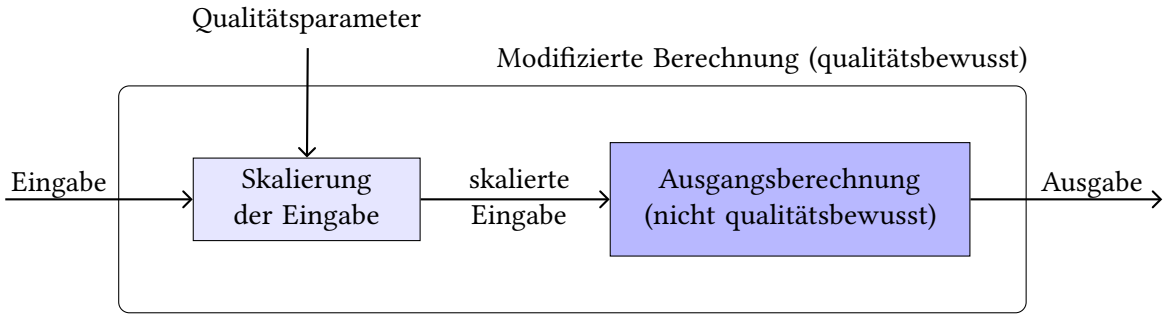

Abbildung 2.2: Mögliche Vorgehensweise zur Überführung bestimmter Algorithmen in eine qualitätsbewusste Form

Außerdem können Algorithmen, deren Eingaben selbst eine Qualität zugeordnet werden kann und die bei Eingaben geringerer Qualität weniger Ressourcen bedürfen, gemäß Abb. 2.2 in qualitätsbewusste Algorithmen umgeformt werden, indem ein zusätzlicher Berechnungsschritt zu Beginn der Berechnung eingeführt wird, der die Eingabe für die eigentliche Berechnung gemäß des übergegebenen Qualitätsparameters skaliert. Ein einleuchtendes Beispiel bietet hier jegliche Form der Bildverarbeitung: Je kleiner das Ausgangsbild, umso schneller können Algorithmen darauf operieren, jedoch wird auch das Ergebnis darunter leiden.

Unter diesen Gesichtspunkten sind für eine große Zahl konventioneller und exakter Algorithmen qualitätsbewusste Implementierungen denkbar, was umso mehr zur Untersuchung qualitätsbewusster Algorithmen motiviert.

2.2 Verwandte Arbeiten

Mehrere bestehende Forschungsarbeiten zielen auf eine Modellierung und Optimierung des Energieverbrauchs von mobilen Geräten. Dabei wird von verschiedenen Technologien Gebrauch gemacht, um *offline*, also als vorbereitenden Schritt, ein Energiemodell aufzustellen, das dann *online* verwendet und gegebenenfalls fortlaufend verbessert werden kann.

Gemeinsam haben die Techniken, dass neben dem zu modellierenden Software-Artefakt zusätzliche externe Informationen benötigt werden. Dabei kann es sich um im Voraus bekannte CPU- [HLHG13] und Systemprofile [HLHG12] oder ähnliche Abbildungen von Codeanweisungen auf den Energieverbrauch [PP16] handeln. Auch Trainingsdaten wie Ergebnisse von Energiemessungen werden verwendet [KLY+13] [BMM12].

Auf Grundlage der erworbenen Informationen wird der Programmcode häufig durch Technologien wie beispielsweise Bytecode-Profiling [HLHG12] oder Program Slicing [KLY+13] gewissen Transformationen unterworfen. Durch diese Veränderungen kann für das angepasste Programm dann zur Zeit der Ausführung die benötigte Energie vorhergesagt werden. In einigen Fällen werden die Vorhersagen stattdessen [BMM12] oder zusätzlich [PP16] durch Ressourcenabfragen zwischen einzelnen Ausführungen angepasst und optimiert.

Im Folgenden werden verschiedene bestehende Ansätze im Einzelnen vorgestellt und die vorliegende Arbeit jeweils davon abgegrenzt.

James Bornholdt et al. [BMM12] suggerieren, dass der Energieverbrauch mobiler Geräte durch Energiemodelle allein nicht hinreichend genau beschrieben und vorhergesagt werden kann, da hier die Diversität der verschiedenen Aspekte des Geräts nicht ausreichend berücksichtigt wird. Stattdessen solle ein hybrider Ansatz verwendet werden, bei dem ebenso akkurate Batteriesensoren wie auch die Modellierung der Systemaufrufe eingesetzt werden, um im Gesamten ein genaueres Bild über den Energieverbrauch zu erhalten. Es wurde ein Windows Phone verwendet; zudem wurde in empirischen Messungen ein deutlich abweichender Grundverbrauch an verschiedenen Tagen festgestellt, was in der hier vorliegenden Arbeit (mit Android-Smartphones) nicht bestätigt werden konnte. Es wird festgestellt, dass Energiemodelle große Ungenauigkeiten aufweisen können und aus praktischen Gründen kaum im Live-Betrieb einsetzbar sind. In der hier vorliegenden Arbeit wird hingegen der Energieverbrauch einzelner lokaler Berechnungen durch im Voraus vorgenommene Energie- und Zeitmessungen und ein geeignetes Energiemodell durchaus akkurat angenähert.

Yongin Kwon et al. [KLY+13] stellen das Framework *Mantis* vor, das durch eine Verknüpfung von maschinellem Lernen und Codeanalyse den Ressourcenverbrauch akkurat vorhersagen kann. Dabei werden für die Ressourcennutzung relevante *Features* (eine abstrakte Bezeichung für verschiedenste Informationen zur Berechnung) identifiziert und in einer Offlinephase mittels Trainingsdaten vorausberechnet, um ein Modell für die Vorhersage des Ressourcenverbrauchs zu erlernen. Durch *Program Slicing* wird die Berechnung auf die für das Ergebnis notwendigen Ausführungen reduziert, um die Performance zu erhöhen.
Der Ressourcenbegriff ist dort weiter gefasst und beinhaltet neben Laufzeit und Energieverbrauch auch Speichergebrauch und Netzwerkgebrauch. Zudem wird der absolute Energieverbrauch dort nicht einmalig durch Messungen am Mobilgerät bestimmt, sondern durch verschiedene Ausführungen und ein daran ansetzendes Machine Learning-Modell erlernt. Infolge dieser Eigenschaften ist das Mantis-Framework schwergewichtiger als der hier vorliegende Ansatz.

Das Mobile Distributed Computing Framework *MobiDiC* [PP16] nimmt sich, ebenso wie die vorliegende Arbeit, des Paradigmas der Approximativen Berechnungen an, um mobile, verteilte Berechnungen ressourceneffizienter zu gestalten. Wie bei *Mantis* gestaltet sich der Einsatz bei einer Anwendung zweiteilig: zunächst wird in einer Offlinephase durch *Rich Workflows* die Berechnung analysiert, wobei approximierbare Teile identifiziert werden. Während des Gebrauchs bestimmt das Framework dann anhand der aktuellen Ressourcenschranken, unter welchen Qualitätsbedingungen die Berechnung durchgeführt werden soll.
Die ausdrucksstarke Repräsentation und Analyse des Programmcodes mithilfe von Workflows in der Offlinephase zeigt, dass MobiDiC in erster Linie für aufwendige, mehrschichtige Berechnungen gedacht ist, welche potentiell über Netzwerke mit weiteren Knoten verbunden sind. Die benötigten Schritte hin zu einem qualitätsbewussten Rechnen umfassen damit verglichen mit der vorliegenden Arbeit deutlich mehr Arbeitsschritte inklusive einer umfassenden Analyse des Quellcodes. Ein weiterer Unterschied zur vorliegenden Arbeit besteht darin, dass unmittelbar vor den energieeffizienten Berechnungen ein (leichtgewichtiger) Algorithmus auf dem Mobilgerät bestimmt, welche *Teile* des Algorithmus in welcher Form ausgeführt werden; die Entscheidungsfindung hin zu einer bestimmten Qualität ist somit deutlich komplexer aufgebaut, kann dafür jedoch vermutlich auch allgemeinere Fälle abdecken als der hier erarbeitete Ansatz speziell für lokale numerische Berechnungen.

Speziell für den Energieverbrauch der CPU auf Androidgeräten wurde die Technologie *eCalc* entwickelt [HLHG12]. Sie verwendet Bytecode-Profiling, um ohne weitere Sensorik Vorhersagen über den Energieverbrauch der CPU zu treffen. Jedoch muss dort das *CPU Profil* bekannt sein, das für die vorhandene CPU einzelne Anweisungen auf einen Energiebedarf abbildet. Das Vorhandensein eines solchen Profils wird vorausgesetzt beziehungsweise die Auslieferung eines solchen von den Entwicklern der Hard- und Softwareplattform erwartet. Durch diese Voraussetzung unterscheidet sich eCalc von dieser Bachelorarbeit, da hier die atomaren Informationen über den Energieverbrauch von Berechnungen über empirische Messungen beschafft werden.

Die gleichen Autoren stellen in [HLHG13] einen weiteren Ansatz namens *eLens* vor, welcher sich derselben Problemstellung durch Programmanalysen annähert. Ähnlich zu dem in dieser Bachelorarbeit entwickelten phasenbasierten Modell werden auch dort Segmente des Programmcodes einem Energieniveau zugeordnet, um akkurat den Energieverbrauch einer gesamten Berechnung zu modellieren. Dort geschieht dies jedoch auf Ebene einzelner Instruktionen und mithilfe von gegebenen *System-Profilen*, welche über den Energieverbrauch der Instruktionen Auskunft geben. Der Ansatz der vorliegenden Arbeit hingegen wird die Berechnung auf einer abstrakteren Ebene aufteilen und dabei keinen Gebrauch von System-Profilen machen.

Die soeben vorgestellten Ansätze analysieren meist den Programmcode und wenden darauf Abstraktionen an (Rich Workflows, Bytecode-Profiling, *Features*, Analyse der Systemaufrufe). In dieser Arbeit wird im Gegensatz dazu zunächst ohne Kenntnis oder Manipulation des Programmcodes ein Modell aufgestellt werden. In einer späteren Modellerweiterung wird der Programmcode leicht modifiziert, jedoch nur um weitere Informationen über die Laufzeit

einzelner Berechnungsphasen zu erhalten. Als vorteilhaft ist bei diesem Ansatz eine hohe Unabhängigkeit von der Art der Implementierung und der Wahl der Programmiersprache[1] zu bewerten. Im Gegenzug kann auf einzelne Programmkonstrukte möglicherweise nicht mit derselben Genauigkeit eingegangen werden und es entfallen einige Möglichkeiten der Approximation der Berechnung durch Anpassung des Programmablaufs an sich.

Einen weiteren Unterschied zu bestehenden Arbeiten stellt hier die Spezialisierung auf lokale Simulationsberechnungen dar, wodurch verglichen mit allgemeinen und verteilten Berechnungen andere Eigenschaften gelten und andere Ansätze verwendbar sind, wie die späteren Untersuchungen zeigen werden.

[1] In der Voruntersuchung und der Evaluation behandelte Berechnungen wurden teilweise mit Java und teilweise mit C++ implementiert.

3 Systemmodell und Anforderungen

Das folgende Systemmodell beschreibt die Komponenten des untersuchten Systems und die Anforderungen an diese Komponenten.

3.1 Komponenten

Das System besteht aus einem Mobilgerät sowie dessen Nutzer. Deren relevante Eigenschaften werden kurz beschrieben.

3.1.1 Mobilgerät

Zentrum der Untersuchungen ist das Mobilgerät, auf welchem in einem gegebenen Zeitintervall verschiedene Dienste und Programme laufen. Dabei nimmt das Gerät die zeitabhängige Leistung $P(t)$ auf, welche integriert über das Zeitintervall den Energieverbrauch W ergibt. Diese Leistung setzt sich aus verschiedenen Komponenten des Geräts wie etwa der CPU, dem Display oder Netzwerkschnittstellen zusammen. Es wird im Folgenden jedoch nur zwischen einer Grundleistung P_0 des Handys und der Leistungsaufnahme während einer qualitätsbewussten Berechnung unterschieden; eine Aufteilung der zu einem Zeitpunkt festgestellten Leistungsaufnahme auf verschiedene Komponenten des Geräts wird nicht vorgenommen.

Zudem besitzt das Gerät einen Akku, welcher eine scharfe, obere Energieschranke für alle Anwendungen darstellt. (Schranken, die darunter liegen, können etwa dann zusätzlich auferlegt werden, falls ein bestimmter Rest-Akkustand bewahrt werden soll.) Die nutzbare Energie eines Geräts sei damit angegeben als ein bestimmtes W_{max}. Im Modell wird davon ausgegangen, dass Akku-Wiederaufladungen während des Betriebs nicht auftreten.

3.1.2 Anwendung

Bei der Anwendung, deren Energieverbrauch vorhergesagt und angepasst werden soll, handelt es sich um die Berechnung bestimmter Simulationen. Dabei wird deren Leistungsaufnahme im Rahmen der Arbeit nur den Hintergrundberechnungen der Anwendung zugeschrieben; ausgenommen ist die Interaktion des Nutzers, Eingabeverarbeitungen und Visualisierung.

Explizit besteht bei den betrachteten Anwendungen *keine Netzwerkverbindung*, es soll sich um rein lokale Berechnungen handeln. Es wird angenommen, dass während der Ausführung der behandelten Berechnungen keine weiteren Benutzer-Apps aktiv sind.

3.1.3 Umgebung

Der *Nutzer* des Geräts bestimmt, zu welcher Zeit welche Anwendung läuft. Er kann somit, abgesehen vom Grundverbrauch P_0, den Energieverbrauch indirekt steuern, ist dabei aber auf die entsprechende Energieeffizienz der Anwendungen angewiesen, die er nutzt.

Zu welcher Zeit welche Anwendungen geöffnet, aktiv verwendet und wieder geschlossen werden, wird innerhalb des Modells als nichtdeterministisch angenommen (unter den in Kapitel 3.1.2 genannten Einschränkungen). Durch Maschinelles Lernen können zwar bestimmte Gewohnheiten des Nutzers identifiziert werden, was jedoch grundsätzlich keine Garantie mit sich bringt; auch nach dem genauen Erlernen eines Tagesablaufs bleiben spontane Abweichungen davon eine Möglichkeit, die berücksichtigt werden sollte.

3.2 Eigenschaften der Berechnung

Die Berechnung der vorliegenden Anwendung soll den Anforderungen an einen qualitätsbewussten Algorithmus genügen. Insbesondere erhält die Berechnung also zu Beginn einen Qualitätsparameter $p \in \mathbb{N}$, der auf die Qualität des berechneten Ergebnisses Einfluss nimmt. Ein höherer Parameter p führt dabei zu einem höheren Ressourcenverbrauch sowie zu einer höheren Qualität des Ergebnisses. Damit lässt sich durch die Wahl von p das Verhalten der Anwendung in Hinblick auf Qualität und Ressourcenbedarf steuern.

Es wird davon ausgegangen, dass die Laufzeit der Berechnung abhängig von der geforderten Qualität monoton ansteigt; die Anwendung muss also umso länger berechnen, je höher die Qualität ist. Für zwei Ausführungen i und j mit den entsprechenden Qualitätsparametern p_i und p_j sowie Laufzeiten t_i und t_j muss gelten:

$$p_i < p_j \Rightarrow t_i \leq t_j \tag{3.1}$$

Gälte dies nicht, so träfen die Grundsätze des qualitätsbewussten Rechnens nicht mehr zu, da eine Berechnung mit mehr Ressourcen eventuell zu einem schlechteren Ergebnis führt.

Trotz dieser Forderung könnten bei Messungen aufgrund von Ungenauigkeiten und Streuungen geringfügige Abstiege des Zeitbedarfs bei eng benachbarten Qualitäten beobachtet werden; dies ist explizit erlaubt und im Entwurf des Modells zu berücksichtigen.

3.3 Anforderungen

Um die energieeffiziente Ausführung von qualitätsbewussten Simulationsberechnungen zu ermöglichen, soll eine Methodik für die Aufstellung eines Vorhersagemodells entwickelt werden.

Gegeben sei eine konkrete lokale und qualitätsbewusste Simulationsberechnung; dann soll durch Verwendung der entwickelten Methodik mithilfe verschiedener Messungen ein mathematisches Energiemodell berechnet werden können, das den Qualitätsparameter und den zu erwartenden Energieverbrauch der entsprechenden Ausführung in Bezug setzt. Das entstehende Modell soll also den Energieverbrauch der zugrunde liegenden Berechnung für verschiedene Qualitätsparameter akkurat abschätzen können. Die Methodik zur Aufstellung des Modells soll dabei möglichst automatisierbar sein.

Auf Grundlage der vorliegenden Berechnung soll das Modell einen beliebigen Qualitätsparameter p auf die Energie W abbilden können, welche das Gerät durch die Berechnung mit Eingabe des Parameters p näherungsweise verbrauchen wird (*Vorhersage*). Ebenso soll durch das Modell auf Grundlage einer vorgegebenen Energieschranke $\hat{W}$ ein Parameter bestimmt werden können, für den die Berechnung möglichst genau die Schranke trifft und dabei die Qualität maximiert (*Optimierung*).

Die „Befragung“ des Modells soll in Hinsicht auf Zeit- und Energieressourcen in beiden Auswertungsrichtungen möglichst leichtgewichtig sein, also nur einen kleinen Teil der Ressourcen benötigen, den die eigentliche Berechnung selbst benötigt.

4 Voruntersuchung

Um einen Einblick in den Energieverbrauch mobiler Geräte bei der Berechnung von Simulationen zu erhalten, wurde zu Beginn eine Voruntersuchung durchgeführt. Diese bietet die Grundlage für die Vorgehensweise im Entwurf der Arbeit.

Für ein möglichst akkurates Bild des Energieverbrauchs wurden Messungen mit externer Peripherie durchgeführt. Zwar ist es bei den in dieser Arbeit behandelten Geräten mit aktuellem Android-Betriebssystem (Android *Lollipop* oder *Marshmallow*) möglich, über die `BatteryManager`-API[1] den aktuellen Ladestand und sogar Größen wie die aktuelle Spannung und die restliche Energie im Akku zu erhalten; diese Informationen können auf Mikro-Ebene jedoch keine probaten Aussagen treffen.[2] Die Laufzeit eines einzelnen Algorithmus liegt mitunter bei mehreren Sekunden, doch um auf Ebene einzelner Ausführungen Analysen vornehmen zu können, müssen die Daten genauer und in höherer Auslösung vorliegen.

Eine Möglichkeit zur genauen Messung besteht in der Nutzung zusätzlicher Hardware, die direkt in den Stromkreis, zwischen Mobilgerät und Akku, geschalten werden kann, um in Echtzeit und mit hoher Auflösung die Leistungsaufnahme zu ermitteln. Derartige Messungen wurden durchgeführt und interpretiert wie im Folgenden beschrieben.

4.1 Verwendeter Algorithmus

Verschiedene Anwendungen auf Mobilgeräten bringen unterschiedliche Anforderungen mit sich, benötigen verschiedene Ressourcen des Gerätes unterschiedlich stark und verhalten sich daher auch in Gesichtspunkten der Energieeffizienz unterschiedlich. Im Verlauf der Arbeit stehen Algorithmen verschiedener Bereiche im Fokus, die jeweils einen klaren Praxisbezug haben und auf qualitätsbewusste Weise implementiert werden können. Die Vorstellung aller behandelten Algorithmen folgt in Kapitel 6; für die Voruntersuchung wurden Berechnungen des **Diffusion-Advection-Problems** verwendet.

Durch Anwendung Linearer Algebra kann ein numerischer Ansatz verwendet werden, um die Diffusion und Advektion von Teilchen auf einem zweidimensionalen Gitter zu berechnen.

