Untersuchung und Modellierung des Energieverbrauchs von DVFS Prozessoren auf Basis von parallelen Berechnungen des wissenschaftlichen Rechnens

Analysing and Modelling of the Energy Consumption of DVFS Processors Based on Parallel Computations of Scientific Calculations

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von

Matthias Markus Stachowski
aus Knurow

1. Gutachter: Prof. Dr. Thomas Rauber
2. Gutachter: Prof. Dr. Jörg Keller

Tag der Einreichung: 13.01.2022
Tag des Kolloquiums: 04.05.2022

Vorwort

Die vorliegende Dissertation wurde während der Anstellung als wissenschaftlicher Mitarbeiter am Lehrstuhl für Parallele und Verteilte System (Angewandte Informatik II) an der Universität Bayreuth angefertigt. Die komplette Publikationsliste, die während dieser Zeit eingereicht und angenommen wurde, befindet sich im Literaturverzeichnis unter [1, 2, 3, 4, 5, 6]. Die Reihenfolge der Nennung der Autoren erfolgt in alphabetischer Reihenfolge.

Während der Tätigkeit in der Lehre wurden auch fachlich passende Abschlussarbeiten ausgegeben und angefertigt. Diese finden sich im Literaturverzeichnis unter [7, 8, 9, 10, 11, 12, 13, 14].

Für die viele Unterstützung bei dieser Dissertation und der Möglichkeit der Promotion möchte ich mich in erster Linie bei Herrn Professor Dr. Rauber bedanken. Ebenfalls herzlich bedanken möchte ich mich bei allen meinen Kollegen am Lehrstuhl für das jederzeit sehr angenehme und immer sehr freundliche Arbeitsklima.

Analysing and Modelling of the Energy Consumption of DVFS Processors Based on Parallel Computations of Scientific Calculations

Untersuchung und Modellierung des Energieverbrauchs von DVFS Prozessoren auf Basis von parallelen Berechnungen des wissenschaftlichen Rechnens

Matthias Stachowski

Abstract

Parallel computations on High-Performance-Computing systems are commonplace. Server farms are trending to more powerful hardware and overcome each other periodically with faster scientific calculations. It is also not foreseeable that this striving for more "flops" will come to an end soon. But the increase in performance ist limited by one important factor: Energy can not be scaled up by just putting more hardware together. So the big goal of this industry is not just to maximize performance, but to get maximum performance with lower energy costs. The aim is, therefore, to significantly increase energy efficiency. There are several ways to do so such as building better cooling systems on the hardware side or lowering the energy consumption on the software side e.g. using DVFS (dynamic voltage and frequency scaling) of a processor which gives the user the ability to change CPU frequencies at runtime. Additionally, since the introduction of the Intel Sandy-Bridge architecture, energy measurements of different components of the CPU withing short time periods are possible. This is the best prerequisite for application-dependent energy optimization of applications.
This thesis gives an insight into what can be done with DVFS and discusses different approaches to energy efficiency. Different applications respond differently to varying frequencies. Using the example of scientific calculations in the form of benchmarks, the energy behavior is investigated and analyzed.
DVFS not only affects the energy consumption but also changes the runtime: In general the runtime decreases linearly with a rising frequency, but the power consumption usually increases polynomially with the frequency. However, this polynomial increase of the power consumption is very hardware dependent. Thus, the energy consumption may behave very differently on different processors. For some applications, one may observe significant increase of the energy consumption when increasing the frequency, althoug h the runtime increases at the same time. Hence, it mostly comes down to a

compromise between runtime and energy consumption.
To measure these compromises, the introduction of new metrics which can capture both, the energy consumption and the performance of an application is needed. In this thesis, already established as well as newly defined metrics are introduced and evaluated for different benchmarks.
Another aspect of this thesis is the modelling of the energy consumption. The usage of an analytical model allows the prediction of the energy consumption of applications. Furthermore, optimal frequencies for the lowest energy consumption can be derived. Typically, the energy consumption has a U-shaped form when depicting its dependence on the frequency. Sometimes the U-shape is more pronounced and sometimes the curve is very flat, almost linear. This shape depends on several characteristics of the processor and the application. Different benchmarks and processors are used to evaluate the energy models.
Another way to reduce energy consumption is the usage of so-called autotuning software. An autotuner is usually a framework which implements different search strategies to find one or several optimal values of a target function depending on different parameters such as frequency or numer of hardware cores used. It is also possible to tune parameters to fit two targets e.g. runtime and energy consumption. This is called multi-objective autotuning and finds the best compromises between different targets. DVFS and autotuning are not limited to CPUs and can also be used with GPUs (Graphical Processing Units). With the help of a specially developed autotuning framework, the great influence of DVFS on graphics cards is shown. The currently very popular cryptocurrencies like Ethereum and Monero serve as use case. Energy efficiency plays a very important role here, because the lower the energy consumption at a constantly high computing power, the more revenue the will be obtained. It is also shown that it is possible to maintain a high hashrate quite automatically and at the same time to reduce energy consumption enormously.
In summary, this thesis is divided into the following five topics:

- Introduction and motivation to DVFS and energy
- Technical introduction to DVFS and energy efficiency
- Introducing new metrics for capturing the energy efficiency of applications
- Model-based optimisation of the energy consumption
- Energy-aware DVFS autotuning on GPUs for cryptomining.

Untersuchung und Modellierung des Energieverbrauchs von DVFS Prozessoren auf Basis von parallelen Berechnungen des wissenschaftlichen Rechnens

Analysing and Modelling of the Energy Consumption of DVFS Processors Based on Parallel Computations of Scientific Calculations

Matthias Stachowski

Zusammenfassung

Parallele Berechnungen auf Hochleistungsrechnern sind heutzutage nicht mehr wegzudenken. Rechenzentren übertreffen sich immer wieder mit noch stärkerer Hardware und noch höheren Geschwindigkeiten, um ein Problem zu lösen. Es ist auch nicht abzusehen, dass das Streben nach immer mehr "Flops" demnächst aufhört. Die Gier nach immer mehr Performance ist im Prinzip nur von einem Faktor beschränkt: Dem Energieverbrauch. Man kann die Rechenzentren theoretisch immer größer mit immer mehr Knoten bauen. Die Stromversorgung lässt sich jedoch nicht so einfach nach oben skalieren. Das große Ziel dieser Industrie heißt also nicht nur maximale Leistung, sondern maximale Leistung mit weniger Energiekosten. Es gilt dementsprechend die Energieeffizienz deutlich zu steigern. Dies geht unter anderem durch Anpassung der Hardware, durch effizientere Abwärmesysteme, oder aber durch eine dynamische und bedarfsorientierte Ausnutzung der Hardware. Eine in der Praxis anwendbare Technologie ist das DVFS (dynamic voltage and frequency scaling), also das dynamische Ändern der Frequenzen und Spannungen der Prozessoren zur Laufzeit. Zugleich ist es seit der Sandy-Bridge-Architektur der Intel Prozessoren möglich, den Energieverbrauch verschiedener Komponenten der CPU zur Laufzeit in sehr kurzen Intervallen auszulesen. Dies ist die beste Voraussetzung für anwendungsabhängige Energieoptimierung von Programmen.

Diese Arbeit gibt einen Einblick in das DVFS und geht noch darüber hinaus. Unterschiedliche Anwendungen reagieren unterschiedlich auf verschiedene Taktraten. Am Beispiel von wissenschaftlichen Berechnungen in Form von Benchmarks werden die Energieverläufe dargestellt und analysiert.

DVFS beinflusst nicht nur den Energieverbrauch, sondern auch die Laufzeit von An-

wendungen: Generell sinkt zwar die Laufzeit etwa gleichmäßig mit steigender Frequenz, die Leistungsaufnahme steigt jedoch in der Regel polynomisch. Der Verlauf der Leistungsaufnahme ist aber stark hardwareabhängig und somit sehr unterschiedlich auf den verschiedenen Prozessoren. So kann es passieren, dass zwar die Laufzeit eines Programms sinkt, aber der Energieverbrauch unverhältnismäßig stark steigt, weil die Leistungsaufnahme quadratisch nach oben verläuft, was die Energiebilanz insgesamt negativ beeinträchtigt. Es läuft also meistens auf einen Kompromiss aus Laufzeit und Energieverbrauch hinaus. Somit sind neue Metriken nötig, um Programme unter Berücksichtigung der Energie zu erfassen und neu zu bewerten. In dieser Arbeit werden eigene bereits etablierte als auch neu definierte Metriken eingeführt und in verschiedenen Benchmarks evaluiert.
Ein weiterer Aspekt ist die Modellierung des Energieverbrauchs von Programmen. Dies ermöglicht nicht nur Vorhersagen des Energieverbrauchs, sondern auch das Bestimmen des Optimums bezüglich der Energie. Typischerweise ergibt sich aus den Verläufen der Leistungsaufnahme und der Laufzeit ein U-förmiger Energieverlauf. Hardwareabhängig ist dieser mal stärker, manchmal auch gar nicht ausgeprägt. An verschiedenen Benchmarks und verschiedenen Prozessoren wird untersucht, inwieweit Modelle geeignet sind, den Energieverbrauch nachzubilden.
Eine andere Herangehensweise, den Energieverbrauch zu optimieren, ist der Einsatz sogenannter Autotuner. Ein Autotuner ist in der Regel ein Framework, welches über bestimmte Suchstrategien das Optimum findet, sei es Performance oder Energieverbrauch. Das automatische Tunen kann man auch auf mehrere Suchkriterien gleichzeitig anwenden und beispielsweise nach dem besten Kompromiss zwischen Laufzeit und Energieverbrauch suchen. Das ist nicht nur den Prozessoren vorenthalten, sondern kann genauso auf Berechnungen auf GPUs (Grafikkarten) angewendet werden. Mithilfe eines eigens entwickelten Autotuning-Frameworks wird gezeigt, wie groß der Einfluss von DVFS auf Grafikkarten ist. Als Anwendungsfall dienen die aktuell sehr populären Kryptowährungen wie Ethereum und Monero. Hier spielt die Energieeffizienz eine sehr wichtige Rolle, denn je geringer der Energieverbrauch bei einer konstant hohen Rechenleistung ist, desto mehr bleibt dem Kryptominer am Ende vom Reward übrig. Außerdem wird gezeigt, dass es möglich ist, ganz automatisch eine hohe Hashrate zu halten und zugleich den Energieverbrauch enorm zu senken.
Zusammengefasst gliedert sich die nachfolgende Arbeit in folgende fünf Themengebiete:

- Einleitung und Motivation in die Energieeffizienz auf Supercomputern
- Technische Einführung in DVFS und Energiemessung
- Metriken zur Bewertung der Energieeffizienz

- Modell-basierte Optimierung des Energieverbrauchs
- Energie-orientiertes DVFS Autotuning auf GPUs am Beispiel bei dem Minen von Kryptowährungen.

Inhaltsverzeichnis

Verzeichnisse **xiv**

Abbildungen xvi

Tabellen xx

Quellcode xxi

1 Einführung **1**

1.1 Green und Sustainable Computing 1

1.2 Von TOP500 zu GREEN500 3

1.3 Die Exascale Challenge 4

1.4 Fokus und Motivation dieser Arbeit 5

1.5 Gliederung der Arbeit 6

2 Technische Grundlagen **8**

2.1 Moore'sches Gesetz: Anfang und Ende 8

2.1.1 Rückblick 9

2.1.2 More Moore, More Than Moore, Beyond Moore 10

2.1.3 Clock Moore und Dark Silicon 11

2.2 Terminologie 12

2.2.1 DVFS - Dynamic Voltage and Frequency Scaling 12

2.2.2 RAPL - Running Average Power Limit 12

2.2.3 Dynamische Frequenzen und Pstates 13

2.3 Energiemessung unter Linux 18

2.3.1 LIKWID 18

2.3.2 PAPI 20

2.3.3 Sysfs 20

2.4 Benchmarks Suites 23

2.4.1 PARSEC Benchmark Suite 23

2.4.2 SPLASH-2 Benchmarksuite 24

3 Metriken zur Bewertung der Energieeffizienz 26
3.1 Einführung und Motivation . . . 26
3.1.1 Motivation . . . 26
3.1.2 Energie und Leistung . . . 28
3.2 Energie- und Performance-Metriken . . . 32
3.2.1 Speedup . . . 32
3.2.2 Runtime Reduction Factor . . . 36
3.2.3 Energy-Delay Product . . . 37
3.2.4 Energy Speedup . . . 38
3.2.5 Energy Reduction Factor . . . 40
3.2.6 Energy per Speedup . . . 41
3.2.7 Power Speedup . . . 42
3.2.8 Power Increase Factor . . . 44
3.2.9 Relative Power Increase Factor . . . 45
3.3 Auswertung der Ergebnisse . . . 46
3.4 Vergleich verschiedener Architekturen . . . 48
3.5 Verwandte Arbeiten . . . 51
3.6 Schlussfolgerungen . . . 51