[1] https://developer.android.com/reference/android/os/BatteryManager.html

[2] https://developer.android.com/training/monitoring-device-state/battery-monitoring.html – *You can't easily continually monitor the battery state*

Der Aufwand ist polynomiell in der Diskretisierung, der Dimension n des Gitters, welche somit auch den Qualitätsparameter darstellt, der die Berechnungsressourcen ebenso wie die Qualität des Ergebnisses beeinflusst. Die Berechnung ist damit qualitätsbewusst implementiert, und für ansteigende Qualitätsparameter n kann auch mit einem ansteigendem Ressourcenaufwand sowie mit ansteigender Güte (weil Genauigkeit) des Ergebnisses gerechnet werden.

Damit kann Diffusion-Advection in dieser Form zu einem Contract-Algorithmus erweitert werden, sofern das Performance-Profil der Berechnung in Bezug auf die zu schonende Ressource bekannt ist und im Voraus verwendet wird, um die Eingabe so zu skalieren, dass die gewünschte Schranke getroffen wird.

4.2 Energiemessung

Der Versuchsaufbau und Ablauf der Energiemessungen werden im Folgenden kurz beschrieben. Weitere Details hierzu befinden sich in Kapitel 6.3.

4.2.1 Aufbau

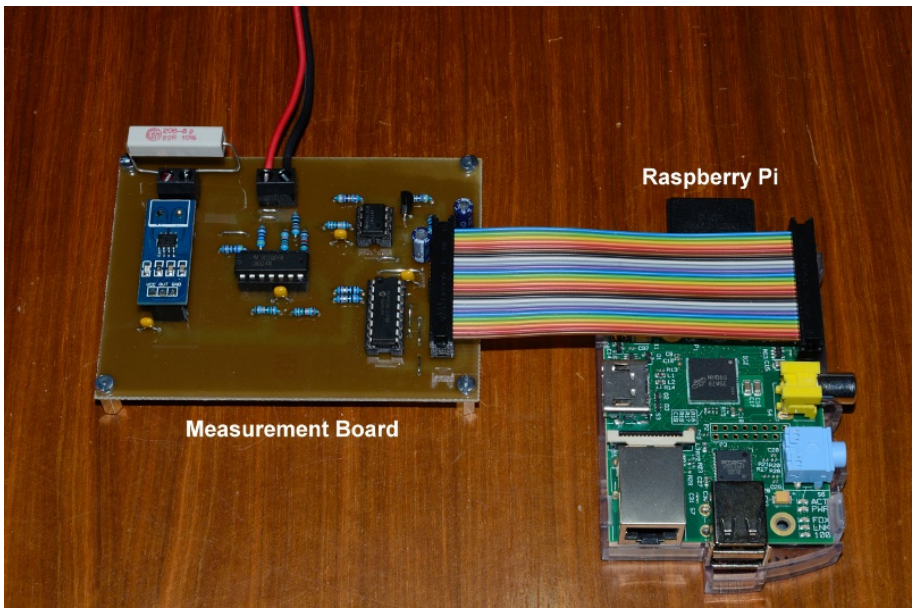

Abbildung 4.1: Aufbau der Messgeräte

Die Energiemessung wurde mithilfe eines Raspberry Pi mit Echtzeitkernel-Patch [Dür15] für das Raspbian-Betriebssystem durchgeführt, welcher an eine speziell für Messungen mobiler Geräte entworfene Schaltung [DD16] angeschlossen wurde. Eine solche Schaltung ist in Abbildung 4.1 dargestellt.

Der Akku des Mobilgeräts wird entnommen und durch einen Platzhalter ersetzt, dessen Anschlüsse über die Messschaltung zum eigentlichen Akku führen. Das Signal der Stromstärke und der Spannung wird verstärkt, gefiltert, durch einen Analog-Digital-Wandler digitalisiert und in hoher Auflösung zum Raspberry Pi übertragen. Dort werden die Daten dann geloggt und stehen für spätere Analysen bereit; so lässt sich aus den Momentanwerten für die Spannung U und die Stromstärke I die Leistung $P = UI$ errechnen, die dann in einem frei wählbaren Zeitintervall zu einem Energieverbrauch integriert werden kann.

Bei dem untersuchten Mobilgerät handelt es sich um ein Samsung Galaxy Note 4 mit einem Qualcomm Snapdragon 805 Chipsatz mit vier 2,7 GHz-Kernen, 3 GB Arbeitsspeicher und dem offiziell bereitgestellten Android-Betriebssystem der Version 6.0 (*Marshmallow*). Die Kapazität des mitgelieferten und entnehmbaren Akkus beträgt 3220 mAh. [are14]

4.2.2 Durchführung

Eine eigens für die Messung entwickelte App bietet eine einfache Touch-Schnittstelle für verschiedene Berechnungen mit verschiedenen Parametern an. Nach einer Berechnung wird eine CSV-Datei in den lokalen Speicher geschrieben, in der Daten wie die Eingabe-Parameter und die gesamte Rechenzeit geloggt werden. Für weitere Informationen zur Implementierung der App und der Auswertung der Daten siehe Kapitel 6.

4.3 Ergebnisse

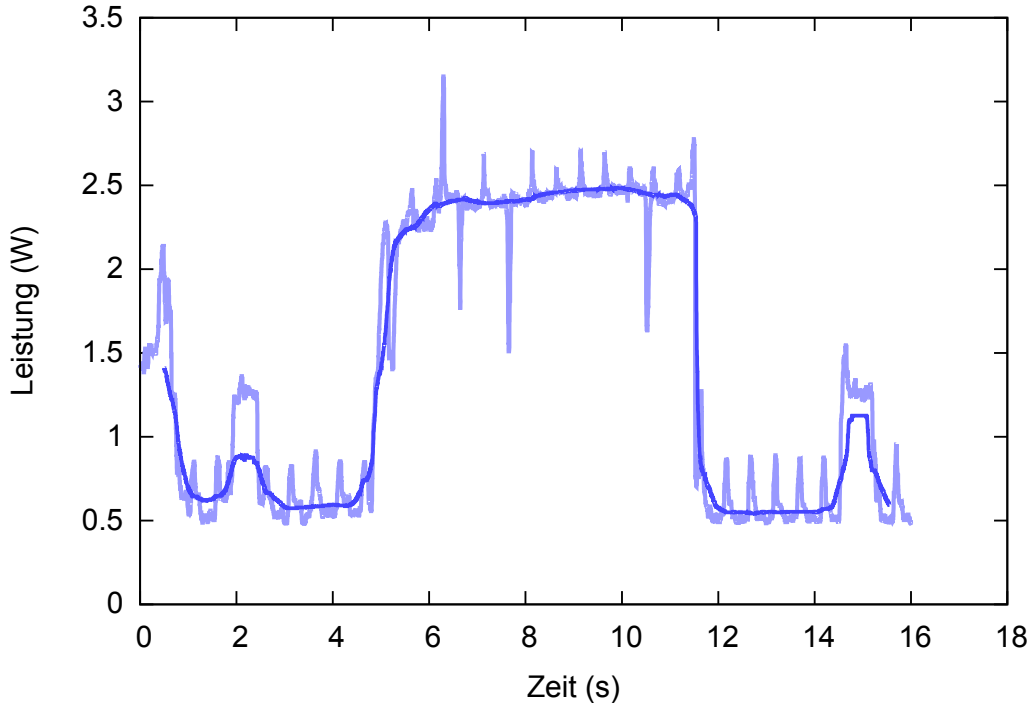

Abbildung 4.2: Leistungsaufnahme des Diffusion-Advection-Algorithmus bei feiner Diskretisierung; die Kurven stellen jeweils den Median aus 50 (hell) und aus 1000 Einzelwerten (dunkel) dar

Im Folgenden werden die Ergebnisse der Voruntersuchung in Form von geeigneten Visualisierungen vorgestellt.

Abbildung 4.2 zeigt beispielhaft die Leistungsaufnahme eines Durchlaufs des Diffusion-Advection-Algorithmus nach einer Glättung durch den gleitenden Median aus jeweils 50 und 1000 Werten. Die erhöhten Leistungswerte während den ersten drei Sekunden sowie in den letzten beiden Sekunden der Aufzeichnung entsprechen dem Starten und dem Beenden der Berechnung inklusive der Thread-Verwaltung, der Aktualisierung der Oberfläche und der Aufzeichnung von Metadaten. Vor und nach der eigentlichen Berechnung wurden Pausen erzwungen, um diese Effekte klar von der eigentlichen Berechnung zu trennen. An der Fünf-Sekunden-Marke wird eben diese Berechnung begonnen. Im weitgehend inaktiven Zustand (bis auf Hintergrundprozesse und erleuchtetes Display) beläuft sich die minimale Leistungsaufnahme auf 0,5W. Während der Berechnung besteht eine gemittelte Leistungsaufnahme von 2,5W. Eine Integration dieses Plateaus führt dann zur insgesamt verbrauchten Energie.

Während der gesamten Berechnung können periodische Leistungsspitzen mit einer Höhe von bis zu 0,4W und einer Frequenz von 4 Hz beobachtet werden. Diese Auffälligkeit kann nicht

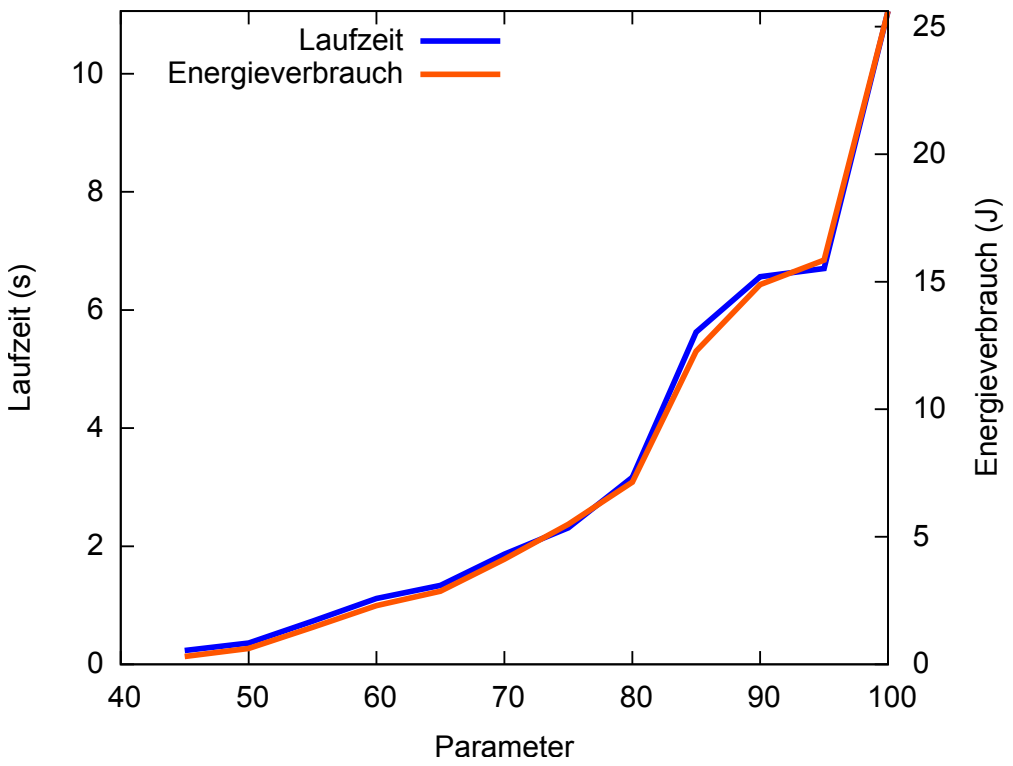

Abbildung 4.3: Laufzeit und Energieverbrauch des Diffusion-Advection-Algorithmus bei zunehmend feiner Diskretisierung

durch den Anwendungscode erklärt werden und ist daher anderen Komponenten des Geräts wie einem periodisch arbeitenden Dienst zuzuordnen.

Für ansteigenden Diskretisierungsgrad wurden jeweils Laufzeit und Energieverbrauch ermittelt. Die Messpunkte für $n < 45$ wurden ausgelassen, da hier sowohl Laufzeit als auch Energieverbrauch im Verhältnis zur Umgebung der Berechnung vernachlässigbar klein sind.

Die für jede einzelne Berechnung ermittelten Performancewerte (Laufzeit und Energieverbrauch) wurden als Grundlage für weitere Untersuchungen verwendet. Für Abbildung 4.3 wurden für jeden Qualitätsparameter der Energieverbrauch und die Laufzeit zu durchgezogenen Kurven aggregiert. Unmittelbar zeigt sich, dass bei einer geeigneten Skalierung (etwa 11s Laufzeit auf 25J Energieverbrauch) die beiden Kurven fast deckungsgleich sind, was bereits einen proportionalen Zusammenhang der Größen vermuten lässt.

Da die Laufzeit der Berechnung zuvor als polynomiell angenommen wurde und der Ressourcenverbrauch monoton ansteigen sollte, sollte die Laufzeitfunktion eine Parabel darstellen. Die Steigung der beiden Kurven, die nicht unmittelbar der erwarteten Parabel gleicht, kann indes nicht zwangsläufig als repräsentativ für die Berechnung angesehen werden, da für jeden

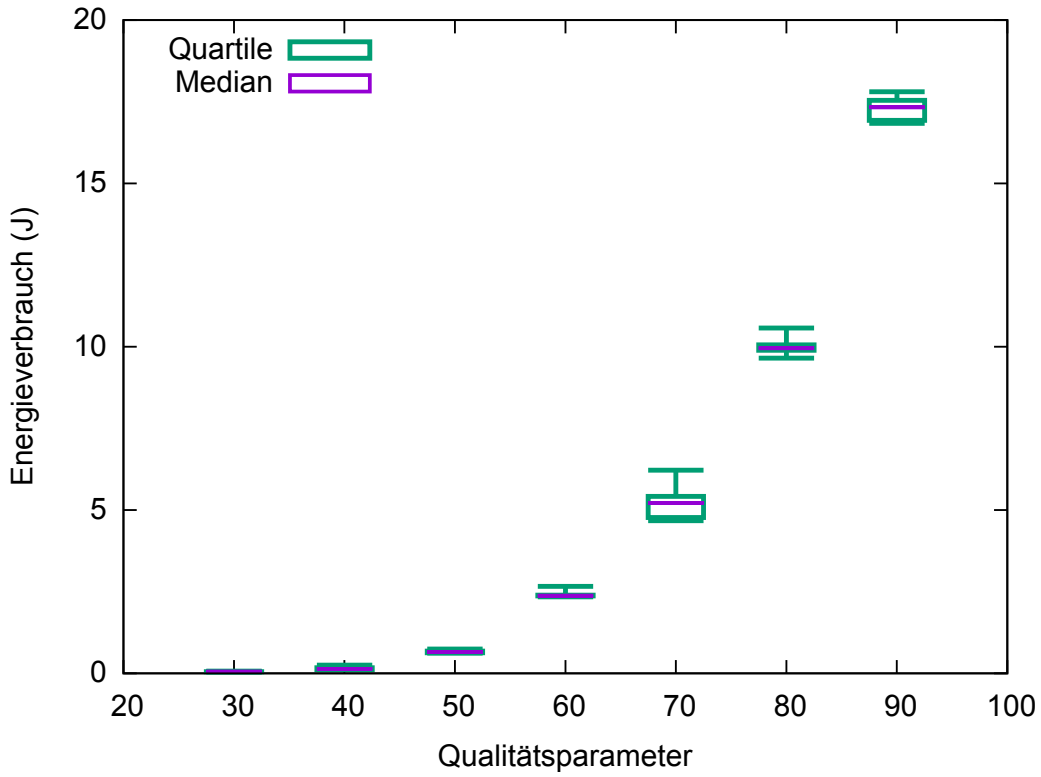

Abbildung 4.4: Streuung der Laufzeiten des Diffusion-Advection-Algorithmus bei verschiedenen Diskretisierungsgraden

Qualitätsparameter nur eine einzige Messung durchgeführt wurde, welche einem gewissen Maß an Streuung unterliegt.

Abbildung 4.4 zeigt eben jene Streuung, oder Varianz, der Laufzeit bei mehrmaliger Ausführung derselben Berechnung. Es wurden für mehrere Diskretisierungsgrade jeweils zwölf Messungen durchgeführt und statistisch ausgewertet. Die *Whiskers* (äußeren Spitzen) geben das nullte beziehungsweise vierte Quartil an; die Boxen den Bereich zwischen dem ersten und dritten Quartil; und die schwarzen Linien den Median (2. Quartil). Die maximale prozentuale Abweichung vom Mittelwert beträgt (für $n = 40$) $\pm 19,1\%$; die minimale Abweichung (für $n = 90$) $\pm 2,9\%$. Bei der späteren Vorhersage durch das Energiemodell sollten also Streuungen einer entsprechenden Größenordnung berücksichtigt werden – sowohl bei der Aufstellung des Modells anhand von Messungen als auch bei der späteren Auswertung, sofern vereinzelte Überschreitungen einer Energieschranke bereits ins Gewicht fallen.

Abbildung 4.5 zeigt einzelne Datenpunkte, deren Laufzeit ihr Energieverbrauch zugeordnet wird. Die sieben erkennbaren Häufungen entsprechen dabei jeweils einer Reihe von Messungen mit Parametern 30, 40, ..., 90. Die dazu berechnete Regressionsgerade

$$W(t) = 2,3347t - 0,258239 \tag{4.1}$$

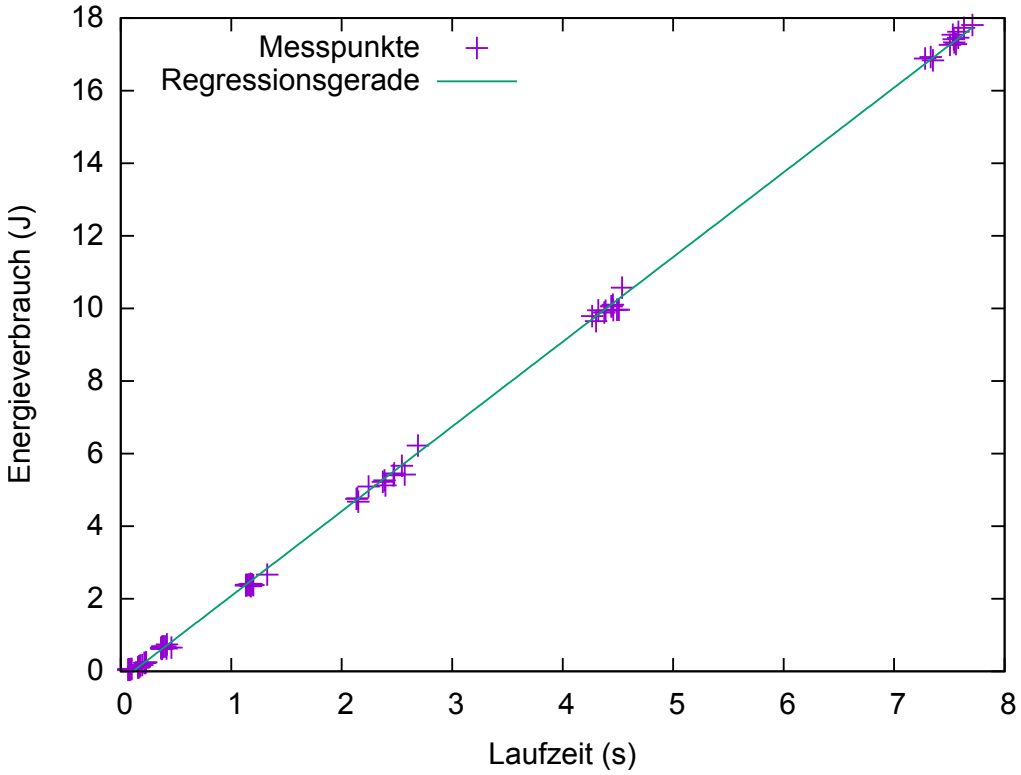

Abbildung 4.5: Laufzeit-Energie-Zuordnung von Messungen mit Regressionsgerade

nähert den Zusammenhang von Laufzeit und Energie akkurat an, wie aus der Visualisierung sowie aus den geringen asymptotischen relativen Standardabweichungen $\sigma_m \approx 0,2\%$ und $\sigma_c \approx 6,5\%$ ersichtlich wird. Der absolute Versatz von $c \approx -0,26$ ist in Bezug auf die absoluten Größen recht gering, wobei es sich bei diesem auch um eine streuungsbedingte Ungenauigkeit handeln kann.