4 Modellbasierte Optimierung des Energieverbrauchs paralleler Anwendungen 55
4.1 Einführung . . . 55
4.2 Leistungs- und Energiemodell für Frequenzskalierung . . . 56
4.3 Entwicklung eines Energiemodells für parallele Programme . . . 58
4.4 Entwicklung eines Energiemodells für das EDP . . . 59
4.5 Messungen und Beobachtungen der Leistung, Energie und EDP . . . 60
4.6 Energie- und Performanceanalyse . . . 63
4.6.1 Bestimmung der anwendungsspezifischen Leistungs-Parameter . . . 64
4.6.2 Anwendungsspezifische Modellierung des optimalen Skalierungsfaktors . . . 68
4.6.3 Anwendungsspezifische optimale EDP Frequenzen . . . 72
4.6.4 Anwendungsunabhängige optimale Frequenzen . . . 75
4.6.5 Anwendungsunabhängiges Optimum der EDP-Frequenzen . . . 76
4.7 Verwandte Arbeiten . . . 77
4.8 Zusammenfassung der Modellierung . . . 79

5 DVFS auf GPUs am Beispiel eines Autotuning Frameworks für energieeffizientes Kryptomining 81
5.1 Motivation und Aufbau des Kapitels . . . 82
5.2 Technischer Hintergrund . . . 83
5.2.1 Einführung in das Autotuning . . . 83
5.2.2 Einführung in die Blockchain . . . 84
5.2.3 Einblick in die Arbeitsweise von GPUs in diesem Kontext . . . 85
5.2.4 Energiemessung auf GPUs . . . 86
5.2.5 DVFS auf GPUs . . . 86
5.2.6 Hardware Setup . . . 87
5.3 Funktionsweise des Autotuning Frameworks . . . 88
5.3.1 Optimierungsverfahren . . . 88
5.3.2 Frequenzoptimierungsphasen . . . 92
5.3.3 Monitoring . . . 97
5.4 Evaluierung . . . 98
5.4.1 Verwendete Währungen . . . 98
5.4.2 Profiling . . . 99
5.4.3 Energieoptimum ETH-Ethash . . . 99
5.4.4 Energieoptimum XMR-Cryptonight . . . 101
5.4.5 Energieoptimum ZEC-Equihash . . . 104
5.4.6 Verwendung eines Power-Limits . . . 109
5.4.7 Frequenzoptimierung durch Suchstrategien . . . 110
5.4.8 Performance Hill Climbing . . . 111
5.4.9 Performance Simulated Annealing . . . 111
5.4.10 Performance Nelder-Mead . . . 113
5.4.11 Optimierung unter Nebenbedingungen . . . 115
5.4.12 Monitoring . . . 119
5.5 Zusammenfassung . . . 120
5.6 Verwandte Arbeiten . . . 123
5.7 Ausblick . . . 124

6 Schluss 125
6.1 Zusammenfassung und Fazit . . . 125

A Systeme 127
A.1 Skylake . . . 128
A.2 Haswell . . . 128

A.3 Kaby Lake 129
A.4 Skylake-W 129
A.5 Odroid XU-4 130

Literaturverzeichnis **131**
Eigene Publikationen 132
Betreute Abschlussarbeiten 132
Fremdquellen 138
Onlinequellen 141

Abbildungsverzeichnis

2.1 Veranschaulichung der Top500. Abgebildet sind jeweils die Summe aller Plätze (Sum), erster Platz (#1), sowie letzter Platz (#500) zwischen 1993 und 2019 [92]. 11

2.2 Verfügbare Power Domänen auf 2 Sockets. Zum Ausrechnen der Uncore Domäne müssen PP0 und PP1 von PKG subtrahiert werden. [70] . . . 14

2.3 Ordner/Datei-Struktur des Intel-Rapl-Interfaces im Userspace am Beispiel eines Intel Haswell i7-4770 mit 4 verfügbaren Domänen unter der Linux-Distribution Suse 15.1. 22

5.1 Nelder-Mead: Änderung des Simplex während einer Iteration (links) und Darstellung des Iterationsverlaufs bei der Optimierung einer zweidimensionalen Zielfunktion (rechts) (aus [68] und [79]) 93

5.2 Schematischer Ablauf des Autotunings im implementieren Framework (aus [7]) 95

5.3 Auslastung der VRAM-Speicherbandbreite (links) und ausgeführte IPC (rechts) beim Mining von Ethereum (ETH, ethminer), Monero, (XMR, xmrstak) und ZCash (ZEC, excavator) auf einer Titan X (Pascal) mit Standardfrequenzen (aus [5]). 100

5.4 Ethereum (ETH): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte) 102

5.5 Ethereum (ETH): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte) 103

5.6 Monero (XMR): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte) (aus [7]) 105

5.7 Monero (XMR): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7]) . . . 106
5.8 ZCash (ZEC): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte) (aus [7]) 107
5.9 ZCash (ZEC): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7]) . . . 108
5.10 Hill Climbing: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7]) 112
5.11 Simulated Annealing: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7]) 114
5.12 Nelder-Mead: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7]) 116
5.13 Optimierung unter Nebenbedingung von ZEC: Verlauf der Frequenzoptimierungen mit Hill Climbing (oben), Simulated Annealing (mitte) und Nelder-Mead (unten) mit einer minimal einzuhaltenden Hashrate (95 % max. Hashrate) auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) 118
5.14 Kalkulierter Mining-Profit der verschiedener Kryptowährungen auf den einzelnen Grafikkarten eines Systems während des Monitorings (aus [7]) 120
5.15 Kalkulierter Mining-Profit (oben), Miningerträge (unten links) und Energiekosten (unten rechts) der jeweils profitabelsten Kryptowährung auf allen Grafikkarten eines Systems während des Monitorings (aus [7]) . . 121

Tabellenverzeichnis

2.1 Übersicht und kurze Beschreibung der Sammlung an likwid Unterprogrammen. 19

3.1 Symbole und Einheiten aus Kapitel 3.1.2. 30

3.2 Performance-, Leistungs- und Energie-Metriken 32

3.3 Runtime Reduction R Factor für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 37

3.4 EDP für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 39

3.5 EDDP für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 39

3.6 ES für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 40

3.7 ER für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 41

3.8 EPS für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 43

3.9 RPI für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor 46

3.10 Auswertung verschiedener PARSEC Benchmarks auf einem Intel i7-4770 Haswell Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads 48

3.11 Auswertung verschiedener PARSEC Benchmarks auf einem Intel i7-6700 Skylake Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads 48

3.12 Auswertung verschiedener SPLASH-2 Benchmarks auf einem Intel i7-4770 Haswell Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads 49
3.13 Auswertung verschiedener SPLASH-2 Benchmarks auf einem Intel i7-6700 Skylake Prozessor:Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads 49
3.14 P für Blackscholes auf XU4 für verschiedene Frequenzen und Threads. 52
3.15 RPI für Blackscholes auf XU4 für verschiedene Frequenzen und Threads. 52
3.16 PI für Blackscholes auf XU4 für verschiedene Frequenzen und Threads. 52
3.17 EPS für Blackscholes auf XU4 für verschiedene Frequenzen und Threads. 52

4.1 Zusammenfassung der Notation der Kapitel 4.2 bis 4.4. 60
4.2 Leisungsaufnahme in Watt des PARSEC Benchmarks Freqmine ausgeführt auf einem Intel i7-6700 Skylake Prozessors für verschiedene Frequenzen und Threads. 61
4.3 EDP für den PARSEC Freqmine Benchmark auf einem i7 Skylake. . . . 63
4.4 Gemessene und modellierte Leistungsaufnahme nach Gleichung (4.3) der SPLASH-2 Benchmarks auf einem Intel i7-4700 Haswell Prozessor. . . 65
4.5 Gemessene und modellierte Leistungsaufnahme nach Gleichung (4.3) der SPLASH-2 Benchmarks auf einem Intel i7-6700 Skylake Prozessor. . . 66
4.6 Evaluation verschiedener **PARSEC** Benchmarks auf einem i7-4770 Haswell Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. **E-Diff** und **E-Diff-glob** zeigen prozentuale Unterschiede in der Energie für anwendungsspezifische und anwendungsunabhängige Modelierung. 69
4.7 Evaluation verschiedener **PARSEC** Benchmarks auf einem i7-6700 Skylake Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. **E-Diff** und **E-Diff-glob** zeigen prozentuale Unterschiede in der Energie für anwendungsspezifische und anwendungsunabhängige Modelierung. 69
4.8 Evaluation der **SPLASH-2** Benchmarks auf einem Intel i7-4700 Haswell Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. 70

4.9 Evaluation der **SPLASH-2** Benchmarks auf einem Intel i7-6700 Skylake Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. 70
4.10 Evaluation und Modellierung des EDP für **PARSEC** Benchmarks auf einem Intel i7-4770 Haswell Prozessor. 73
4.11 Evaluation und Modellierung des EDP für **PARSEC** Benchmarks auf Intel i7-6700 Skylake Prozessor. 73
4.12 Evaluation und Modellierung des EDP für **SPLASH-2** Benchmarks auf einem Intel i7-4770 Haswell Prozessor. 74
4.13 Evaluation und Modellierung des EDP für **SPLASH-2** Benchmarks auf Intel i7-6700 Skylake Prozessor. 74
4.14 Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob} und korrespondierende Frequenzen für PARSEC Benchmarks. 75
4.15 Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob} und korrespondierende Frequenzen für SPLASH-2 Benchmarks. 75
4.16 Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob}^{EDP} und die entsprechenden Frequenzen für die PARSEC Benchmarks. 76
4.17 Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob}^{EDP} und die entsprechenden Frequenzen für die SPLASH-2 benchmarks. 76

5.1 Auflistung der verwendeten Grafikkarten, die zur Auswertung genutzt wurden 88
5.2 ETH: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7]) 101
5.3 XMR: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7]) 104
5.4 ZEC: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7]) 109
5.5 Gesetzte Core-Frequenz, tatsächlich gemessene Leistungsaufnahme und erzielte Hashrate auf der Titan X bei einem Power-Limit von 125W und auf der Quadro P4000 bei einem Power-Limit von 60W für verschiedene Währungen (aus [7]) 110
5.6 Hill Climbing: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7]) 113

5.7 Simulated Annealing: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7]) 115
5.8 Nelder-Mead: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7]) 117
5.9 Ergebnis der Suche in Hashes pro Joule inkl. der dazugehörigen Frequenzen und der Anzahl an Funktionsauswertungen der Optimierung von ZEC mit den verschiedenen Suchalgorithmen unter der Nebenbedingung einer minimal einzuhaltenden Hashrate von 95 % (aus [7]) 119

Quellcodeverzeichnis

2.1 Übersicht der verfügbaren Informationen zum Frequenzmanagement eines AMD EPYC 7551 15
2.2 Einstellbare Frequenzen Haswell acpi-Treiber. 16
2.3 Einstellbare Frequenzen Haswell Intel Pstate Treiber. 16
2.4 Ausgabe der aktuell eingestellten Core-Frequenzen und verfügbaren Uncore-Frequenzen mit likwid. 17
5.1 Struktur für die unterstützten Frequenzen auf den Grafikkarten (device_clock_info) (aus [91]) 87
5.2 Struktur zur Abspeicherung einer Messung (measurement) (aus [91]) . . 94