Die erworbenen Erkenntnisse über den Energieverbrauch des Mobilgeräts bei der Berechnung von Simulationen werden im folgenden Entwurf genutzt, um ein akkurates Vorhersagemodell aufzustellen.

5 Entwurf

Im Folgenden wird ausgehend von den Erkenntnissen der Voruntersuchung eine Methodik zur Berechnung des geforderten Vorhersagemodells erarbeitet.

5.1 Auswertung der Voruntersuchung

Aus den vorliegenden Messungen aus Kapitel 4 kann auf einen proportionalen Zusammenhang von Laufzeit und Energieverbrauch der Berechnung geschlossen werden.

So kann in Abbildung 4.2 beobachtet werden, dass die Leistungsaufnahme dieser Berechnung näherungsweise ein Plateau konstanter Höhe darstellt (es variiert also nur die Laufzeit, nicht aber die Leistungsaufnahme, abhängig von der Qualität). Für konstante Leistungsaufnahmen gilt $W = Pt$, woraus ein proportionaler Zusammenhang zwischen Zeit- und Energieverbrauch (mit Faktor P) geschlossen werden kann.

Weiter unterstützt wird dies durch den gemeinsamen Anstieg des Energieverbrauchs und des Zeitaufwands (Abb. 4.3) sowie durch die Zuordnung von Laufzeiten zu ihrem entsprechenden Energieverbrauch; wie in Abbildung 4.5 zu sehen ergibt sich ein Datensatz, der gut durch eine Gerade bzw. sogar durch eine Ursprungsgerade angenähert werden kann.

Die Erkenntnis eines proportionalen Zusammenhangs zwischen Zeit und Energie für die vorliegende Art von Berechnungen wird im Folgenden für ein initiales mathematisches Modell für den Energieverbrauch verwendet. Zunächst wird eine möglichst einfache Anwendung betrachtet, die nur aus einer isolierten Berechnung variabler Qualität besteht, die keinerlei externe Schnittstellen nutzt.

Der Energieverbrauch $W(q)$ in Abhängigkeit des Qualitätsparameters p (wobei ein höheres p eine höhere Qualität bedeutet) kann damit modelliert werden als

$$W(p) := \alpha T(p), \quad \alpha \in R^+, \tag{5.1}$$

wobei $T(p)$ das Performance-Profil der Laufzeit des Algorithmus bei entsprechender Qualität darstellt. Bei dem Koeffizienten α, welcher die Laufzeit der Berechnung auf den Energieverbrauch skaliert, handelt es sich aufgrund von $W = Pt$ um die mittlere Leistungsaufnahme P während der Berechnung. Der aus Messungen extrapolierte Energieverbrauch bei einer Laufzeit von 0s ist sehr klein (vgl. Gl. 4.1) und muss auch wegen des leeren Zeitintegrals über

die Leistung als 0 angenommen werden. Aus diesen Gründen ist die Annahme von Proportionalität gegenüber bloßer Linearität und damit das Fehlen eines konstanten Summanden c in Gleichung 5.1 gerechtfertigt.

Aus diesem ersten Modell folgt, dass bei Kenntnis des Performance-Profils $T(p)$ und des entsprechenden konstanten Faktors α direkt der erwartete Energieverbrauch berechnet werden kann.

Zeitmessungen von Algorithmen sind sehr unkompliziert, exakt und ohne weitere Hilfsmittel durchführbar, während Energiemessungen von Mobilgeräten im Allgemeinen aufwendiger sind, sofern ein einigermaßen genaues Ergebnis erwünscht ist (vgl. Aufbau zur Energiemessung, Kapitel 4.2). Wird ein Vorhersagemodell über das Performance-Profil der Laufzeit ermittelt und im Anschluss mithilfe weniger Energiemessungen zum Performance-Profil des Energieverbrauchs umgerechnet, so können gegenüber eines direkten Profilings des Energieverbrauchs viele kostenintensive Energiemessungen vermieden werden.

Im Folgenden wird also ein Verfahren für die Ermittlung des Performance-Profils der Laufzeit eines Algorithmus vorgestellt, sodass von diesem aus unmittelbar auf den Energieverbrauch geschlossen werden kann. Die Erarbeitung dieses Verfahrens wird in zwei Teilen geschehen; zunächst durch ein initiales Energiemodell beruhend auf der Annahme von Berechnungen mit konstanter Leistungsaufnahme und anschließend durch ein erweitertes Modell, das auf das initiale Verfahren aufbaut und es um weitere Anwendungsfälle erweitert.

5.2 Initiales Energiemodell

Zunächst soll ein einfaches Modell entwickelt werden, das Vorhersagen für den Energieverbrauch von solchen Berechnungen treffen kann wie sie in der Voruntersuchung auftraten.

5.2.1 Überblick

Ausgehend von einem qualitätsbewussten Algorithmus auf mobilem Endgerät wird ein Energiemodell aufgestellt, das für die Vorhersage und Optimierung des Energieverbrauchs verwendet werden kann. Es folgt ein Überblick über die durchzuführenden Schritte gemäß Abbildung 5.1; im Anschluss daran werden diese Schritte im Einzelnen behandelt.

1. *Ermittlung des Performance-Profils der Laufzeit.* Es werden für verschiedene Qualitätsparameter p_i (z.B. äquidistant gewählt) jeweils vier Zeitmessungen der Berechnung vorgenommen. Aus der zweithöchsten gemessenen Laufzeit folgt ein Datenpunkt (p_i, t_i). Mit der Gesamtheit dieser Datenpunkte wird anschließend eine stetige Funktion $T(p)$ entweder interpoliert (z.B. lineare Splines) oder durch Regression angenähert (z.B. Least-Squares mit ansteigendem Polynomgrad unter Abbruchbedingung). Diese Funktion ist ein Performance-Profil; sie gibt die qualitätsabhängige Laufzeit der Berechnung an.

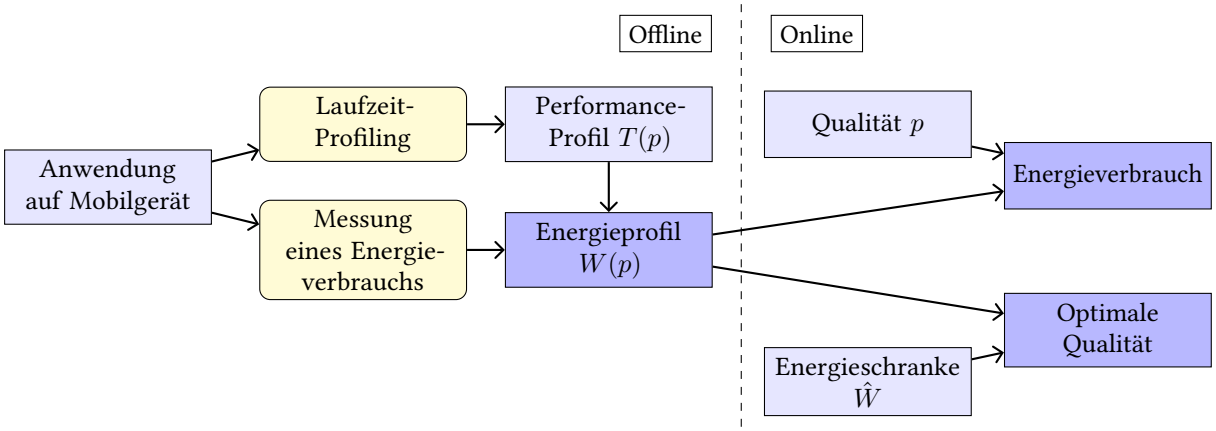

Abbildung 5.1: Workflow zur Berechnung und Anwendung des initialen Energiemodells

2. *Übertragung auf Energieverbrauch.* Der Koeffizient α, welcher multipliziert mit $T(p)$ das Energieprofil $W(p)$ ergibt (Gl. 5.1), wird ermittelt. Daher muss zumindest der Energieverbrauch $W(\hat{p})$ zu einem einzelnen Parameter $\hat{p}$ gemessen werden. Hier ist es aufgrund der Streuung wiederum sinnvoll, beispielsweise vier Werte zu messen und den zweithöchsten zu verwenden. Durch

$$\alpha = \frac{W(\hat{p})}{T(\hat{p})} \tag{5.2}$$

   kann der Koeffizient berechnet werden, was dann durch Formel 5.1 zur kontinuierlichen Funktion $W(p)$ für den Energieverbrauch abhängig vom Qualitätsparameter führt.

3. *Verwendung* für Vorhersagen und Optimierung. Der Energieverbrauch der Berechnung zu einem bestimmten Parameter ergibt sich unmittelbar durch Einsetzen in $W(p)$. Der korrekte Berechnungsparameter für eine gegebene Energieschranke $\hat{w}$ ergibt sich durch Nullstellensuche von $W(p) - \hat{w} = 0$.

Die genaue Herleitung der einzelnen Schritte wird im Folgenden erläutert.

5.2.2 Profiling der Laufzeit

Durch leicht durchführbare Laufzeitmessungen einer Berechnung können Datenpunkte

$$\{(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n) \mid n \in \mathbb{N}\}$$

erhoben werden, welche die Laufzeiten einer Berechnung in Abhängigkeit des Qualitätsparameters darstellen. Aus diesen Daten soll nun ein kontinuierliches Performance-Profil der Laufzeit, $T(p)$, ermittelt werden. Dazu können lineare Regressionen verschiedener Art verwendet werden, wie im Folgenden beschrieben.

Ein einfacher Ansatz ist mit nur zwei verschiedenen Qualitätsparametern je eine Berechnung durchzuführen. Durch die Messpunkte $\{(p_1, t_1), (p_2, t_2)\}$ lässt sich eine Gerade

$$g(p) = \frac{p - p_1}{p_2 - p_1} t_2 + \frac{p_2 - p}{p_2 - p_1} t_1 \tag{5.3}$$

legen, die eine erste Näherung der qualitätsabhängigen Laufzeit darstellt. Jedoch sind qualitätsabhängige Laufzeiten der meisten numerischen Simulationen kaum linear. Wird etwa eine Erhöhung der Qualität durch eine feinere Diskretisierung vorgenommen, so ist die Laufzeit bei jedem zwei- und höherdimensionalen Simulationsbereich bereits mindestens quadratisch. Stattdessen muss das Laufzeitprofil mit anderen Kurven angenähert werden – bei polynomieller Laufzeit etwa durch eine Parabel n-ter Ordnung.

Eine mögliche Herangehensweise ist mit einer linearen Regression zu beginnen und dann den Grad des zugrundeliegenden Polynoms schrittweise zu erhöhen, bis der durch einen weiteren Schritt gewonnene Zuwachs an Genauigkeit vernachlässigbar ist. Da die Funktion zur Approximation der Laufzeit nur im Ersten Quadranten zulässig ist und von den zugrundeliegenden Datenpunkten eine weitgehend monoton steigende Laufzeit abhängig von der Qualität erwartet wird (vgl. Gl. 3.1), wird diese Regression mit ansteigendem Polynom-Grad im betrachteten Intervall in akzeptabler Laufzeit zu einem hinreichend genauen Ergebnis führen.

Ein Beispiel: Zur Regression der Datenpunkte wird eine Parabel dritter Ordnung aufgestellt. Die Laufzeit in Abhängigkeit der Qualität p, und damit das Performance-Profil, ist somit

$$T(p) = ap^3 + bp^2 + cp + d. \tag{5.4}$$

Die angenäherte Laufzeit des Algorithmus bei einem gegebenem Qualitätsparameter $\hat{p}$ (*Vorhersage*) lässt sich damit unmittelbar durch Auswertung des Polynoms an der Stelle $\hat{p}$ berechnen. Soll hingegen der Parameter $\hat{p}$ ermittelt werden, zu dem eine gegebene Laufzeit $\hat{T}$ eingehalten wird (*Optimierung*), ist die Gleichung

$$a\hat{p}^3 + b\hat{p}^2 + c\hat{p} + d = \hat{T} \tag{5.5}$$

zu lösen. Dieses Problem kann unter anderem als Nullstellensuche der Funktion $a\hat{p}^3 + b\hat{p}^2 + c\hat{p} + d - \hat{T}$ interpretiert und analytisch sowie numerisch gelöst werden. Unter der Voraussetzung, dass die Funktion monoton steigt und nur im Ersten Quadranten relevant ist, ist die Lösung eindeutig. Doch selbst mit der Forderung von monotonen Messpunkten in den Ausgangsdaten, also dass

$$\forall (p_i, t_i), (p_j, t_j) : p_i < p_j \Rightarrow t_i \leq t_j \tag{5.6}$$

gilt, kann durch polynomielle Regression grundsätzlich trotzdem eine nicht-monotone Funktion berechnet werden.[1] Hat in diesem Fall Gleichung 5.5 mehrere Lösungen, so sollte prinzipiell die

[1] Seien beispielsweise Datenpunkte {(1,1), (2,2), (9,3), (10,4)} gegeben (wobei die y-Werte nach x-Werten sortiert sogar *streng* monoton ansteigen), so erhält man durch eine polynomielle Regression vom Grad 3 (welche somit zugleich eine Interpolation darstellt) eine Funktion, die innerhalb dieses Intervalls einen Hoch-, einen Tief- sowie einen Wendepunkt besitzt. Das bekannte Problem der „Oszillierung“ um wenige Datenpunkte herum bei Regressionen höheren Grades führt zu derartigen Effekten.

Lösung mit höherer Qualität verwendet werden, da bei gleichem Energieresultat eine bessere Qualität der niedrigen vorzuziehen ist.

Falls die durch Regression berechnete Laufzeitfunktion deutliche Abstiege aufweisen sollte, die nicht mehr durch übliche Streuung erklärt werden können, bedeutet dies, dass die gemessene Berechnung ein Intervall enthält, in welchem trotz steigenden Aufwands (in Form der Qualität) die Laufzeit absinkt. Die Berechnung benötigt hier also bei zunehmender Qualität weniger Ressourcen. Dies entspricht nicht den Anforderungen an einen qualitätsbewussten Algorithmus (Gleichung 3.1), weshalb in diesem Fall der zu berechnende Algorithmus einer Überprüfung unterzogen werden sollte.

5.2.3 Ungenauigkeit und Streuung

Die Laufzeit bei einem gegebenen Qualitätsparameter p ist nicht exakt, sondern variiert je nach Ausführung. Da es sich bei den untersuchten Mobilgeräten um eine vielschichtige Hardware- und Softwarearchitektur handelt, können diese Abweichungen nicht mit akzeptablem Aufwand ergründet werden, sondern müssen vielmehr als nichtdeterministische Streuung hingenommen werden.

Eine Lösung für diese Ungenauigkeiten der Messdaten bietet die Einteilung der Datenpunkte in Quantile. Es lässt sich beispielsweise das Quantil $Q_{0,90}$ identifizieren, in welchem sich erwartet 9 von 10 Ausführungen befinden werden. Dies gewährt eine Orientierung für die Wahrscheinlichkeit, dass eine Ausführung des Algorithmus mit Parameter p die Ausführungsdauer $Q_{0,90}$ nicht überschritten wird. Wird dieses Quantil für die Abfragen von Laufzeit und Qualität verwendet (anstelle des Durchschnitts), ist mit hoher Wahrscheinlichkeit von einer Einhaltung der Zeitschranke auszugehen – vor allem dann, wenn der Algorithmus mehrmals ausgeführt wird und die Summe der Laufzeiten den Ausschlag gibt.

Die Messungen zur Streuung der Laufzeiten (siehe Abbildung 4.4) zeigen, dass konkret etwa das dritte Quartil jeder Messung gewählt werden kann, um auf der einen Seite den Energiebedarf nach Möglichkeit nicht zu unterschätzen und auf der anderen Seite bei mehrmaliger Ausführung mit hoher Wahrscheinlichkeit davon ausgehen zu können, dass die Schranke insgesamt eingehalten wird.

5.2.4 Zusammenfassung

Das entstandene Energiemodell beruht auf dem proportionalen Zusammenhang von Zeit und Energie. Trotz der Berechnung des Performance-Profils über die Laufzeit müssen einige wenige Energiemessungen durchgeführt werden.

Unter der Annahme, dass der relative Anstieg des Berechnungsaufwands über verschiedene Geräte hinweg derselbe ist (dabei können weitere Einschränkungen wie etwa auf eine spezielle Version des Betriebssystems sinnvoll sein), könnte der Parameter α als ein geräteabhängiger

„Eichungs“-Faktor angesehen werden, während die Laufzeiten nur für ein einzelnes Gerät ermittelt werden müssen (die absoluten Unterschiede der Laufzeiten bei verschiedenen Geräten würden dann zusammen mit dem Zeit-Energie-Koeffizienten verrechnet). Hierfür wären jedoch weitere Untersuchungen notwendig.

5.3 Mehrstufige Berechnungen

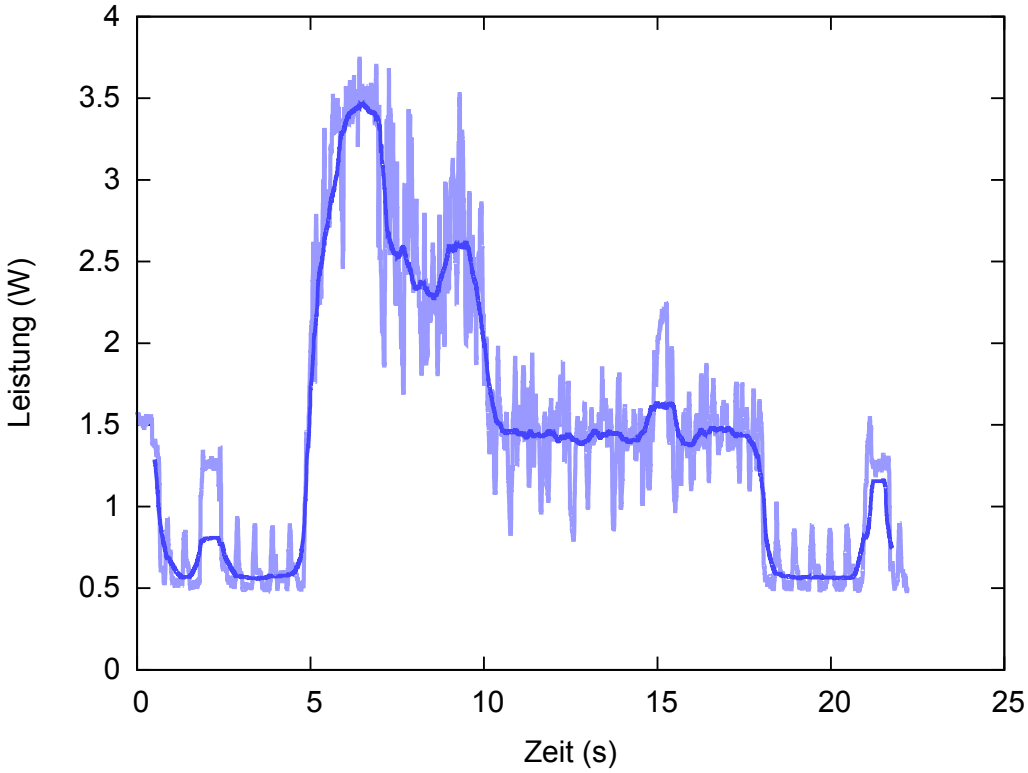

Abbildung 5.2: Leistungsaufnahme einer Berechnung zur approximativen Lösung des Rucksack-Problems

Die bisher betrachtete Berechnung ist eine einfache; sie wird vollständig im Arbeitsspeicher ausgeführt, verwendet keine energieaufwendigen Features wie GPS oder Mobildaten und macht keinen Gebrauch von System-APIs. Bei komplexeren Berechnungen müssen allerdings auch Szenarien in Betracht gezogen werden, bei denen etwa das Gerät periodisch Daten auf die SD-Karte schreibt und/oder davon ausliest; zum Beispiel bei Operationen auf sehr großen Datenmengen.