1 | Einführung

Performance zu geringeren Energiekosten: So lautet immer wieder das Motto der Chip-Hersteller. Egal ob es sich um Desktop- oder ARM-Prozessoren im Smartphone-Bereich handelt - der Energieverbrauch muss sinken. Besonders bei letzterem rückt die Steigerung der Energieeffizienz immer stärker in den Mittelpunkt. Die Zeiten, in denen nur der höhere Takt und eine hohe Gigahertz-Zahl für steigende Verkaufszahlen gesorgt haben, sind vorbei. Der Trend nach immer mehr Transistoren und Rechenkernen neigt sich mit der *Dark-Silicon-Ära* [42] dem Ende hin. Stattdessen strebt die Entwicklung nach Exascale-Systemen, die jedoch durch die alleinige Steigerung der Anzahl an Prozessoren und Beschleunigern wie Grafikkarten nicht zu bewältigen ist. Durch den hohen Energieverbrauch wäre eine vollständige Energieversorgung nicht garantiert und gleichzeitig würden die Stromkosten immens ansteigen und die Rechenzentren in wirtschaftliche Bedrängnis führen. Zudem verhalten sich verschiedene Prozessoren im Hinblick auf ihren Energieverbrauch und ihre Leistungsaufnahme sehr unterschiedlich. Schaut man sich die Leistungsaufnahme beispielsweise unter Volllast an einem Intel Core i7-7700 Kaby Lake Prozessor (Anhang A.3) bei verschiedenen Frequenzen an, einmal als Desktop-CPU und einmal als Notebook-CPU, so erkennt man, dass die Notebook-Variante steiler ansteigt als die Desktop-Variante (Abbildung 1.1). Die Desktop-Variante bietet zudem noch eine deutlich größere Spanne an Frequenzen, die man einstellen kann. Bei 2.2 GHz haben beide Kaby Lakes einen sehr ähnlichen Verbrauch, der etwa um die 15 Watt liegt. Ein i7-6700 Skylake-Prozessor (Anhang A.1) verläuft in etwa zwischen den beiden Kaby Lake Prozessoren. Dieser verbraucht jedoch bei sehr niedrigen Frequenzen deutlich weniger Strom.
In genau diese und weitere Aspekte führt dieses Kapitel ein.

1.1 Green und Sustainable Computing

Die Welt um uns herum scheint immer grüner zu werden. Dies ist jedoch wenig in einer Zunahme von Wiesen und Wäldern begründet, denn in Zeiten der Urbanisierung

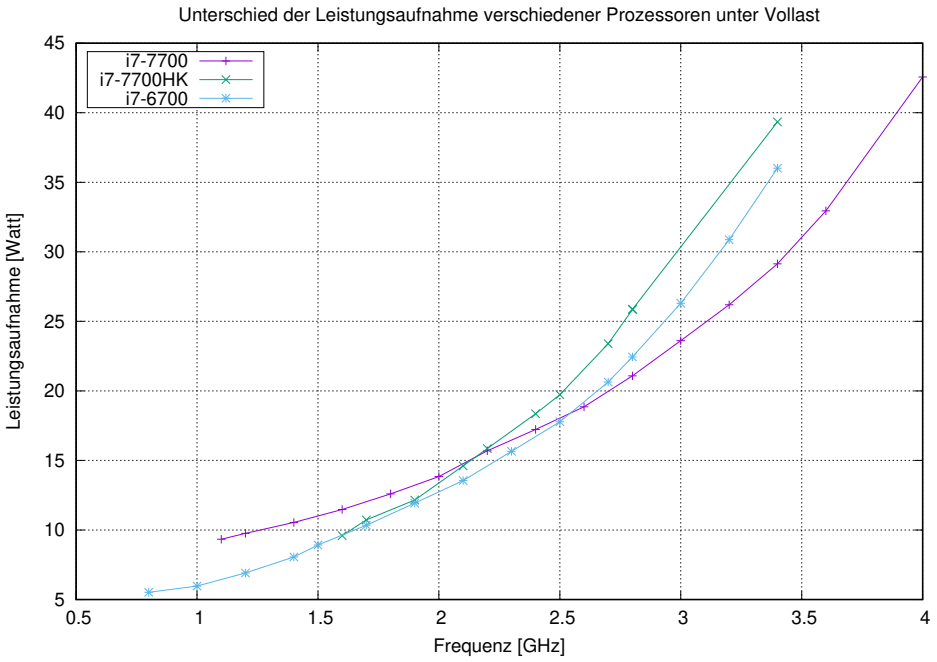

Abb. 1.1: Vergleich der Leistungsaufnahme eines Intel i7-7700 Kaby Lake Porzessors, eines i7-7700HK (Notebook Variante) und eines i7-6700 Skylake Prozessors unter Volllast (aus [13]).

und Globalisierung sieht die Realität eher gegenteilig aus. Vielmehr scheint es am gegenwärtigen Gebrauch des Begriffes *grün* zu liegen, der ein sehr breites Spektrum umfasst. Wurde die Farbe Grün in den 1970er Jahren noch mit Greenpeace und in den 1980er Jahren mit dem Einzug der Grünen in den Bundestag[1] in Verbindung gebracht, scheint seit dem letzten Jahrzehnt nahezu jedes Produkt im Supermarkt grün zu sein. Grün ist zu einer Marketing-Strategie geworden. Es ist populär und gilt als Synonym für *gut für die Umwelt* oder schlicht nachhaltig. In dieser Dissertation geht es jedoch weder um Greenpeace noch um den Regenwald, sondern um das *Green Computing*, oft auch als *Sustainable Computing* bezeichnet.

Mit dem Begriff Nachhaltigkeit (engl. Sustainability) verhält es sich ähnlich: Assoziierte man ihn ursprünglich mit der Landwirtschaft, Energiegewinnung und Entwicklungshilfe, so findet man ihn heute in regelrecht jedem Bereich: Ein Lebensstil ohne ausschweifenden Ressourcenverbrauch ist nicht mehr üblich und auch wirtschaftlich nicht profitabel. Also müssen die Ressourcen nachhaltig produziert werden - oder zumindest auf den ersten Blick so scheinen [78].

Auch in der Informatik haben sich die Begriffe nachhaltig und grün etabliert. Der Strom-

[1] `https://de.wikipedia.org/wiki/Bündnis_90/Die_Grünen`

verbrauch elektronischer Geräte steigt durch zunehmend komplexere Berechnungen, seien diese lokal auf dem eigenen Smartphone/PC oder durch Simulationen auf einem Supercomputer. Rechnet man die Anzahl der im Umlauf befindlichen Smartphones in der EU zusammen, kommt man laut Studie auf einen Verbrauch von 1.330.062.550 Kilowattstunden pro Jahr [90]. Auch wenn heutzutage Strom auch nachhaltig produziert wird und die Rechenzentren die Abwärme der Rechner anderweitig nutzen, z.B. zur Produktion von Algen die man dann wiederum verzehren kann[2], sollte dies kein Grund dagegen sein, den Fokus beim Design darauf zu legen, bei steigender Berechnungsleistung gleichzeitig den Stromverbrauch zu senken. Dies geht sowohl auf Hardware- als auch auf Softwareebene. Es wird in der Praxis versucht, beides zu optimieren. Diese Dissertation setzt den Fokus auf die Softwareebene.
Dass Green- und Sustainable Computing nicht nur eine Randerscheinung sind, zeigen auch zahlreiche neue Konferenzen und Journals, die in den letzten Jahren entstanden sind. Dazu zählen unter anderem [77, 75, 55].

1.2 Von TOP500 zu GREEN500

Lange Zeit haben sich Institutionen und Hersteller in der Liste der TOP500 darin gemessen, wer den schnellsten Supercomputer betreibt. Diese Liste ist ein auf den High-Perfomance LINPACK-Bechmark gewertetes und seit 25 Jahren geführtes Verzeichnis der schnellsten Rechensysteme der Welt. Der Rechner mit dem höchsten Score bildet die Spitze. Stand November 2020 dominiert diese der *Supercomputer Fugaku*. Er steht im RIKEN Center for Computational Science in Japan und erreicht mit seinen 7,630,848 Kernen eine theoretische Spitzenleistung (Rpeak) von 537,212 TFlop/s. Die getestete maximale Spitzenleistung ist nur knapp dahinter mit 442,010 TFlop/s. Mit einer Leistungsaufnahme von 29,899 kW ist diese jedoch schon fast dreimal so hoch wie der nächstplatzierte Superrechner.
Platz zwei geht an *Summit*, welcher in den USA im Department of Energy's (DOE) im Oak Ridge National Laboratory (ORNL) zum Einsatz kommt. Hersteller hier ist IBM. Mit knapp über 2,4 Mrd Rechnerkernen verbraucht das System während des Benchmarks über 10 Megawatt und erreicht einen Score von 148,600 TFlop/s im Test. Seit 2008 wird neben der Perfomance auch der Energieverbrauch bzw. die Leistungsaufnahme protokolliert.
Gleichzeitig zu der TOP500 wird die GREEN500 geführt: Auch hier wird der High-

[2] `https://www.heise.de/news/Deutscher-Windcloud-Server-CO2-positives-Vorzeigeprojekt-mit-Algenfa`
`html`

Perfomance LINPACK als Benchmark verwendet. Jedoch zählt zur Bewertung nicht allein die Anzahl an Rechenoperationen pro Sekunde. Auch spielt der Energieverbrauch eine Rolle. Als Einheit gilt hier die Performance pro Watt, typischerweise in *Flops/watt*. Summit landete hier im November 2019 mit 14.719 GFlops/watts noch auf Platz fünf der GREEN500. In der November 2020 Liste schaffte es Summit nicht mehr in die Top 10. Platz eins erreicht hier Nvidia in den USA mit ihrem *NVIDIA DGX SuperPOD* und einer Power Effizienz von 26.195 GFlops/watts. Dieser Supercomputer ist auf einer AMD EPYC Plattform mit je 64 Kernen aufgebaut, welche nur mit 2,25 GHz takten. Das lässt bereits darauf schließen, dass das Energieoptimum ein Kompromiss aus Laufzeit und Prozessortakt ist, welche sich gegenseitig stark beeinflussen. Der Takt ist heutzutage nicht mehr fix auf einen Wert festgelegt, sondern kann während der Laufzeit dynamisch geändert werden - sei es vom Benutzer oder Betriebssystem. Besitzt ein Prozessor diese Eigenschaft, handelt es sich um einen *dynamic voltage and frequency scaling*, kurz *DVFS* Prozessor. Bei der TOP500 und GREEN500 Liste sollte beachtet werden, dass sich die einzelnen Systeme auch mit der Zeit ändern. So war beispielsweise der Summit im November 2019 mit 14.668 GFlops/watts gelistet, also minimal weniger als ein Jahr später. Die aktuellsten Daten und die kompletten Listen finden sich beide in [93, 74].

1.3 Die Exascale Challenge

Die Exascale Challenge besteht darin, einen Exaflop pro Sekunde beim LINPACK-Benchmark zu übertreffen. Dabei sollen 20 bis 40 Megawatt nicht überschritten werden, wie es das Department of Energy (USA) vorgibt. Das sind eine Trillion Fließkommaoperationen pro Sekunde. Wenn man die vorgegebene Leistungsaufnahme von 20 Megawatt einbezieht, wäre es über 50 GFlops/watt. Im Vergleich dazu schafft der Supercomputer *Summit* über 143 Peta-Flops pro Sekunde und das heute effizienteste Rechnersystem *NVIDIA DGX SuperPOD* etwas über 26 GFlops/watt.
Die Anzahl der Rechenkerne kann man problemlos linear steigern, nicht jedoch den Stromverbrauch des Systems, denn dieser ist mit hohen Kosten verbunden. In den USA kostet ein Megawatt pro Jahr eine Million Dollar. Die Kosten in Deutschland wären deutlich höher. Geht man von einem Preis von 30 Cent (Stand 2021) pro Kilowattstunde aus, dann kommt man bei einem Megawatt auf Stromkosten in Höhe von 2,628,000 Euro. Doch auch die Leistung der Kraftwerke ist beschränkt. Typischerweise produziert ein Kraftwerk zwischen 200 und 300 Megawatt und versorgt damit mehrere größere Städte in der Region. Zusätzliche 20 Megawatt würden einen teuren Ausbau

der Stromnetz-Infrastruktur mit sich ziehen.
Der Energieverbrauch ist zwar der primäre Flaschenhals auf dem Weg zu Exascale, nicht aber der einzige: Die Speicherbandbreite korrelliert nicht mit den steigenden Rechenoperationen, was beispielsweise beim Rendering von Objekten zu Einbußen führt. Um den Energieverbrauch zu senken, müssen die Taktraten abnehmen. Somit muss die Anzahl an Recheneinheiten auf einem einzelnen Chip wiederum erhöht werden. Viele (konkurrierende) parallele Prozesse sind die Folge. Netzwerkkosten, sowohl im Energieverbrauch, als auch bei der Performance, verbessern sich nicht so schnell wie die Prozessorleistung. Somit ist es den Algorithmen überlassen, den Datenaustausch anstatt die Anzahl an Berechnungen gering zu halten. Auch die Compiler müssen angepasst werden.
Die Zuverlässigkeit und Fehleranfälligkeit werden bei Exascale-Systemen einen kritischen Bereich erreichen: Fehler aufgrund von Versagen der Komponenten und der Fertigungsvariabilität spielen eine viel größere Rolle als bei Petascale-Rechnern heute. Das Kostenmodel für Berechnungen ändert sich von kostspieligen Flops mit nahezu vernachlässigbarem Datenaustausch, bis hin zu frei verfügbaren Flops zu teurem Datenaustausch [94, 59].