Aber auch bereits eine einfache Berechnung, welche verschiedene Stadien mit unterschiedlichem Leistungsanspruch durchläuft, kann durch das bisherige Modell nicht mehr vollständig

erfasst werden. In Abbildung 5.2 ist die Leistungsaufnahme eines Algorithmus zur approximativen Lösung des Rucksack-Problems abgebildet. Die erste Phase des Algorithmus, in dem alle eingegebenen Gegenstände analysiert und manipuliert werden, ist in Form von Java-Objekten und der Manipulation von deren Attributen und Methoden implementiert. Die Zeitdauer dieser Phase hat eine lineare Komplexität in Abhängigkeit der Gegenstände. Die zweite, lange Berechnungsphase hingegen ist mit speichereffizienten Arrays implementiert, die nur ein einziges Mal allokiert und danach geschickt manipuliert werden. Diese Phase hat eine polynomielle bzw. exponentielle Laufzeit. Es ist klar zu sehen, dass die Leistungsaufnahme während der Ausführung der „High-Level“ Java-Implementierung deutlich höher ist als während der speichereffizienten Berechnung.

Bei Berechnungen dieser Art kann also nicht mehr davon ausgegangen werden, dass die Leistungsaufnahme des Geräts in Abhängigkeit der Zeit ein näherungsweise ebenes Plateau darstellt (vgl. Abb. 4.2); stattdessen wird die Berechnung verschiedene „Stationen“ durchlaufen, deren Zeitdauern abhängig von der Qualität möglicherweise verschiedenartig skalieren. Es besteht die Notwendigkeit einer Modellerweiterung, die derartige Berechnungen mit mehreren Phasen berücksichtigen kann. Dazu wird im Folgenden eine geeignete Abstraktion in Form von Berechnungspfaden eingeführt.

5.3.1 Energiezustände und deren Transitionen

Ein *Berechnungspfad* B bestehe aus einer geordneten Menge von *Zuständen* z_i, die jeweils für eine bestimmte Phase der Berechnung stehen. Für eine solche Phase soll im Sinne der bisherigen Untersuchungen gelten, dass die Leistungsaufnahme konstant ist und daher der Energieverbrauch der Phase gut durch deren Laufzeit multipliziert mit der entsprechenden Leistungsaufnahme angenähert werden kann. Eine Phase wird also definiert durch ihre Leistungsaufnahme und ein Performance-Profil der Laufzeit:

$$z_i = (T_i(p), P_i)$$

Mögliche Phasen wären hier beispielsweise

- Das Einlesen von Daten (hohes Energieniveau, da der interne Speicher oder die SD-Karte aktiv genutzt wird);
- Berechnungen durch Manipulation von Java-Objekten; oder
- eine niedrig abstrahierte Berechnung, etwa mit integriertem C-Code oder besonders speichereffizientem Java-Code (niedriges Energieniveau, vgl. Abb. 5.2).

Unter Berücksichtigung dieser Eigenschaften kann einem Berechnungspfad

$$B = (z_1, z_2, \ldots z_n)$$

ein Energieverbrauch

$$W(p) = \sum_{i=1}^{n} T_i(p) P_i \tag{5.7}$$

in Abhängigkeit der Qualität p zugeordnet werden. Diese Identität ergibt sich aus dem Zusammenhang von Leistung und Energie ($W = \int_{t_1}^{t_2} P(t)dt$) unter einer geeigneten Partitionierung des Integrals, wobei die Leistung $P_i(t)$ jeder Partition i gemäß Gleichung 5.1 (aus dem initialen Modell) angenähert wird.

Durch dieses Modell können die Ergebnisse des initialen Energiemodells auf allgemeinere Fälle übertragen werden, etwa den mehrfachen Aufruf einer Berechnung nach einem einmaligen Preprocessing (*Multi Queries*) oder vorhersehbare nebenläufige Berechnungen als Teil einer ansonsten sequentiellen Berechnung. Für das zuvor genannte Beispiel der Implementierung des Rucksack-Problems (vgl. Abb. 5.2) kann der dazugehörige Berechnungspfad aussehen wie in Abbildung 5.3 dargestellt.

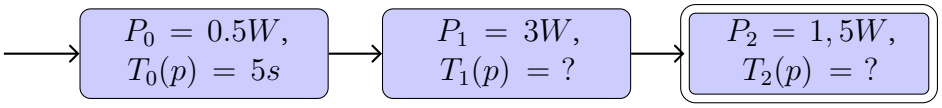

Abbildung 5.3: Beispielhafte Modellierung der Berechnung zur Lösung des Rucksack-Problems durch Energiephasen

Bei der Berechnung der verbrauchten Gesamtenergie ist die Reihenfolge der Phasen irrelevant; es können beispielsweise alle Phasen desselben Energieniveaus zusammengefasst werden, indem die Laufzeitfunktionen aufeinander addiert und dann mit der gemeinsamen Leistung P_i multipliziert werden. Interessant für die Ergebnisse ist somit nur noch, welche Energieniveaus während der Berechnung auftreten und für welche gesamte Zeitdauer die Niveaus jeweils bestehen.

5.3.2 Ermittlung der Phasen

In Anlehnung an das bisherige Verfahren muss nach wie vor die Laufzeit der Berechnung ermittelt werden; da es sich um verschiedene Phasen handelt, müssen für die einzelnen Phasen individuelle Laufzeiten $T_i(p)$ ermittelt werden. Mögliche Ansätze zur Identifikation der Phasen umfassen eine empirische (Black-Box-)Analyse der Leistungsaufnahme, die statische Analyse des Programmcodes (Glass-/White-Box) oder eine Kombination der beiden Techniken.

Analyse der Leistungsaufnahme

Wie zuvor kann für die Berechnung unter verschiedenen Qualitätsparametern die Leistungsaufnahme im Verlauf der Zeit gemessen werden. Dabei ergibt sich angenähert eine Sequenz

verschiedener Plateaus, welche den Phasen der Berechnung entsprechen. Der gesamte Energieverbrauch lässt sich dann berechnen durch Addition des Energieverbrauchs der Phasen, der jeweils durch Integration über die Leistungsaufnahme ermittelt werden kann (vgl. Gleichung 5.7). Für eine aussagekräftige Visualisierung der Plateaus sollten hierbei jedoch mehrere Ausführungen der Berechnung gemessen und dargestellt werden, um Irrtümer aufgrund von Streuung zu vermeiden. Die Kenntnis des Programmcodes ist bei dieser Analysetechnik nicht notwendig.

Code-Analyse

Unter der Voraussetzung, dass für die verschiedenen Operationen des betrachteten Programmcodes bekannt ist, welchem Energieniveau sie näherungsweise zuzuordnen sind, kann für den Code im Voraus approximiert werden, welche Teile jeweils welche Leistungsaufnahme aufweisen werden. Dies geschieht im Rahmen verschiedener Forschungsarbeiten auf Ebene einzelner Instruktionen etwa mithilfe spezieller System-Profile [HLHG13] oder durch Analyse des kompilierten Bytecodes unter Nutzung von CPU-Profilen [HLHG12].
Im hier vorliegenden Entwurf wird das Vorhandensein derartiger Profile jedoch nicht vorausgesetzt; stattdessen wird mithilfe einfacher Mittel der Programmcode in grobgranulare Teile gegliedert, wie im Folgenden beschrieben.

In vielen Berechnungen können Phasen identifiziert werden, die unterschiedlichen Leistungsaufnahmen zuzuordnen sind. Das vorherige Beispiel zur Lösung des Rucksack-Problems etwa zeigt, dass die konsequente Nutzung von Java-Objekten und deren Manipulation zu einem gewissen, höheren Energieniveau führt als die speichereffiziente Verwaltung von Arrays. Innerhalb der Berechnung führte dieser Unterschied zu zwei verschiedenen Energieniveaus. Ähnlich interessant für den Energieverbrauch ist der Einsatz von System-API-Aufrufen gegenüber einer Berechnung ohne derartige Schnittstellen. [LHGH14, S.5] suggeriert, dass bei 75% der untersuchten Apps der Anteil von API-Aufrufen am gesamten Energieverbrauch bei über 82% liegt.

Sofern die angesprochenen „interessanten“ Operationen über verschiedene Teile der Berechnung verteilt sind, kann an den Übergangsstellen dieser Teile ein Übergang der energetischen Phase vermutet werden. Für derartige Übergangsstellen wird der Begriff der *Indikatoren* (für Energiezustandsübergänge) eingeführt.

Für sich allein ist ein Indikator weder hinreichend noch notwendig für einen tatsächlichen Energiezustandsübergang; doch sobald es gelingt, sämtliche für den Code relevanten Indikatoren zu sammeln, sind diese in ihrer Gesamtheit ein *notwendiges* Kriterium. Das bedeutet: liegt für eine Stelle im Code *kein* Indikator vor, so kann ausgeschlossen werden, dass an dieser Stelle ein Energiezustandsübergang auftritt. Alle Stellen *mit* einem oder mehreren Indikatoren müssen hingegen weiteren Prüfungen unterzogen werden, ob tatsächlich ein signifikanter Energiezustandsübergang auftritt. Die Teile des Programms, die durch zwei Indikatoren eingeschlossen

werden, werden zunächst als *vermutete Phasen* bezeichnet, bis ein tatsächlich unterschiedliches Energieniveau erwiesen oder widerlegt ist.

Bei Berechnungen, welche inhärent in semantische Abschnitte unterteilt werden können, bietet es sich an zunächst die Übergänge dieser Phasen im Code näherer Beobachtung zu unterziehen, ob hier ein Indikator zutrifft.

Kombination der beiden Ansätze

Die beiden Ansätze zur Identifikation der Plateaus – eine Betrachtung des Codes und Ermittlung unterschiedlicher Energieaufwände sowie die Messung und Einteilung der Leistungsaufnahme bei der Ausführung – können folgendermaßen kombiniert werden:

- Die möglichen n Stellen des Codes, welche Indikatoren für die Veränderung des Energieniveaus enthalten (identifizierbar etwa durch das Auftreten von API-Aufrufen, parallelen Berechnungen oder die Nutzung unterschiedlicher Features der Programmiersprache), werden mit „Zeitmarkierungen“ $t_i, 0 \leq i \leq n$ versehen (in Java durch `long time = System.nanoTime();`); diese Zeiten werden im Anschluss an die Berechnung zusammen mit einem eindeutigen Bezeichner geloggt.
- Die Berechnung wird unter verschiedenen Qualitätsparametern ausgeführt und dabei die Leistungsaufnahme gemessen.
- Die Leistungsaufnahme wird in der Visualisierung mit den Zeitmarkierungen verbunden (Zeitmarkierungen können dabei als vertikale Linien dargestellt werden); zugleich kann automatisch für jede vermutete Phase i die durchschnittliche Leistungsaufnahme

$$\tilde{P}_i = \frac{1}{t_{i+1} - t_i} \int_{t_i}^{t_{i+1}} P_i \, dt \tag{5.8}$$

  berechnet werden.
- Zuverlässige Aussagen über die energetischen Phasen der Berechnung lassen sich durch signifikante Unterschiede zwischen benachbarten $\tilde{P}_i$ treffen. Durch menschliche Interaktion können alternativ auch signifikante Plateaus zwischen den Zeitmarkierungen in der Visualisierung identifiziert werden.

Unter der Voraussetzung, dass die angesetzten Indikatoren *vollständig* sind, also bei jedem Energiezustandsübergang mindestens einer der Indikatoren auftritt, werden durch die genannte Methodik alle Energiezustände gefunden.

5.3.3 Ermittlung der Laufzeiten und Koeffizienten

Wurden die energetischen Phasen der Berechnung identifiziert, so besteht der nächste Schritt darin, für diese Phasen jeweils – gemäß des initialen Modells – ein Laufzeit-Profil $T_i(p)$ sowie eine Leistung P_i zu identifizieren, um daraus entsprechende Energie-Profile $W_i(p)$ zu berechnen.

Die Laufzeiten $T_i(p)$ der Phasen ergeben sich durch

$$T_i = t_{i+1} - t_i \tag{5.9}$$

für jeden in der Messung berücksichtigten Qualitätsparameter. Wurden mehrere vermutete Phasen mangels eines signifikanten Leistungsunterschieds zu einer einzelnen Phase zusammengeführt, so ergibt sich die Laufzeit analog durch Addition der Teillaufzeiten. Wie im initialen Ansatz führt für Phase i dann eine Interpolation oder Regression zu einer Laufzeitfunktion $T_i(p)$.

Auch für die – pro Phase als konstant angenommenen – Leistungsaufnahmen P_i, welche dem Koeffizienten α aus dem initialen Ansatz entsprechen, werden direkt die gemittelten $\tilde{P}_i$ verwendet.

Damit sind alle benötigten Werte vorhanden.

5.3.4 Verwendung des Modells

Nach Bestimmung aller benötigten Funktionen und Werte ($T_i(p)$ und P_i, $0 \leq i \leq n$) kann für einen gegebenen Qualitätsparameter p gemäß Formel 5.7 der Energieverbrauch bei diesem Parameter bestimmt werden.

Um den optimalen Parameter p für eine gegebene Energieschranke $\hat{W}$ zu ermitteln muss die Gleichung

$$\sum_{i=1}^{n} T_i(p) P_i - \hat{W} = 0 \tag{5.10}$$

nach p aufgelöst werden. Da die einzelnen Laufzeiten Polynome darstellen (sofern eine polynomielle Regression zur Bestimmung der Laufzeiten der einzelnen Phasen verwendet wurde), kann die Gleichung, wie im initialen Ansatz, durch eine einfache Nullstellensuche numerisch sowie analytisch bestimmt werden.

5.3.5 Zusammenfassung

Das erweiterte Modell kann verwendet werden, um den Energieverbrauch von Berechnungen zu modellieren, die keine homogene Leistungsaufnahme aufweisen. Durch die folgenden Schritte lässt sich das Modell errechnen und nutzen:

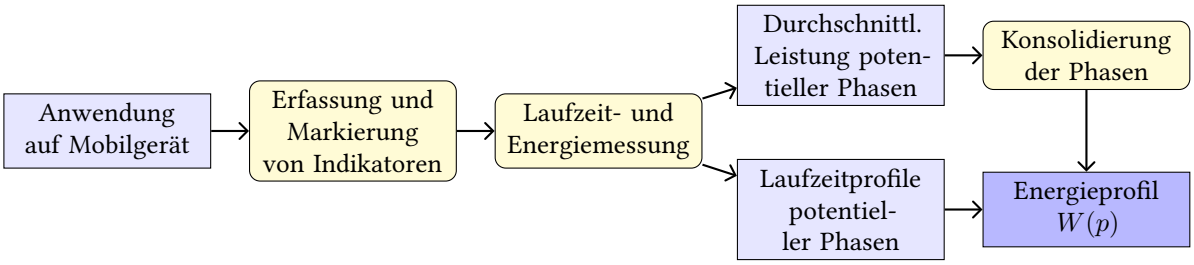

Abbildung 5.4: Workflow zur Berechnung des erweiterten Energiemodells; die Anwendung verläuft identisch zum initialen Modell, vgl. Abb. 5.1

1. *Ermittlung der Energiephasen.* Der Code wird an allen Stellen, die *Indikatoren* für einen Energiezustandsübergang aufweisen, mit Zeitmarkierungen versehen. Energiemessungen mit Zeitnahme werden anschließend für verschiedene Qualitätsparameter durchgeführt. Für jede vermutete Phase i und jeden Qualitätsparameter p wird die durchschnittliche Leistungsaufnahme P_i (Gl. 5.8) sowie die Laufzeit $T_{i,p}$ (Gl. 5.9) berechnet. Die tatsächlichen Phasen ergeben sich aus dem Vergleich benachbarter P_i; hier können Metriken angesetzt werden, die beispielsweise ab einem Unterschied von 20% einen Phasenübergang markieren.
2. *Ermittlung des Performance-Profils der Laufzeit jeder Phase.* Die bereits durchgeführten Messungen werden verwendet, um gemäß des ursprünglichen Modells Performance-Profile $T_i(p)$ zu berechnen.
3. *Übertragung auf Energieverbrauch.* Das Energie-Performance-Profil $W(p)$ ergibt sich durch Gleichung 5.7 aus den bereits berechneten Werten und Funktionen.
4. *Verwendung.* Analog zum ursprünglichen Modell kann der Energieverbrauch zu einem bestimmten Parameter durch Einsetzen in $W(p)$ berechnet werden und der optimale Berechnungsparameter für eine gegebene Energieschranke $\hat{w}$ durch Nullstellensuche von $W(p) - \hat{w} = 0$ bestimmt werden.

Den Abschluss des Entwurfs bildet nun ein kurzer Blick auf die benötigte Anzahl der Energiemessungen für das erweiterte Modell.

Das vorliegende Konzept für ein phasenbasiertes Energiemodell benötigt in seiner Ausgangsform einige Energiemessungen, da für jede der vermuteten Phasen die durchschnittliche Leistungsaufnahme errechnet werden muss. Wird hier die Durchschnittsleistung über alle abgedeckten Qualitätsparameter hinweg gefordert, so erhöht sich die Anzahl der erforderlichen Energiemessungen gegenüber des initialen Modells stark. Um allerdings eine möglichst geringe Anzahl an Energiemessungen zu erreichen, kann in Betracht gezogen werden, wiederum nur für einen einzigen Qualitätsparameter Energiemessungen durchzuführen und für die restlichen Parameter ausschließlich die Laufzeiten zu messen. Damit ist es nach wie vor möglich, durch die durchschnittlichen Leistungsaufnahmen der vermuteten Phasen aus den einzelnen Messungen die tatsächlichen Phasen zu bestimmen und deren Laufzeiten wie bisher zu berechnen.

Jedoch sollten in jedem Fall für die einzelne Qualität mehrere Messungen durchgeführt werden, da die Varianz bezüglich Laufzeit und Energieverbrauch ansonsten zu erheblichen Verfälschungen führen kann (vgl. Abb. 4.4).

Die erarbeitete Methodik wird in Kapitel 7 auf verschiedene Aspekte, darunter auf die Eignung für verschiedene Anwendungsfälle und auf die Genauigkeit der Vorhersagen, geprüft. Auch der nötige Umfang von Energiemessungen wird dabei behandelt.

6 Implementierung

In diesem Kapitel wird die Umsetzung der Voruntersuchung und der Evaluation auf technischer Ebene beschrieben.

6.1 Qualitätsbewusste Algorithmen

Es wurden zur Analyse und Messung des Energieverbrauchs verschiedene qualitätsbewusste Algorithmen verwendet, deren Implementierung im Folgenden näher beschrieben wird.

6.1.1 Diffusion-Advection

Das Diffusion-Advection-Problem beschreibt die Bewegung von Teilchen (beispielsweise Sandkörner oder Staub) oder Zuständen (wie Energie oder Temperatur) durch Diffusion und Fluss (*Advektion*). Es handelt sich um eine Problemstellung aus der statistischen Physik. Da sich das Problem durch eine partielle Differentialgleichung beschreiben lässt, handelt es sich bei dem zur Lösung verwendeten Verfahren um die Grundlage für viele numerische Simulationen. Daher stellt das Diffusion-Advection-Problem ein gut geeignetes Beispiel für Simulationsberechnungen dar.

Zur Lösung des Diffusion-Advection-Problems lässt sich ein konzeptionell einfaches Lösungsverfahren für lineare Gleichungssysteme der Form $Ax = b$ verwenden, sofern die Problemmatrix A und die rechte Seite b entsprechend aufgestellt werden. Da es sich in der Regel um *dünn besetzte* Strukturen handelt, lohnt es sich auch entsprechende Lösungsstrategien für dünne Matrizen zu verwenden. In der Implementierung wurde eine `SparseLU`-Zerlegung verwendet, eine Spezialisierung der bekannten LU-Zerlegung auf dünn besetzte Matrizen.

Das Lösungsverfahren für das Diffusion-Advection-Problem wurde nicht nur in der Android-App-Standardsprache Java, sondern auch mittels des *Native Development Kit* (NDK) in C++ implementiert. Bei der NDK-Implementierung wird zur Laufzeit der App ein zuvor abgelegtes shared-object (.so-Datei) angesprochen und dann ausgehend vom Java-Code über eine `native` Methode ausgeführt. Für die Lösung der Linearen Algebra wurde die Bibliothek *Eigen* verwendet, wobei zusätzlich Gebrauch von *NEON* (Single Instruction, Multiple Data für ARM-Architekturen) gemacht wurde. Durch einzelne, statistisch nicht weiter ausgewertete Laufzeitvergleiche konnte gegenüber einer naiven Implementierung mit Java und der *Apache*

Commons Math Bibliothek ein Performancezuwachs um mehr als das 280-fache beobachtet werden.

6.1.2 Canny Edge Detection

Die *Canny Edge Detection* (Canny-Kantenerkennung) ist ein Verfahren zur Erkennung und Hervorhebung von Kanten in einem Graustufen-Bild. Die Qualität des Algorithmus lässt sich hierbei durch eine anfängliche Skalierung des Bilds auf eine bestimmte Dimension festlegen, da der Algorithmus bei kleineren Bildern weniger Ressourcen benötigt, jedoch auch, ausgehend vom ursprünglichen Bild, ein ungenaueres Ergebnis erbringt.

Kantenerkennungen sind ein zentrales Werkzeug für Bildverarbeitung und -erkennung, da diese unter den richtigen Parametern komplexe Bilder auf die wesentlichen Teile reduzieren können und damit die Grundlage für viele weitere Algorithmen darstellen [ZDM08]. Damit sind Kantenerkennungen besonders in Bezug auf Augmented-Reality Anwendungen interessant, welche häufig bestimmte Objekte aus dem Sichtbereich extrahieren und erkennen müssen.

Abbildung 6.1: Visualisierung des Vorgehens zur Kantenerkennung. V.l.n.r: ursprüngliches Bild (2,3MB), skaliertes Bild auf 800x600px (0,64MB), skaliertes Bild in Graustufen (0,59MB), invertiertes und verstärktes Ergebnis der Kantenerkennung (0,55MB)

Zur Berechnung wurden Algorithmen des `Catalano`-Frameworks [Cat16] für wissenschaftliche Berechnungen auf Java und Android verwendet. Die Erzeugung und Manipulation von Bildern erfolgt mit Objekten einer eigenen Klasse `FastBitmap`. Auf solche Objekte kann dann eine Vielzahl von Filtern gelegt werden (etwa Graustufen-Konvertierung oder Kantenerkennung), welche *in-place* angewandt werden können. Wie in Abbildung 6.1 zu sehen beläuft sich die implementierte Berechnung auf die folgenden Schritte:

- Einlesen und Skalierung des Bildes auf eine niedrige Qualität (abhängig vom übergebenen Parameter) sowie die Speicherung des skalierten Bilds
- Laden des skalierten Bilds und Überführung in Graustufen (in-place)
- Durchführung der Canny-Kantenerkennung und Speicherung des finalen Bilds

Die Zwischenspeicherungen der Bilder wurden vorgenommen, um Erkenntnisse über die Zwischenergebnisse der Berechnung zu erhalten sowie um die Auswirkungen auf den Energieverbrauch bei Speichervorgängen während der Berechnung zu betrachten. Auch im Sinne der Anwendung ist es durchaus als realistisch anzusehen, dass mitunter die Zwischenergebnisse von Interesse sind und daher ebenso abgespeichert werden müssen.

6.1.3 Rucksack-Problem

Das Rucksack-Problem ist ein weit verbreitetes und NP-vollständiges Problem. Es geht darum, eine Auswahl aus einer vorgegebenen Menge von Gegenständen $G = \{g_1, g_2, \dots g_B\}$ mit jeweiligen ganzzahligen Gewichts- und Werteeigenschaften $g_i = (w_i, v_i)$ zu treffen, sodass der Gesamtwert $V = \sum_{i=1}^{B} v_i$ maximiert wird, jedoch unter der Voraussetzung ein vorgegebenes Maximalgewicht W_{max} der ausgewählten Gegenstände nicht zu überschreiten.

Es existiert ein Algorithmus, der mittels Dynamischer Programmierung eine $B \times W_{max}$-Matrix mit Teillösungen befüllt und stets die optimale Lösung berechnet. Die Laufzeit ist dabei aber pseudo-polynomiell und damit bei binär (oder dekadisch) kodierter Eingabe exponentiell. Ein anderer Ansatz über die stetige Erweiterung und Aktualisierung von Listen möglicher Lösungen führt in seiner Grundform zur gleichen asymptotischen Laufzeit, kann jedoch in einer approximierten Form angewandt werden. Die approximative Form des Algorithmus kann qualitätsbewusst implementiert werden, indem ein ε so gewählt wird, dass die Laufzeit und damit der Energieverbrauch ausreichend gering ist, um die vorhandene Schranke einzuhalten.

Es wurde eine approximative Form des Algorithmus zur Lösung des Rucksack-Problems in Java implementiert und Messungen damit durchgeführt. Akzeptable Effizienz wurde durch die Nutzung von Arrays anstelle von zusammengesetzten Java-Objekten sowie durch die Vermeidung von aufwendigen Umspeicherungen erreicht. Zwei große Arrays werden zu Beginn der Berechnung allokiert und dann nur elementweise überschrieben. Damit entfallen ressourcenintensive periodische Allokationen von neuem Speicher.

Zusätzlich zur sequentiellen Berechnung wurde auch eine parallele Berechnung implementiert, bei welcher ähnlich zu einem *Merge-Sort*-Verfahren die momentanen Lösungen auf Threads aufgeteilt und nach den Teilberechnungen wieder vereinigt wurden. Trotz einiger Optimierungen konnte hier bei nebenläufiger Ausführung kein signifikanter Performance-Gewinn gegenüber der sequentiellen Ausführung beobachtet werden.

Es stellte sich bereits in einem frühen Stadium heraus, dass die Laufzeit sowie der Energieverbrauch kaum merklich mit dem Parameter ε skalieren (wobei insbesondere kein klarer monotoner An- oder Abstieg des Ressourcenverbrauchs in Abhängigkeit von ε zu verzeichnen war) und der Algorithmus damit nicht geeignet für die geplanten Untersuchungen war.

Eine Anwendung fand die Berechnung im Rahmen der Motivation hin zu einem erweiterten Energiemodell (s. Abb. 5.2).

6.2 Entwickelte Applikation

Für die Durchführungen der Messungen – für die Voruntersuchung (Kapitel 4) sowie für die Evaluation (Kapitel 7) – wurde eine App für Android geschrieben, um bestimmte Berechnungen in einem möglichst isolierten Umfeld und mit nachträglichem Zugriff auf die entsprechenden Performance-Daten ausführen zu können. Insbesondere waren die folgenden Anforderungen umzusetzen:

- eine möglichst leichtgewichtige Oberfläche, um den Energieverbrauch so wenig wie möglich zu beeinflussen;
- eine einfache Möglichkeit, wesentliche Parameter der Berechnung anzupassen;
- ein möglichst geringer Performance-Overhead der Berechnung durch die Auswahl geeigneter Bibliotheken und effiziente Ausnutzung des Prozessors;
- die persistente Speicherung der Metadaten von Berechnungen zur späteren Analyse; sowie
- (für die Evaluation) die automatische Ausführung einer Reihe von Berechnungen und unmittelbar darauf die Annäherung eines Performance-Profils.

Die Implementierung der App fand in mehreren Iterationen – jeweils anhand des aktuellen Stands der Arbeit – statt, wodurch die Anforderungen Schritt für Schritt realisiert wurden. Es wurde die Integrierte Entwicklungsumgebung *Android Studio* auf einem Linux-System verwendet.

6.2.1 Entwicklung mit NDK

Mittels des *Android Native Development Kit* (NDK) wurde C++-Code für die numerische Berechnung einer Diffusion-Advection-Instanz in die Java-App integriert. Der C++-Code selbst benutzt wiederum die *Eigen*-Bibliothek für effiziente Lineare Algebra. Diese Bibliothek verwendet ausschließlich Header (.h-Dateien) und keinerlei weitere Bibliotheken oder Binärobjekte, sodass sich die Integration in den NDK-Code einfach gestaltete. Innerhalb des Java-Codes kann durch eine virtuelle Methode mit dem Modifier `native` zur Laufzeit auf die entsprechend benannte C-Methode in den `jni` (Java Native Interface)-Objekten zugegriffen werden.

6.2.2 Oberfläche

Es wurde eine möglichst einfache Oberfläche entwickelt, um den Bereich um die tatsächliche Energiemessung herum möglichst ressourcenschonend zu gestalten.

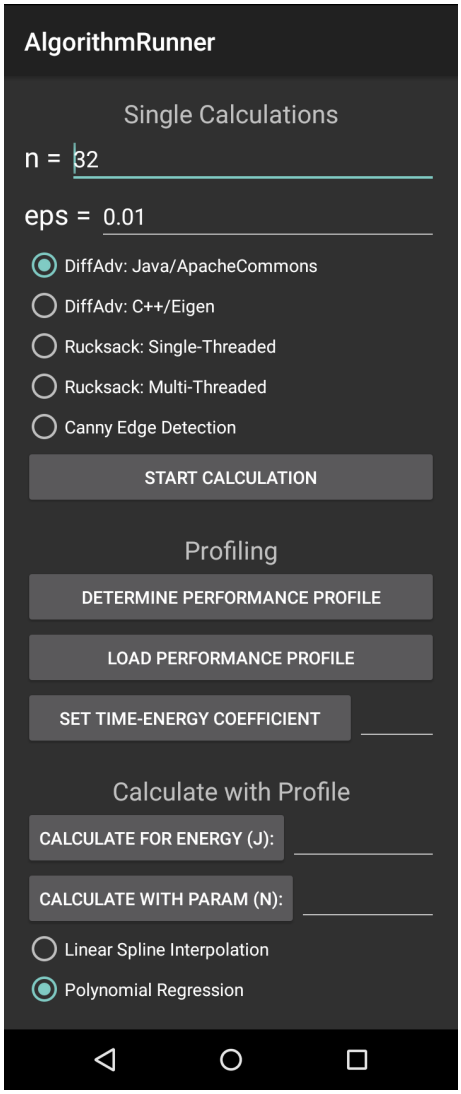

Abbildung 6.2: Oberfläche der für die Messungen genutzten App auf einem Sony Xperia Z1 Compact mit CyanogenMod 12.1[1]

Abbildung 6.2 zeigt die vollständige Oberfläche der App mit den folgenden Gliederungspunkten:

- *Einzelne Berechnungen*: Führt den ausgewählten Algorithmus für die gegebenen Qualitätsparameter einmalig aus und schreibt die Metadaten (Parameter, Uhrzeit, Laufzeiten) in eine CSV-Datei. Mit dieser Option wurden Messungen für die Voruntersuchung sowie für die Evaluation durchgeführt.

[1] Es handelt sich um eine scrollbare Oberfläche, die für diese Grafik aus mehreren Screenshots zusammengesetzt wurde.

- *Profiling*: Das Performance-Profil (bezglich der Laufzeit) des Diffusion-Advection-Algorithmus kann durch eine automatisierte Durchführung mehrerer Berechnungen mit aufsteigender Qualität approximiert und anschließend geladen werden (das Performance-Profil wird in serialisierter Form persistent zwischengespeichert, um mehrfache Nutzung zu ermöglichen). Ebenso kann ein durch separate Messungen ermittelter Zeit-Energie-Koeffizient α gesetzt werden, um für anschließende Berechnungen unter dem Gliederungspunkt *Berechnungen mit dem Profil* verwendet zu werden.
- *Berechnungen mit dem Profil*: Bei geladenem Performance-Profil und vorhandenem α kann nun eine *Vorhersage* oder *Optimierung* berechnet werden. Das Ergebnis wird ausgegeben und kann dann mit tatsächlichen Messwerten verglichen werden.

6.2.3 Einstellungen des mobilen Geräts

Die Energiemessungen des Mobilgeräts wurden im Flugzeugmodus, also ohne Netzwerkverbindung, durchgeführt. Das Gerät wurde unmittelbar vor den Messungen neu gebootet und keinerlei weitere Apps waren während der Messungen aktiv. Der Bildschirm blieb während den Messungen eingeschaltet; die Ausführung der Algorithmen wurde dennoch mit `WakeLocks` implementiert, sodass auch bei ausgeschaltetem Bildschirm die Berechnung korrekt fortgesetzt wird. Nicht weiter ausgewertete Vergleichsmessungen mit ausgeschaltetem Display ähnelten in der Berechnungsphase sehr stark den restlichen Messungen (mit Ausnahme einer geringeren Grundleistungsaufnahme).

6.2.4 Ausführung der Algorithmen

Auf Betätigung des Knopfs *Start Calculation* wird ein `AsyncTask` mit den eingegebenen Parametern gestartet. Dieser führt vor Beginn der eigentlichen Berechnung ein `Thread.sleep(5000);` aus, wartet also fünf Sekunden lang, bis er die Berechnung beginnt. Dadurch lässt sich der Ressourcenverbrauch der grafischen Oberfläche und Nutzereingaben besser vom Ressourcenverbrauch der Berechnung abgrenzen. Nach der Berechnung folgt ein weiterer `sleep` mit drei Sekunden. Anschließend wird eine .csv-Datei geschrieben, die den Zeitpunkt der Berechnung, die Eingabeparameter sowie die exakte Berechnungsdauer enthält. Diese Ausgabe kann dann mit der Ausgabe der Energiemessung des Raspberry Pi verknüpft werden.

Informationen über den Stand der Berechnung wurden mithilfe des Status des entsprechenden Berechnungsknopfes (inaktiv und ausgegraut während der Berechnung) sowie durch *Toasts* (unaufdringliche Popup-Meldungen am Rand des Bildschirms) vermittelt.

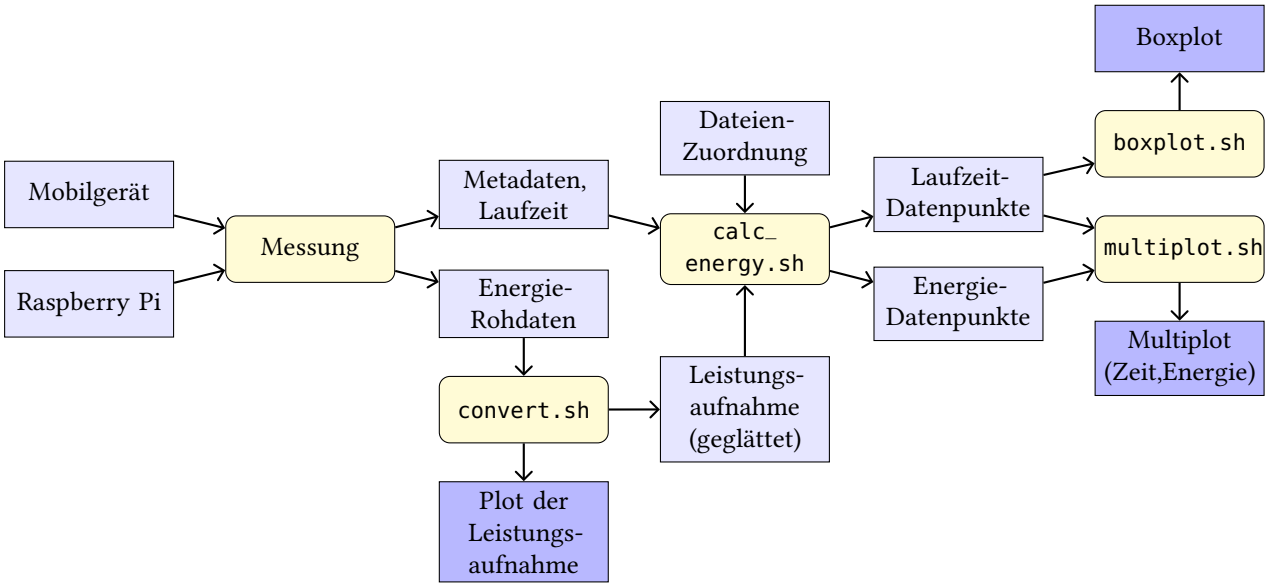

Abbildung 6.3: Workflow der Datenanalyse und -visualisierung der Messungen im Rahmen der Voruntersuchung

6.3 Durchführung der Messungen

Abbildung 6.3 stellt den Workflow der Durchführung und Analyse der Energiemessungen vereinfacht dar.

Nach abgeschlossenen Messungen wurden die Daten durch Bash- und Python-Skripte aufbereitet und mithilfe von `Gnuplot` auf geeigneten Schaubildern visualisiert. Einzelne Skripte sind jeweils als eine Transformation einer Menge von Eingabedateien auf eine Menge von Ausgabedateien zu verstehen, wobei die Ausgabedateien mit jedem weiteren Analyseschritt eine aggregiertere und komprimiertere Form annehmen. Diese Transformationen und Berechnungen wurden auf Seiten der `bash`-Skripte größtenteils von den Utilities `awk`, `sed` und `grep` vorgenommen. Dabei haben die Datendateien eine einfache, tabellarische Struktur mit Leerzeichen als Feldtrennung, sodass sie ohne weitere Modifikationen von Gnuplot als Datenpunkte eingelesen werden können.

Für die Evaluation wurde ähnlich wie in Abbildung 6.3 vorgegangen. Neue Anforderungen umfassten insbesondere die Ermittlung der Fehler des aufgestellten Energiemodells $W(p)$ (vorliegend als mathematische, von Gnuplot lesbare Funktion) zu den Prüfmessungen, was hauptsächlich mit `awk` umgesetzt wurde. Für die Visualisierung und die separate Berechnung der verschiedenen vermuteten Phasen mussten ebenfalls weitere Skripte geschrieben werden.

6.3.1 Synchronisation der Daten

Die Messdaten des Raspberry Pi enthalten Zeitstempel in Nanosekunden gezählt seit dessen Bootvorgang. Die Zeitstempel in den Logs der App geben analog die Anzahl der Nanosekunden seit dem Bootvorgang des Handys wieder. Da hier eine exakte Synchronisation der Bootvorgänge weder durchführbar noch praktikabel ist, wurde die Zusammenführung der Daten durch eine Bestimmung des Versatzes vorgenommen.

Hierbei wurde eine beliebige Berechnung hinreichend hoher Qualität ausgewählt und dessen zeitabhängige Leistungsaufnahme mit einem interaktiven Gnuplot-Fenster visualisiert. Der Beginn der Berechnung lässt sich hier durch Vergrößerung recht genau (im Bereich von Mikrosekunden) feststellen. Die Differenz dieses Zeitstempels zum Zeitstempel `CALCULATION_START` des dazugehörigen App-Logs führt zum genauen Versatz, mit dem sich die Zeitstempel ineinander umrechnen lassen. Dieser Versatz ist dann gültig für die komplette Reihe von Messungen, da ein Reboot während der Messungen vermieden wurde. Mögliche Taktversätze der beiden Geräte hatten keine merklichen Auswirkungen; der Vergleich verschiedener Start-Zeitstempel mit den visualisierten Leistungsaufnahmen ergab auch bei zeitlich späteren Ausführungen ein exaktes Bild, wenn die erste Ausführung für die Ermittlung des Zeitversatzes verwendet wurde.

Mit dieser Technik lässt sich dann durch den angegebenen Start und die Zeitdauer der Berechnung aus dem App-Log sowie durch die Messdaten aus dem Log des Raspberry eine genaue Integration des Energieverbrauchs der Berechnung erzielen.

7 Evaluation

Um die Eignung der entwickelten Energiemodelle zur Vorhersage eines bestimmten Energieverbrauchs zu ermitteln, wurden weitere Messungen durchgeführt, auf deren Grundlage nach der erarbeiteten Vorgehensweise Modelle aufgestellt und danach mit Testmessungen verglichen wurden.