1.4 Fokus und Motivation dieser Arbeit

Diese Arbeit konzentriert sich auf softwareseitige Optimierung des Energievebrauch von frequenzvariablen Multi-Prozessoren (CPUs) paralleler Programme. Anwendungen laufen heute nicht mehr sequenziell ab, sondern sind parallel geschrieben, um die Ressourcen eines Prozessors optimal auszunutzen. Dabei skalieren Laufzeit und Energieverbrauch oft nicht proportional zueinander, was den Schwerpunkt auf die Untersuchung und Modellierung deren Energieverbrauchs legt. Haupteinsatzgebiet sind x86 Prozessoren von Intel, da diese in dem Zeitraum der Erstellung dieser Arbeit die meisten Modifikationen zuließen und für Testreihen in einer großen Anzahl verschiedener Modelle an der Universität Bayreuth vorhanden waren. Nichtsdestotrotz werden auch andere Prozessoren wie Grafikkarten (GPGPUS) und mobile Prozessoren (ARM) in Unterkapiteln betrachtet.
Ziel ist es außerdem, einen genauen Überblick über dieses Themengebiet zu geben und aufzuzeigen, was gegenwärtig von softwareseitigen Optimierungen und Modellen eingesetzt wird. Weiter ist Autotuning ein großes Thema, welches sich sehr gut auf die Optimierung des Energieverbrauchs anwenden lässt. Automatisches Tuning einer

Software bedeutet in erster Linie eine selbstadaptive Anpassung von Software über diverse Parameter an die ausführende Hardware. Multi-kriterielles Autotuning führt dies noch eine oder mehrere Dimensionen weiter: Hier wird versucht, den besten Kompromiss aus mehreren Tuning-Zielen zu finden, beispielsweise den geringsten Energieverbrauch bei einer noch akzeptablen Laufzeit.

1.5 Gliederung der Arbeit

Die eingereichte Arbeit gliedert sich in 5 Hauptkapitel. Nachfolgende Liste fasst kurz den Inhalt der einzelnen Kapitel zusammen:

- Das erste Kapitel führte in die Thematik ein. Es soll eine Motivation und gleichzeitig die Essenz aufzeigen, wieso das Thema Energieeffizienz im HPC Bereich notwendig ist.
- Kapitel 2 führt die wichtigsten technischen Grundlagen ein. Angefangen beim Fortschritt in der Entwicklung von Prozessoren in den letzten Jahren wird hier deutlich gemacht, dass das Moorsche Gesetz - die Verdopplung der Rechenleitung alle 18 bis 24 Monate und die Steigerung der Anzahl der Transistoren auf der Chipfläche kaum zu halten ist. Weiter werden Begriffe aufgeführt und erklärt, die zum weiteren Verständnis beitragen. Abgeschlossen wird dieses Kapitel durch eine Übersicht der Möglichkeiten, zur Laufzeit den Energieverbrauch zu messen.
- Kapitel 3 führt etablierte und neue Metriken zur Bewertung der Energieeffizienz ein. Das Kapitel startet mit dem Unterschied zwischen der Energie und der Leistung (Power), da diese beiden Begriffe oft fälschlicherweise als Synonyme verwendet werden. Sowohl bereits etablierte Metriken wie das Energy-Delay-Produkt (EDP) werden hier näher ausgeführt als auch neue Metriken wie der Energie-Speedup (ES) eingeführt.
- Kapitel 4 konzentriert sich auf Energiemodelle. Dabei werden physikalische Zusammenhänge der CPU Architektur zusammengefasst und darauf ein Modell gebildet. Dieses Modell erlaubt es, den Verlauf des Energieverbrauchs für verschiedene Threadzahlen und unterschiedliche Frequenzen abzubilden. Dies ermöglicht es wiederum, den Energieverbrauch von Anwendungen vorherzusagen und das Energieoptimum abzuleiten.
- Kapitel 5 beschäftigt sich mit DVFS auf Grafikkarten. Auch Grafikkarten zählen zu den Prozessoren, die jedoch anders fokussiert sind als CPUs. Zudem gibt es hier einen Einblick in das Autotuning - dem automatischen Tunen der Frequenzen

zur Laufzeit über ein eigens entwickeltes Framework. Als Anwendungsfall für die Evaluierung werden hier die sehr populären Kryptowährungen verwendet und über DVFS in der Energieeffizienz optimiert.

- Kapitel 6 fasst die gesamte Arbeit zusammen.

2 | Technische Grundlagen

Auf der Softwareseite hat sich durch die Einführung der Sandy Bridge Architektur bei den Intel Prozessoren viel getan. Diese Prozessorreihe ermöglichte es erstmals, während der Laufzeit den aktuellen Energieverbrauch einer CPU ohne Gebrauch von Fremdhardware auszulesen. Realisiert wird die Messung über das modellgetriebene RAPL (Running Average Power Limit) Interface, welches die modellspezifischen (Perfomance-)Register (MSR) ausliest [51]. Auf diese und weiteren wichtigen Begrifflichkeiten wird der Fokus dieses Kapitels gelegt. Unter anderem werden das dynamic voltage and frequency scaling, DVFS, im Zusammenhang mit Intels Power-Management States *P-States* sowie die Unterscheidung der CPU-Frequenzen (Core- und Uncorefrequenz) beleuchtet.
Zu Beginn widmet sich das Kapitel der grundsätzlichen Frage, weshalb das Frequency-Scaling von solch großer Bedeutung ist, um Systeme energieeffizienter zu betreiben. Dabei spielt das Moorsche Gesetz eine essenzielle Rolle. Weiter geht es in dem Kapitel mit Informationen und Benutzung der gängigsten Tools, die es ermöglichen, den Energieverbrauch zur Laufzeit des Betriebssystems auszulesen, wie etwa Papi oder Likwid. Abschließen wird das Kapitel mit einer Übersicht über die eingesetzten Benchmarksuites PARSEC und SPLASH-2.

2.1 Moore'sches Gesetz: Anfang und Ende

Das Moore'sche Gesetz besagt, dass sich die Anzahl der Transistoren auf einem integrierten Schaltkreis (IC) alle 18 bis 24 Monate verdoppelt. So zumindest wird es gemeinhin bekannt. Mehr als 50 Jahre lang hat sich die Halbleiter Industrie an dieser Gesetzmäßigkeit orientiert. Die Erhöhung der Integrationsdichte durch das Verkleinern der Transistoren führte in den Anfangsphasen dazu, dass der jeweilige Energieverbrauch bei steigender Geschwindigkeit sank. Diese Regelmäßigkeit wird auch Dennard Scaling genannt. Die Rechenleistung sollte allerdings nicht nur im Hinblick auf die Anzahl der Transistoren pro Fläche bestimmt werden. Vielmehr sollte auch die Zeit betrachtet werden: Die Zeit in der Mikroelektronik ist umgekehrt proportional (Frequenz = 1/Zeit)

mit der Taktfrequenz (clock frequency). Dieses Kapitel zeigt auf, inwiefern das Moore'sche Gesetz in seiner bisherigen Form kaum weiter skaliert werden kann und eine dynamische Anpassung der Taktfrequenz nötig ist.
Einen genauen Überblick über das Moore'sche Gesetz, aus dem Auszüge für dieses Kapitel verwendet wurden, ist in [63] zu finden.

2.1.1 Rückblick

Das Moore'sche Gesetz gilt als Symbol technologischen Fortschritts. 1965 beobachtete Gordon Moore, dass sich die Anzahl an Komponenten pro Schaltkreis jedes Jahr verdoppelte. 1975 wurde die Vorhersage auf zwei Jahre abgeändert. Für die Kombination aus Rechenleistung und Transistorzahl wurde das 18-Monats-Intervall eingeführt. Über 5 Jahrzehnte hinweg hat sich diese Annahme gehalten. In der Halbleiterindustrie und in der Forschung wurde diese Zeitperiode für weitere Planungen verwendet. In der Vergangenheit gab es drei evolutionäre Stufen des Moore'schen Gesetzes: Version 1.0, von Anfang der 70er bis Ende der 80er Jahre verwendet, beschränkte sich auf die Dichte an Transistoren eines einzelnen Prozessors (CPU). Version 2.0 hob sich Mitte der 1990er Jahre heraus. Eine Steigerung der Rechenleistung wurde durch höhere Taktraten erreicht. Jedoch war das Limit bereits um die Jahrtausendwende ausgereizt. Der zusätzlichen Abwärme konnten die Chips nicht mehr standhalten. Version 3.0 integrierte weitere Funktionen (SoC) auf die Prozessoren. Seien es mehrere Prozessorkerne, integrierte Grafikkarten (GPUs) oder eingebaute Wifi-Schnittstellen.
Heutzutage werden Chips mit 21 Milliarden Transistoren produziert, wie Nvidias Volta GV100 Grafikkarte. Problematisch wird es, wenn der Durchmesser der Verbindungsleitungen und das Gatter zu klein werden. In diesem Fall fangen die Elektronen an zu streuen und können einen Kurzschluss produzieren. Die fundamentale physische Größe ist der Abstand zwischen zwei Siliciumatomen und liegt bei etwa 0,5 nm. Eine Zelle aus neun Siliciumkristall-Einheiten ist deshalb nicht weniger als 4,5 nm breit. Mit neuen Technologien können voraussichtlich 5-7 nm erreicht werden. Dieses Limit wurde bereits erreicht: AMD war der Vorreiter bei der 7nm Produktion auf Grafikkarten[1] und erst 2020 hat Apple mit der M1 Reihe ein Notebook, basierend auf 5nm Fertigung im Portfolio[2] zum Verkauf für Endkunden.
Zurück zu den neunziger Jahren hat die Halbleiterindustrie ihre Roadmap alle zwei Jahre aktualisiert, um die industrieweiten Anstrengungen diesbezüglich aufzuzeigen. Seit 2017 gab es keine Aktualisierung mehr. Dies ist ein klares Zeichen, dass das konventionelle

[1] `https://www.amd.com/de/technologies/vega7nm`
[2] `https://de.wikipedia.org/wiki/Apple_M1`

Hochskalieren der CMOS Transistoren dem Ende nah ist.

2.1.2 More Moore, More Than Moore, Beyond Moore

Bei dem Moore'schen Gesetz geht es aber nicht primär um Halbleiter, Computer, Performance oder Elektronik. Es handelt sich vielmehr um ökonomische Zusammenhänge. Das exponentielle Wachstum der Anzahl an Transistoren kann nicht beliebig lange fortgeführt werden. Aus Sicht der Konsumenten kann man unter dem Gesetz jedoch auch den *Nutzwert* verstehen. Dieser verdoppelt sich alle zwei Jahre. In diesem Kontext sind drei Ansätze zu nennen, wie man aus den räumlichen Verhältnissen noch Mehrwert erhalten kann: *More Moore*, *More than Moore* und *Beyond Moore*.
Der More than Moore Ansatz impliziert die Idee der kontinuierlichen Verkleinerung der Transistoren, um somit immer weniger Raum und Material zu benötigen. Während das Moore'sche Gesetz besagt, dass immer mehr Transistoren auf den Chips untergebracht werden können, versichert das Dennard Scaling, dass von Generation zu Generation die jeweiligen Transistoren einen zunehmend geringeren Energieverbrauch haben und kühler bleiben.
Der More than Moore Ansatz beruht auf dem Entschluss, sich nicht nur auf die Erhöhung der Rechenleistung zu konzentrieren, sondern das Ganze von der anderen Seite zu betrachten. Anstatt den Anwendungsfall zu verbessern, indem man den Chip weiterentwickelt, wird von dem Anwendungsfall selbst ausgegangen und geprüft, welche Arten von Architektur benötigt werden, um diesen zu unterstützen. Hier kommt neben der Rechenleistung auch die Effizienz des gesamten Systems mit einem wesentlich höheren Stellenwert zum Tragen. Es werden mehr Funktionen integriert und dabei gleichzeitig die Systemkosten reduziert.
Beyond Moore geht noch einen Schritt weiter. Anstatt wie bisher Leiterbahnen in 2-dimensionale Platinen zu ätzen, verfolgt dieser einen 3-dimensionalen Ansatz, indem mehrere Schichten an Silizium gestapelt werden. Aufgrund der Abwärme erfolgt die praktische Realisierung bisher jedoch lediglich bei Speicher-Chips. Ein anderer Ansatz ist der Verzicht auf CMOS Transistoren und