7.1 Methodologie

Zunächst wird die Vorgehensweise der Evaluation vorgestellt. Diese ist aufgeteilt in die Auswertung des initialen Modells und die Auswertung des erweiterten Modells; beide Modelle wurden separat ausgewertet, wobei das erweiterte Modell zusätzlich mit einer Anwendung des initialen Modells verglichen wurde.

7.1.1 Initiales Energiemodell

Für das initiale Modell (Kapitel 5.2), das einen proportionalen Zusammenhang zwischen der Zeitdauer der Berechnung und dem Energieaufwand derselben aussagt, wurde eine automatisierte, sequentielle Durchführung bestimmter Berechnungen implementiert, für die jeweils die genaue Zeitdauer aufgenommen wurde. Nach Abschluss aller Berechnungen folgte die Herstellung eines Modells (mit linearen Splines bzw. mit polynomieller Regression bis zu einem festen Grad) ausgehend von den Daten. Sodann wurden für einen einzelnen Qualitätsparameter mehrere Berechnungen inklusive Energiemessung durchgeführt, um den Koeffizienten α zu bestimmen, für den näherungsweise $W(p) = \alpha T(p)$ gilt. Schließlich wurden weitere Energie- und Zeitmessungen durchgeführt, deren realer Verbrauch dann mit der Vorhersage aus dem erlernten Modell verglichen wurde.

7.1.2 Mehrstufige Berechnungen

Um die Methodik des Energiemodells über verschiedene Energiezustände hinweg (Kapitel 5.3) zu evaluieren, wurde eine beispielhafte, qualitätsbewusste Berechnung der Canny-Kantenerkennung implementiert (Kapitel 6.1.2), bei der ein Bild eingelesen, skaliert, in Graustufen konvertiert und anschließend der Kantenerkennung unterzogen wird. Im Vorfeld wurden

Indikatoren für Energiezustandsübergänge identifiziert und gemäß des vorgestellten Vorgehens mit Zeitmarkierungen versehen. Mehrere Messungen für verschiedene Qualitätsparameter ($n = 200, 400, 600, \ldots, 1600$) wurden durchgeführt und die durchschnittlichen Leistungsaufnahmen P_i der möglichen Energiephasen sowie deren Dauer $T_i(p)$ berechnet (letztere mit einer polynomiellen Regression). Im Anschluss wurden anhand dieser Ergebnisse die tatsächlichen Energiephasen der Berechnung identifiziert und eine Formel für die Gesamtleistung aufgestellt. Schließlich wurde diese Formel dazu verwendet, Werte für den Energieverbrauch vorherzusagen und die Werte wurden mit tatsächlichen Messungen verglichen.

Zudem wurde für die mehrstufige Berechnung nochmals das initiale Modell aufgestellt, um einen direkten Vergleich der Modelle zu ermöglichen.

7.2 Ergebnisse

Im Folgenden werden die Ergebnisse der beschriebenen Evaluationen vorgestellt, interpretiert und davon ausgehend die erarbeiteten Modelle bewertet.

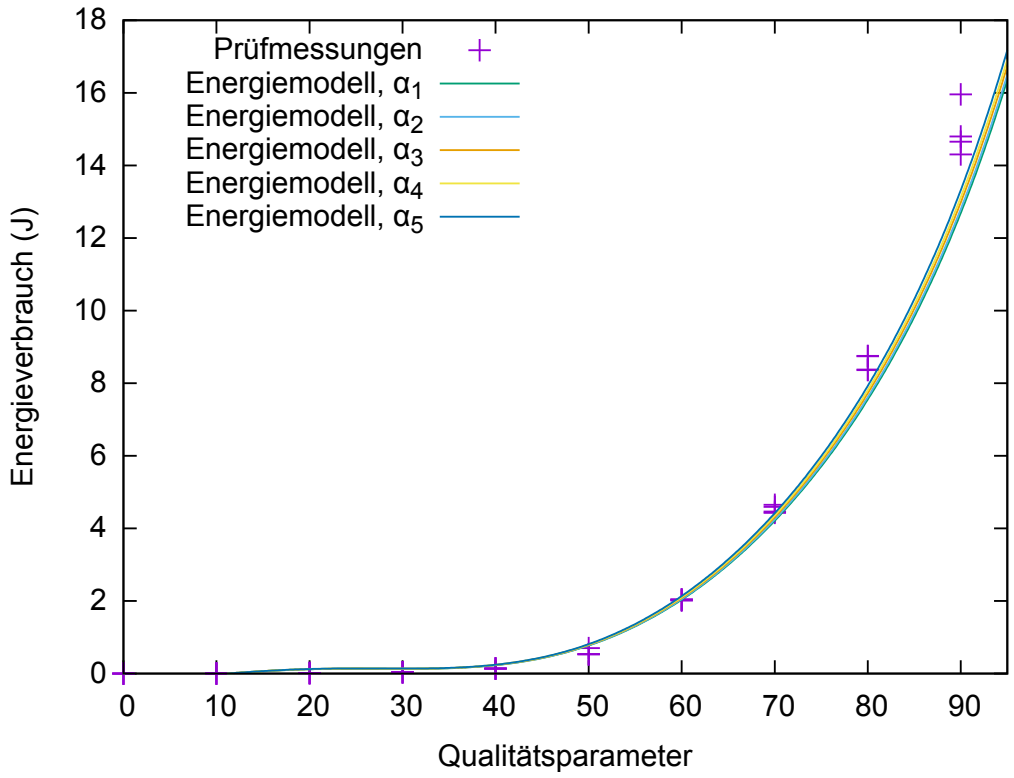

Abbildung 7.1: Vergleich des erlernten Modells mit dem tatsächlichen Energieverbrauch, mit den Quartilen der zwölf gemessenen Zeit-Energie-Koeffizienten

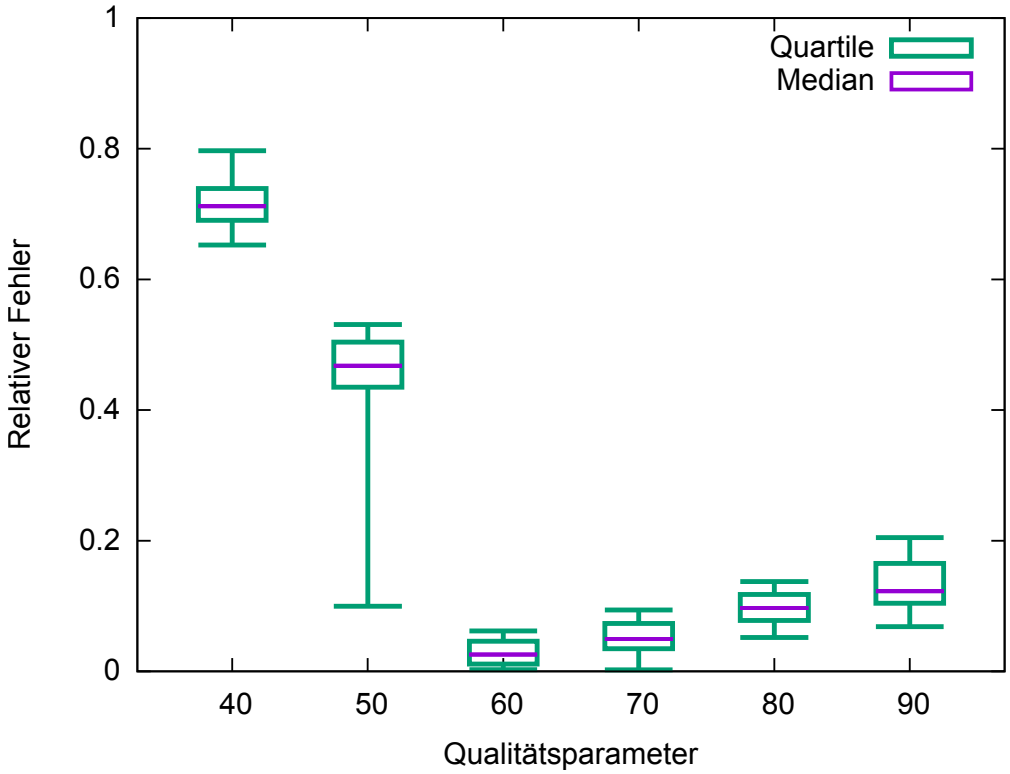

Abbildung 7.2: Relative Fehler der Vorhersage zur anschließenden Messung, abhängig von der Qualität, über alle Zeit-Energie-Koeffizienten α_i

7.2.1 Initiales Energiemodell

Die ermittelte Modellfunktion gemäß der im Entwurf beschriebenen Vorgehensweise lautet gerundet

$$W(n) = \alpha(1.33E-7x^6 + -3.70E-5x^5 + 3,99E-3x^4 - 0.185x^3 + 3.74x^2 - 23.941x + 3.86)$$

für ein geeignetes α, das der durchschnittlichen Leistungsaufnahme während der Berechnung entspricht.

Die zusätzlichen Energiemessungen bei $n = 60$, um ein α zu bestimmen, ergaben für diese Qualität einen Energieverbrauch zwischen 2,14J und 2,49J; die daraus errechneten α_i sind damit gestreut zwischen 1,84254 und 1,93405.

Für jedes dieser α_i wurden schließlich die entsprechenden $W_i(n)$ mit vergleichenden Testmessungen verglichen. In Abbildung 7.2 sind für jeden gemessenen Qualitätsparameter die quadrierten, relativen Fehler ($e = \frac{(W(p)-W_{gemessen})^2}{W_{gemessen}}$) zusammengefasst.

Das errechnete Modell stimmt an der Stelle, an welcher Energiemessungen durchgeführt wurden, genau mit den Prüfmessungen überein, während es in Richtung geringer sowie

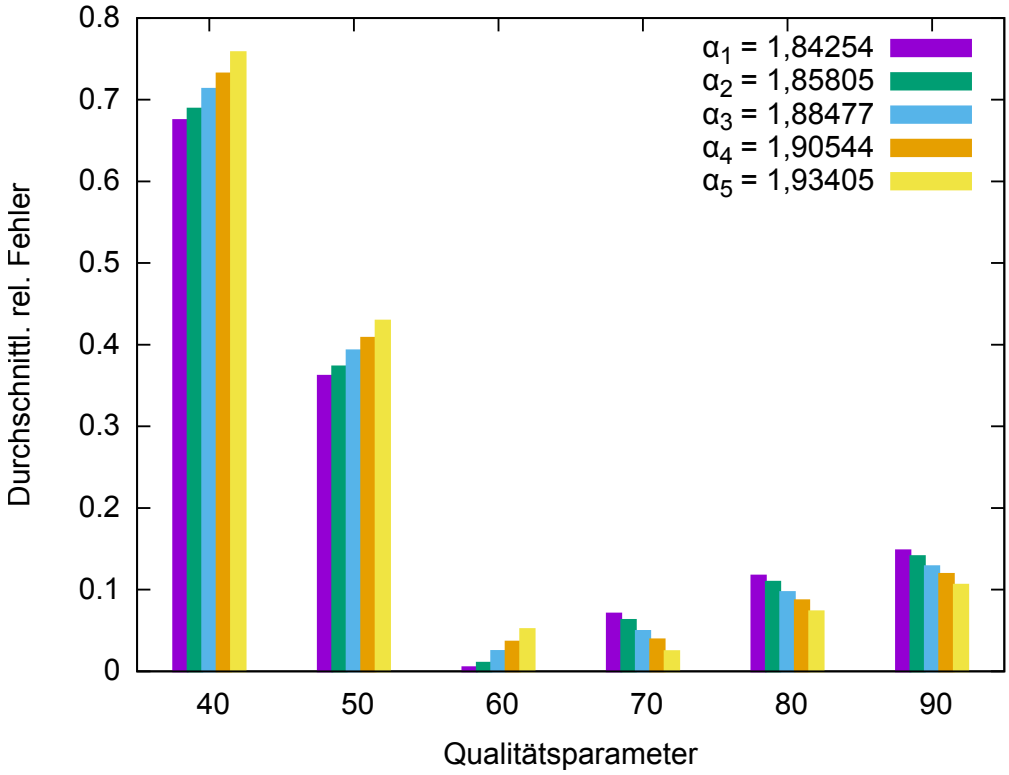

Abbildung 7.3: Vergleich der durchschnittl. relativen Fehler unter verschiedenen α

hoher Qualität zunehmend von der Realität abweicht. Wie in Abb. 7.1 zu sehen, liegt das aus vorherigen Laufzeiten errechnete Modell am Punkt $n = 60$ genau auf den Prüfmessungen. Für geringere Qualität fällt die Vorhersage dagegen höher und für höhere Qualität geringer aus. Auch in Abbildung 7.2 sind die Abweichungen zwischen Modell und Realität für $n = 60$ minimal, während sie in beide Richtungen hiervon zunehmen. Für Qualitätswerte $p \geq 60$ wurden Fehler bis maximal 20% beobachtet, doch für kleinere Qualität ergaben sich sehr hohe relative Abweichungen, etwa bei $n = 40$ bis zu 80%. Qualitätswerte kleiner als 40 wurden aus Gründen der Übersicht nicht abgebildet; die Fehler steigen für diese kleinen Werte noch weiter an und erreichen Abweichungen bis um das Achtfache der Realität.

Die Wahl des α kann konservativ, ausgeglichen oder optimistisch geschehen; gemäß Abbildung 7.3 kann ein konservativ gewähltes α, wie hier α_5, den Fehler für hohe Qualitäten möglichst gering halten; dafür werden jedoch die Fehler für niedrigere Qualitäten höher. Lässt man niedrige Qualitäten aufgrund ihres (absolut gesehen) sehr geringen Energieverbrauchs außer acht, so ist in hier das höchste α die beste Wahl, ausgehend von der Gesamtheit der relativen Fehler. Die Wahl eines hohen α (was einer vorsichtigeren Vorhersage entspricht) ist zusätzlich von Vorteil, da somit die Gefahr der Überschreitung einer Energieschranke verringert wird.

Zur Erklärung der festgestellten Abweichungen des Modells kann ein Ausschnitt der Leistungsaufnahme während den automatisch ausgeführten Messungen herangezogen werden

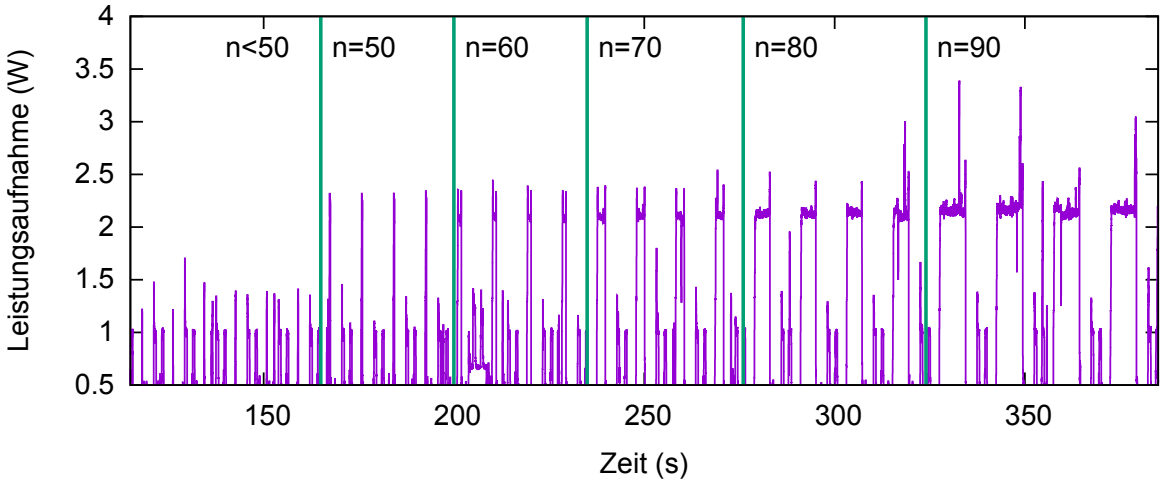

Abbildung 7.4: Leistungsaufnahme im zeitlichen Verlauf des Profilings; die vertikalen Linien grenzen Ausführungen verschiedener Qualitätsparameter voneinander ab

(Abbildung 7.4). Auffallend ist hier der Mangel von Leistungsspitzen um 2 bis 2,5W für Qualitätsparameter $n < 50$ (vgl. hierzu Abb. 4.2, in welcher ein entsprechendes Plateau dieser Höhe sichtbar ist). Für $n \geq 50$ existieren Leistungsaufnahmen dieser Höhe, und mit zunehmender Qualität treten zunehmend weitere Spitzen bis hin zu 3,5W auf. Das Modell, das eine konstante Leistungsaufnahme variabler Zeit annimmt, trifft daher für kleine sowie für große n nicht den genauen Energieverbrauch.

Die Herkunft der Ungenauigkeiten des Modells für *niedrige* Qualität kann leicht erklärt werden: die Voruntersuchungen, die die Basis für das Modell darstellen, wurden nur für Qualitätsparameter $n \geq 45$ durchgeführt; kleinere n sind hier nicht berücksichtigt. Für diese n ist daher zu erwarten, dass die aus dem Profiling hervorgegangenen niedrigen Leistungswerte auch bei isolierten Ausführungen auftreten. Das Modell kann also den realen Energieverbrauch für sehr geringe Qualitäten nicht vollständig darstellen. Relativierend ist hierbei aber zu sagen, dass bezogen auf die Anwendung der Energieverbrauch in diesem Bereich uninteressant ist, da er sehr niedrig ist (kleiner als 0,1J) und daher insbesondere in Kombination mit weiteren Teilen der Anwendung wie Visualisierung und Nutzerinteraktion nicht ins Gewicht fällt. Aus diesem Grund wurde auch die Voruntersuchung nur mit Qualitätsparametern ab $n = 45$ durchgeführt.

Die Abweichungen im Bereich hoher Qualität sind dagegen der verwendeten Methodik zuzuordnen. In der Voruntersuchung wurde durch mehrere Messungen festgestellt, dass Laufzeit und Energieverbrauch auch für hohe Qualität in guter Näherung proportional zusammenhängen; in der Evaluation zeigt sich für denselben Algorithmus im Bereich hoher Qualität jedoch ein abweichendes Bild. Damit ist die Art der Durchführung des in der Evaluation verwendeten Profilings als mögliche Ursache für die Abweichungen zu berücksichtigen. Die Datenpunkte der Voruntersuchung wurden durch einzelne manuelle Starts der Berechnung

mit entsprechenden Pausen ermittelt, während für das Profiling eine automatisierte Routine alle Berechnungen hintereinander ausgeführt hat. Trotz entsprechender `Thread.sleep()`-Anweisungen innerhalb der Hintergrundthreads (jeweils vor und nach der Berechnung) war hier zu praktisch jedem Zeitpunkt genau ein Hintergrundthread anwesend, während bei vorherigen Messungen jeweils ein gewisser natürlicher Abstand (eine Ruhephase) zwischen den Berechnungsprozeduren lag. Die erhöhten Leistungsspitzen gegen Ende des Profilings könnten daher interne Symptome der Ressourcenauslastung darstellen, beispielsweise erhöhte Aktivität der Java-Speicherverwaltung (Heap-Vergrößerung und Garbage Collection).

Sofern man berücksichtigt, dass die Ergebnisse durch die Methodik etwas verfälscht wurden, sind die Vorhersagen des Modells im Gesamten zufriedenstellend, und geben zusammen mit den Voruntersuchungen Anlass zur Erwartung, dass bei einer umsichtigen Re-Implementierung des automatischen Profilings (etwa mit „tatsächlichen“ Ruhephasen der CPU zwischen den Berechnungen) akkurate Vorhersagen erzielbar sind. Die im Folgenden beschriebene Evaluation des erweiterten Energiemodells hat diese Erwartung (für eine andere Berechnung) bestätigen können.

7.2.2 Mehrstufige Berechnungen

Es wird die Durchführung der Evaluation des erweiterten, phasenbasierten Modells beschrieben und die Ergebnisse erörtert.

Alle Übergänge im Anwendungscode, die potentiell eine Änderung der Leistungsaufnahme herbeiführen könnten, wurden mit Zeitmarkierungen versehen. Folgende Punkte in der Berechnung wurden markiert:

1. Start der Berechnung
2. Nach dem Einlesen des (unveränderten) Bilds
3. Nach der Berechnung der Skalierung
4. Nach dem Abspeichern des skalierten Bilds
5. Nach der Konversion des skalierten Bilds in Graustufen (in-place)
6. Nach der Kantenerkennung (in-place)
7. Nach dem Abspeichern des Bilds mit erkannten Kanten (Ende der Berechnung)

Bei der gemessenen Leistungsaufnahme der vermuteten Phasen (Abbildung 7.5) kann neben einer allgemein recht hohen Streuung festgestellt werden, dass die erste Phase mit einem Median von 1,67 W ein signifikant geringeres Leistungsniveau aufweist als die darauf folgenden Phasen mit den Medianen 2,25W bis 2,48W. Die Streuung ist bei den Phasen 1 und 5 mit einer relativen Abweichung von 9,3% und 10,0% gegenüber des Durchschnittswerts deutlich geringer als bei den restlichen Phasen mit Abweichungen von 16,3% bis hin zu 26,4%.

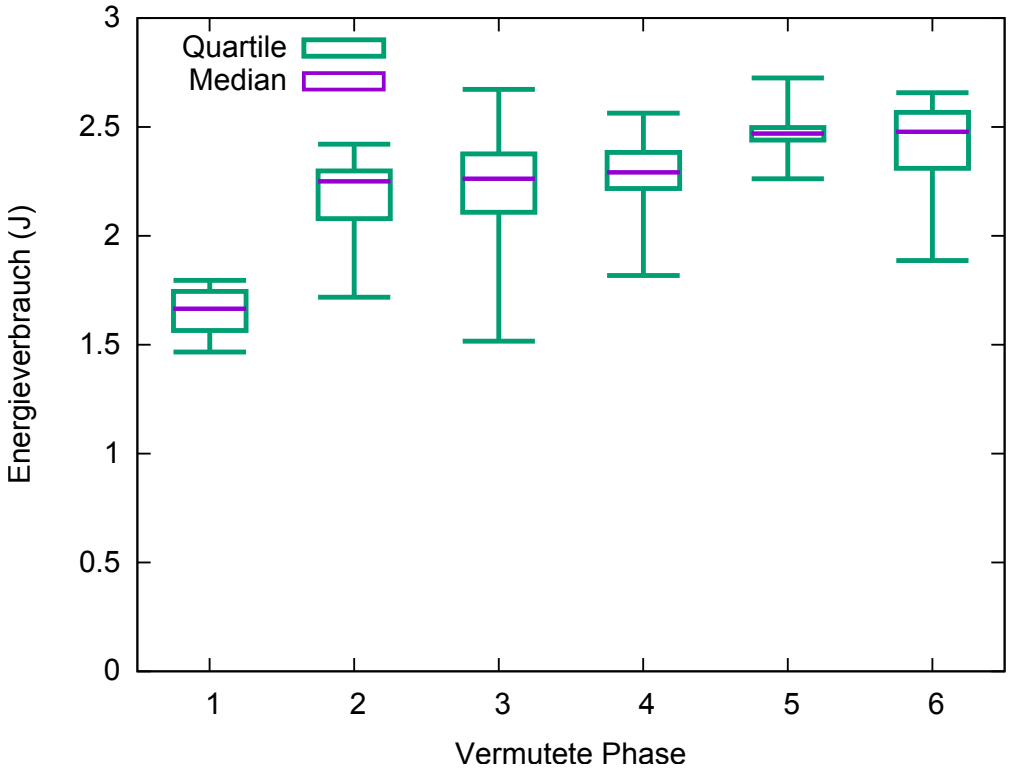

Abbildung 7.5: Streuung der vermuteten Phasen bzgl. ihrer durchschnittlichen Leistungsaufnahme

Diese geringe Streuung der ersten Phase ist dadurch zu erklären, dass die erste Phase unabhängig von der Qualität stets genau dieselbe Aufgabe erfüllt, und zwar das unskalierte Bild einzulesen. Dementsprechend kann das Modell – unter einem gewissen Verlust von Genauigkeit – auf zwei Phasen reduziert werden; das Einlesen des unskalierten Bildes und die restliche Berechnung.

Zur weiteren Einschätzung der Berechnungsphasen werden die Graphen der Leistungsaufnahmen für verschiedene Qualitätsparameter hinzugezogen. Abbildung 7.6 zeigt die Leistungsaufnahme einer einzelnen Berechnung mit Bildskalierung auf 200px, wobei die potentiellen Phasenübergänge durch vertikale Linien eingetragen sind. Die erste Phase – das Skalieren des ursprünglichen Bilds – entspricht nicht einem Plateau, sondern eher einem unregelmäßigen Anstieg der Leistungsaufnahme von 0,5W auf 2,4W. Während Phase 5 besteht eine Leistungsaufnahme von durchschnittlich 2,45W; die restlichen Phasen leisten mit einer summierten Laufzeit von 26,5 ms (3,9% der Gesamtlaufzeit) keinen signifikanten Beitrag zum Energieverbrauch. Verglichen mit der gleichen Berechnung, jedoch einer Bildskalierung auf weitaus höhere 1600px (s. Abb. 7.7) ist wiederum zu erkennen, dass die Einlesephase im Verhältnis zur gesamten Berechnung nur noch einen sehr kleinen Anteil an der Gesamtberechnung einnimmt (5,2% des Energieverbrauchs und 7,9% der Laufzeit), da die Zeitdauer der restlichen Phasen

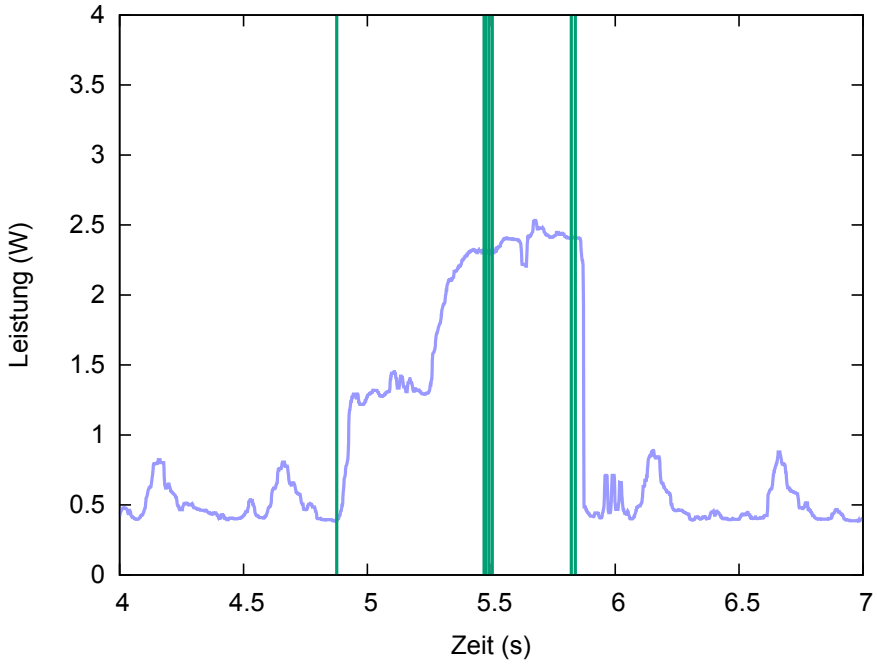

Abbildung 7.6: Leistungsaufnahme für eine Skalierung des Bilds auf 200px Breite; vertikale Linien geben die potentiellen Energiezustandsübergänge an

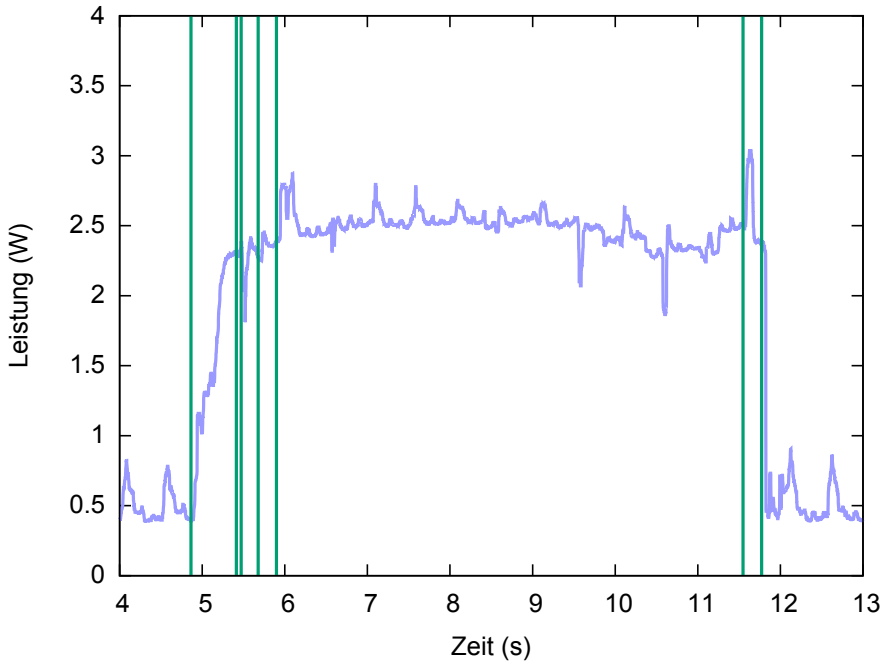

Abbildung 7.7: Leistungsaufnahme für eine Skalierung des Bilds auf 1600px Breite

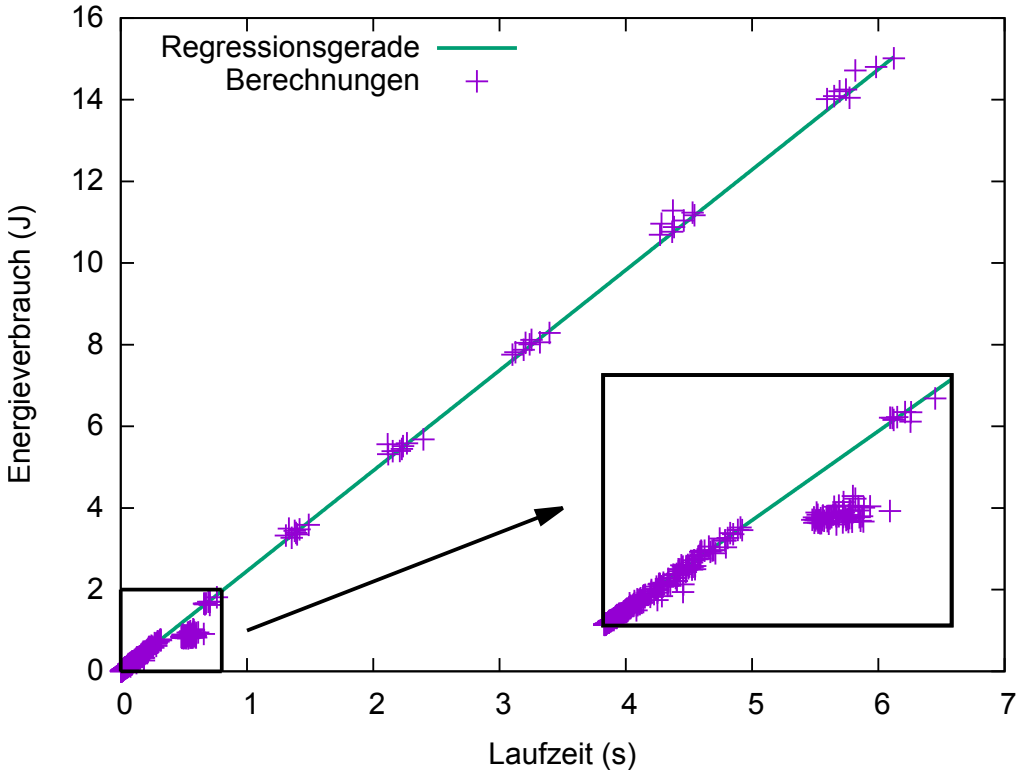

Abbildung 7.8: Abbildung der Laufzeiten verschiedener Ausführungen der Canny Edge Detection auf den dazugehörigen Energieverbrauch, mit Regression $f(x) = mx$

entsprechend stark zugenommen hat. Im Umkehrschluss wird in diesem Bereich hoher Qualität die Näherung der Leistungsaufnahme auf ein einzelnes, großes Plateau zunehmend besser. Im Bereich geringerer Qualität kann diese Vereinfachung jedoch zu großen relativen Fehlern führen.

Für die vorliegende Berechnung ist die Annahme eines proportionalen Zusammenhangs zwischen Laufzeit und Energieverbrauch nicht mehr vollständig korrekt. Abbildung 7.8 zeigt die Abbildung der Laufzeiten verschiedener Ausführungen der Canny Edge Detection auf den dazugehörigen Energieverbrauch, gemeinsam mit einer Regressionsgeraden. Es fällt auf, dass die Regression hier nicht akkurat ist; kleine Laufzeiten im Bereich um 0,5 Sekunden kann das ursprüngliche Modell hier nicht erfassen.

Auch wenn für die vorliegende Implementierung der Kantenerkennung die Einführung des erweiterten Modells nicht zwangsläufig notwendig wäre, um im Bereich größerer Berechnungen akzeptable Voraussagen zu erhalten, zeigt sich durch die mehrphasige Berechnung die Notwendigkeit der Erweiterung. Denn sobald unterschiedliche Phasen in ihrer energetischen Relevanz abhängig von der Qualität variieren, kann das initiale Modell mit proportionalem Zusammenhang von Zeit und Energie keine akkuraten Voraussagen mehr treffen.

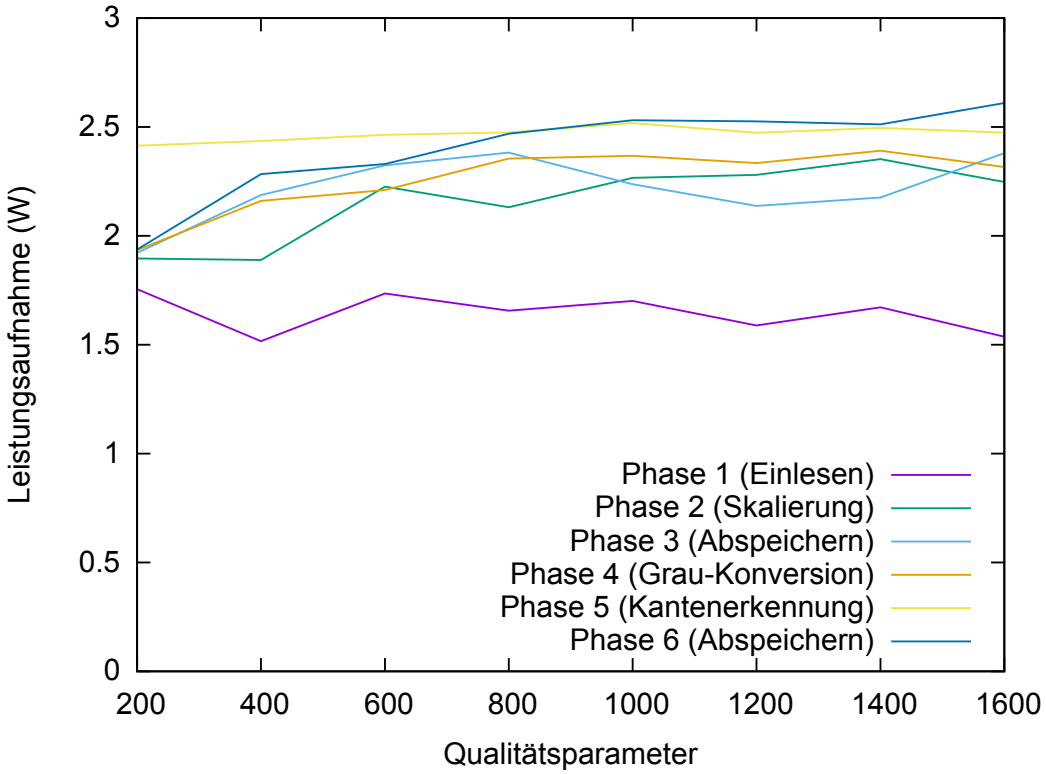

Abbildung 7.9: Durchschnittliche Leistungsaufnahmen der einzelnen potentiellen Phasen in Abhängigkeit des Qualitätsparameters

Um zu evaluieren inwieweit Energiemessungen für viele verschiedene Qualitätsparameter notwendig sind oder möglicherweise nur für eine einzige Qualität durchgeführt werden müssen, werden die durchschnittlichen Leistungsaufnahmen der einzelnen, vermuteten Phasen in Abhängigkeit der Qualität betrachtet. Abbildung 7.9 zeigt die Schwankungen der Leistungsaufnahmen verschiedener Phasen in Abhängigkeit der Qualität. Die erste (Einlese-)Phase schwankt zwischen den Werten 1,5W und 1,8W und könnte somit auch durch den Wert einer einzelnen Qualität angenähert werden. Die restlichen Phasen schwanken teilweise stärker, doch ein klarer Aufwärtstrend für hohe Qualität ist nur bei den Phasen 3 und 6 auszumachen; diese entsprechen der jeweiligen Abspeicherung des skalierten Bilds sowie der fertigen Kantenerkennung. Günstigerweise sind diese Phasen unabhängig von der Qualität sehr klein im Verhältnis zur gesamten Berechnung (vgl. Abb. 7.7), weshalb der Anstieg in der Leistungsaufnahme keine erheblichen Ungenauigkeiten des Modells bewirkt. Die Phase, welche der eigentlichen Berechnung entspricht – Phase 5 – ist dagegen die stabilste. Hierbei kann vermutet werden, dass die vergleichsweise lange Laufzeit dieser Phase insgesamt zu einer geringeren Streuung führt („Gesetz der großen Zahlen“), was dem Modell im Allgemeinen zugute kommt.

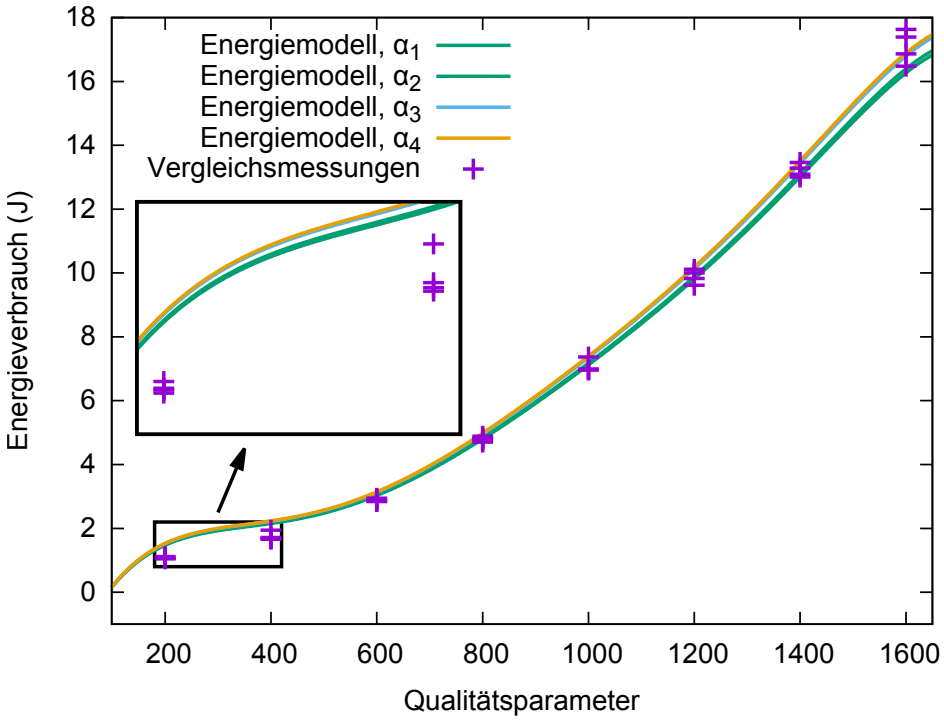

Abbildung 7.10: Initiales Modell für Kantenerkennung mit Prüfmessungen

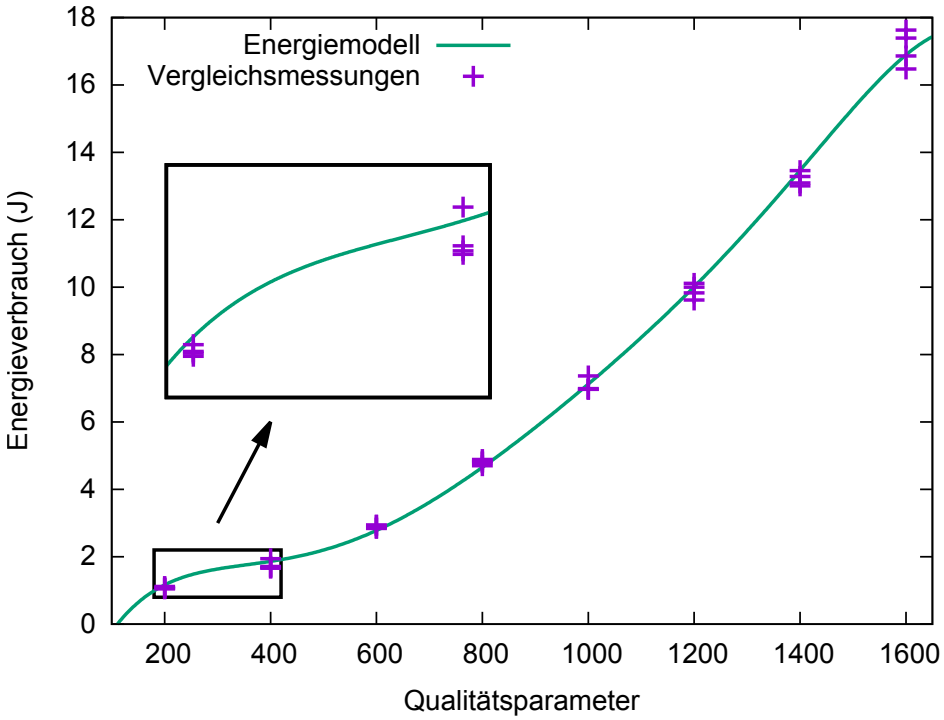

Abbildung 7.11: Erweitertes Modell für Kantenerkennung mit Prüfmessungen

Vergleich der Modelle

Aus den zugrunde liegenden Daten für die Kantenerkennungs-Berechnung wurde sowohl das initiale Modell als auch das erweiterte Modell errechnet, um einen direkten Vergleich zu ermöglichen. Um das initiale Modell zu erhalten, wurde aus den Laufzeiten für die einzelnen Qualitätsstufen gemäß des Entwurfs eine polynomielle Regression berechnet und Energie-Zeit-Koeffizienten $\alpha_i, i \in 1..4$ mithilfe von vier Energieverbrauchswerten bei $n = 1000$ berechnet. Das Ergebnis des initialen Modells, verglichen mit separaten Prüfmessungen, ist in Abb. 7.10 zu sehen. Das Modell ist erwartungsgemäß immer noch leidlich realistisch, da für größere Werte keine wirklichen relevanten Energiephasen bestehen. Im hier kritischen Bereich, in dem sowohl die Einlese- als auch die Berechnungsphase einen relevanten Anteil am gesamten Energieverbrauch besitzen (s. Werte bei $n = 200$ und $n = 400$), weicht das Modell sichtbar von den tatsächlichen Messungen ab. Dies lässt sich so veranschaulichen, dass durch das initiale Modell für die gesamte Laufzeit ein einziges Rechteck bzgl. der Leistungsaufnahme integriert wird. Der lineare Anstieg der Einlesephase wird hier unzureichend berücksichtigt, was zu einer falschen, weil zu hohen Vorhersage führt.

Das erweiterte Modell wurde auf Grundlage zweier Phasen berechnet; der Einlesephase (vormals *Phase 1*) und der Berechnungsphase (*Phase 2 bis 6*). Für jede dieser beiden Phasen wurde wiederum pro Qualität die zweithöchste Laufzeit verwendet und eine polynomielle Regression auf diesen Punkten berechnet. Die Zusammenführung der beiden resultierenden Funktionen $\{T_1(p), T_2(p)\}$ ergab sich gemäß Gleichung 5.7. Die nötigen Durchschnittsleistungen P_1 und P_2 wurden aus den Medianen der Leistungsaufnahmen errechnet, die sich bei den Berechnungen während den entsprechenden Phasen ergeben haben.

Der Vergleich des Modells mit Prüfmessungen zeigt, dass die tatsächlichen Werte über den gesamten Qualitätsbereich hinweg mit hoher Genauigkeit getroffen werden. In Abb. 7.11 ist gut zu erkennen, dass das erweiterte Modell in den höheren Bereichen ebenso genau ist wie das initiale Modell, jedoch auch im kritischen Bereich um $n = 200$ bis $n = 400$ einen korrekten Energieverbrauch vorhersagt. Es ergibt sich über alle Datenpunkte hinweg eine durchschnittliche Abweichung von 4,1% und einzelnen Abweichungen zwischen 0% und 12%.

Die Ergebnisse der durchgeführten Berechnung zeigen nur wenig Verbesserung des erweiterten Modells gegenüber dem initialen, da für die Bereiche relevanter Qualität die Näherung auf eine konstante Leistungsaufnahme hinreichend genau ist. Für Berechnungen, bei welchen die beobachteten Ungenauigkeiten im Bereich höherer Qualität auftreten, ist die Modellerweiterung jedoch sinnvoll und notwendig. Dabei hat es hier bereits genügt, eine sehr grobe Einteilung in Phasen vorzunehmen, um sehr genaue Ergebnisse zu erzielen.

Ein Nebenergebnis der Evaluation des erweiterten Modells ist, dass (verglichen mit den Abweichungen bei der Evaluation des initialen Modells) auch für hohe Qualitäten gute Vorhersagen getroffen werden (vgl. Abb. 7.10). Da hier eine andere Methodik verwendet wurde (manuelles Starten der einzelnen Berechnungen anstatt einem automatisierten, pausenlosen Start der

Berechnungsthreads) unterstützt dies die Vermutung, dass die Implementierung des automatischen Profilings und die damit zusammenhängenden Aktivitäten des Betriebssystems zu einem veränderten Ergebnis bei der Evaluation des initialen Modells geführt haben.

7.2.3 Bewertung der Modelle

Abschließend wird die Genauigkeit der Modelle bewertet und deren Eignung in Hinblick auf verschiedene Anwendungsfälle eingeschätzt.

Sollte sich die vorliegende Berechnung nicht um eine konzeptionell simple handeln (wie das Diffusion-Advection-Problem), sondern zusätzliche Schnittstellen (auf den Speicher, das Netzwerk oder weiteres) aufweisen, ist das erweiterte Modell dem initialen vorzuziehen, da die stärkere Differenzierung zu genaueren Ergebnissen führen wird. Da es sich um eine Erweiterung handelt, enthält das phasenbasierte Modell insbesondere alle Fähigkeiten des initialen Modells und kann daher im Prinzip grundsätzlich dem initialen Modell vorgezogen werden. Zudem ist für das erweiterte Modell eine ähnlich niedrige Zahl von Energiemessungen vertretbar wie im initialen Modell.

Das phasenbasierte Vorhersagemodell kann für die behandelten Berechnungen verwendet werden und ergibt dabei ein realistisches Bild des tatsächlichen Energieverbrauchs. Ungenaue Ergebnisse sind dagegen – auch beim erweiterten Modell – dann zu erwarten, wenn nicht nur die *Laufzeit*, sondern zudem die *Leistungsaufnahme* in Abhängigkeit mit der Qualität erheblich skaliert. Bei der Berechnung zum Diffusion-Advection-Problem ist dies im Bereich sehr kleiner Qualität aufgetreten, ebenso wie in Abb. 7.9 bei einigen der Phasen ersichtlich. Diese beobachteten Abweichungen von einer konstanten Leistungsaufnahme hatten jeweils nur eine geringe Relevanz und die errechneten Energiemodelle konnten somit zufriedenstellende Ergebnisse aufweisen. Bei einer Anwendung, bei welcher die Leistungsaufnahme über das relevante Qualitätsintervall hinweg signifikant ansteigt, können die aufgestellten Modelle bei konventioneller Nutzung keine genauen Ergebnisse liefern.

Denkbar wäre unter anderem ein nicht kontinuierlicher, sondern sprunghafter Anstieg der Leistungsaufnahme, sofern ab einer gewissen Qualität eine neue Lösungsstrategie verfolgt wird oder das Betriebssystem vor Herausforderungen gestellt wird, die bei kleineren Eingaben nicht bestehen (vgl. die unterschiedlichen Leistungsspitzen bei Abb. 7.4 in den Bereichen $n < 50$, $50 \leq n < 90$ sowie $n \geq 90$). In diesem Fall kann der Energieverbrauch derart modelliert werden, dass eine Partitionierung des Qualitäts-Definitionsbereichs gefunden wird, in welchen die Leistung jeweils annähernd konstant ist und für jede dieser Partitionen ein separates Energiemodell (wie zuvor) berechnet werden kann. Für eine zu hohe Anzahl solcher Intervalle (bis hin zu einem kontinuierlichen Anstieg der Leistungsaufnahme) degeneriert diese Methode jedoch zu einer Modellerstellung „ausschließlich“ durch Energiemessungen, wobei die Aufstellung separater Laufzeit-Performance-Profile keinen Vorteil mehr mit sich bringt.

8 Zusammenfassung und Ausblick

Im Folgenden werden die zentralen Punkte der Arbeit zusammengefasst und Konsequenzen daraus gezogen. Ein Ausblick weist auf mögliche anschließende Forschungsarbeiten hin.

8.1 Zusammenfassung

Die heute allgegenwärtigen Mobilgeräte sind äußerst vielseitig einsetzbar, leiden jedoch unter den Konsequenzen eines hohen Energieverbrauchs. Ziel der Arbeit war es daher speziell für Simulationen, die qualitätsbewusst implementiert und berechnet werden können, eine Methodik für die Aufstellung eines Energiemodells zu erarbeiten, welche ermöglicht, die Energieausgaben einer Berechnung mit gegebenem Qualitätsparameter vorherzusagen sowie den Parameter unter der Einhaltung einer gegebenen Energieschranke zu optimieren.

Um den Energieverbrauch mobiler Geräte bei konzeptionell einfachen Simulationsberechnungen zu erfassen, wurden Messungen verschiedener Ausführungen des Diffusion-Advection-Problems durchgeführt. Dabei wurde festgestellt, dass Laufzeit und Energieverbrauch in Abhängigkeit der Qualität proportional zueinander verlaufen. Dieser Zusammenhang wurde genutzt, um ein einfaches Energiemodell aufzustellen, das hauptsächlich auf Laufzeitmessungen beruht und das zur Laufzeit Aussagen über den voraussichtlichen Energieverbrauch treffen kann. In der Evaluation zeigte sich unter anderem, dass die Implementierung eines automatisierten Profilings umsichtig geplant werden muss, um keine Ergebnisse zu erhalten, die von den tatsächlichen Ausführungen der Berechnung abweichen.

Für Berechungen, die aus unterschiedlichen Phasen bestehen, also im Verlauf der Berechnung Abschnitte mit unterschiedlichen Leistungsaufnahmen aufweisen, reicht dieses Modell jedoch nicht aus, da der proportionale Zusammenhang zwischen Laufzeit und Energieverbrauch nur für die einzelnen Phasen, jedoch nicht für die gesamte Berechnung gilt. Infolge dessen wurde das Modell auf ein mehrstufiges erweitert, wobei durch Energie- und Zeitmessungen sowie durch Analyse des Quellcodes Phasen der Berechnung identifiziert werden, die dann wiederum mit der Vorgehensweise des ursprünglichen Modells einem Profiling unterzogen werden. Eine Evaluation mit Berechnungen der Canny-Kantenerkennung zeigte die Notwendigkeit der Modellerweiterung ebenso wie die Verbesserung der Güte der Voraussagen durch das erweiterte Modell.

Grenzen der Genauigkeit des Modells deuteten sich insbesondere für mögliche Anwendungsfälle an, in welchen die Leistungsaufnahme einer Berechnung (oder einer relevanten Berechnungsphase) im Bereich der untersuchten Qualität signifikant ansteigt.

Die Untersuchungen im Rahmen der Arbeit haben gezeigt, dass es sich bei der Modellierung des Energieverbrauchs mobiler Geräte um ein vielschichtiges Problem handelt, welches aber unter bestimmten Annahmen dennoch durch überraschend einfache Methoden zufriedenstellend erfasst werden kann. Die angenäherte Proportionalität von Laufzeit und Energieverbrauch – ob auf Ebene der gesamten Berechnung oder auf Ebene einzelner Phasen – eröffnet dabei eine Reihe von Möglichkeiten, die das Aufstellen des Modells deutlich vereinfachen.

Der Einsatz qualitätsbewusster Algorithmen ist angesichts der heutigen Situation mobiler Geräte ein vielversprechendes Paradigma, um den Energieverbrauch mobiler Applikationen zu verringern und dabei deren Eignung erheblich zu erhöhen. Dieser Ansatz kann als eine Form *defensiver Programmierung* angesehen werden, welche nicht die makellose Ausführung der Anwendung in den Mittelpunkt stellt, sondern vielmehr die optimale Nutzbarkeit des Geräts unter Berücksichtigung der gesamten Situation. Werden derartige Methoden durch weitere Forschung und benutzerfreundliche Frameworks zum Regelfall bei ressourcenintensiven mobilen Anwendungen wie im Feld der *Augmented Reality* und anderen Simulationen, so wird die praktische Nutzbarkeit der Anwendungen stark erhöht und damit weitere Entwicklungen in diesem aussichtsreichen und vielfältigen Bereich der mobilen Simulationen vorangetrieben.

8.2 Ausblick

Abschließend werden Ansätze für zukünftige Arbeiten vorgestellt.

8.2.1 Einbettung

Das entwickelte Modell ist dafür geeignet, in ein Framework für qualitätsbewusste Anwendungen eingebunden zu werden. Die Schritte zur Aufstellung des Energiemodells können automatisiert werden. In Verbindung mit einem groben, API-basierten Echtzeit-Monitoring der noch zur Verfügung stehenden Energie kann der Energieverbrauch der Anwendungen über den Tag hinweg kontrolliert und angepasst werden. Über Ansätze wie maschinelles Lernen können zu einem gewissen Grad Erfahrungen über den täglichen Verlauf des Akkus gesammelt werden, um intelligente Entscheidungen über die Qualität der Berechnungen zu treffen.

8.2.2 Energieverbrauch bei Multicore-Architekturen

Mobile Geräte wie das Samsung Galaxy S7 verfügen über mehrere Kerne, welche unterschiedlich beansprucht und unter bestimmten Bedingungen übertaktet werden [Her16]. Hieraus

entstehen neue Anforderungen an ein Energiemodell, um den Energieverbrauch über alle CPU-Kerne hinweg akkurat abbilden zu können. Zusätzliche Herausforderungen ergeben sich durch den Einsatz paralleler Berechnungen und deren Auswirkungen auf den gesamten Energieverbrauch des Geräts.

8.2.3 Optimierung durch mehrere Qualitätsparameter

In der Praxis kann es vorkommen, dass nicht nur ein einzelner, sondern mehrere Parameter den Ressourcenbedarf und die Qualität einer Berechnung bestimmen. Es ergibt sich eine mehrdimensionale Problemstellung, wodurch sich neue Herausforderungen in Hinblick auf die Anzahl der Messungen, den Aufbau des Energiemodells sowie die Optimierung einer energiebeschränkten Berechnung ergeben.

Literaturverzeichnis

[are14] areamobile.de. *Samsung Galaxy Note 4 Datenblatt - Technische Daten*. 2014. URL: http://www.areamobile.de/handys/4108-samsung-galaxy-note-4/datenblatt (zitiert auf S. 29).

[BMM12] J. Bornholt, T. Mytkowicz, K. S. McKinley. „The model is not enough: Understanding energy consumption in mobile devices". In: *Power (watts)* 1.2 (2012), S. 3 (zitiert auf S. 19).

[Cat16] D. Catalano. *Catalano-Framework*. https://github.com/DiegoCatalano/Catalano-Framework. 2016 (zitiert auf S. 50).

[DD16] F. Dürr, C. Dibak. *rpi-powermeter*. https://github.com/duerrfk/rpi-powermeter. 2016 (zitiert auf S. 28).

[DDR15] C. Dibak, F. Dürr, K. Rothermel. „Numerical Analysis of Complex Physical Systems on Networked Mobile Devices". In: *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE. 2015, S. 280–288 (zitiert auf S. 11).

[DK14] C. Dibak, B. Koldehofe. „Towards Quality-aware Simulations on Mobile Devices". English. In: *Proceedings of the 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI) (Informatik 2014)*. Lecture Notes in Informatics (LNI). Gesellschaft für Informatik (GI), Sep. 2014 (zitiert auf S. 11).

[Dür15] F. Dürr. *Raspberry Pi Going Realtime with RT Preempt*. 2015. URL: http://www.frank-durr.de/?p=203 (zitiert auf S. 28).

[Eik16] R. Eikenberg. *Pokémon fangen ohne Akku-Frust*. 2016. URL: http://www.heise.de/ct/artikel/Pokemon-fangen-ohne-Akku-Frust-3279725.html (zitiert auf S. 10).

[Her16] E. Herrmann. *Samsung Galaxy S7 im Test: Das wasserdichte Gamer-Smartphone*. 2016. URL: https://www.androidpit.de/samsung-galaxy-s7-test (zitiert auf S. 72).

[Hil16] S. Hill. *30 Best Portable Battery Chargers*. 2016. URL: http://www.digitaltrends.com/mobile/best-portable-battery-chargers/ (zitiert auf S. 9).

[HLHG12] S. Hao, D. Li, W. G. Halfond, R. Govindan. „Estimating Android applications' CPU energy usage via bytecode profiling". In: *Proceedings of the First International Workshop on Green and Sustainable Software*. IEEE Press. 2012, S. 1–7 (zitiert auf S. 19, 20, 43).

[HLHG13] S. Hao, D. Li, W. G. Halfond, R. Govindan. „Estimating mobile application energy consumption using program analysis“. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, S. 92–101 (zitiert auf S. 19, 20, 43).

[Ked12] N. Kedem. *Six things to know about smartphone batteries*. 2012. URL: http://www.cnet.com/news/six-things-to-know-about-smartphone-batteries/ (zitiert auf S. 9).

[KLY+13] Y. Kwon, S. Lee, H. Yi, D. Kwon, S. Yang, B.-G. Chun, L. Huang, P. Maniatis, M. Naik, Y. Paek. „Mantis: automatic performance prediction for smartphone applications“. In: *Proceedings of the 2013 USENIX conference on Annual Technical Conference*. USENIX Association. 2013, S. 297–308 (zitiert auf S. 19).

[LHGH14] D. Li, S. Hao, J. Gui, W. G. Halfond. „An empirical study of the energy consumption of android applications“. In: *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2014, S. 121–130 (zitiert auf S. 43).

[PP16] P. Pandey, D. Pompili. „MobiDiC: Exploiting the untapped potential of mobile distributed computing via approximation“. In: *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2016, S. 1–9 (zitiert auf S. 19, 20).

[ZDM08] L. Zhai, S. Dong, H. Ma. „Recent methods and applications on image edge detection“. In: *Education Technology and Training, 2008. and 2008 International Workshop on Geoscience and Remote Sensing. ETT and GRS 2008. International Workshop on*. Bd. 1. IEEE. 2008, S. 332–335 (zitiert auf S. 50).

[Zil96] S. Zilberstein. „Using anytime algorithms in intelligent systems“. In: *AI magazine* 17.3 (1996), S. 73 (zitiert auf S. 15, 16).

Alle URLs wurden zuletzt am 06. 09. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift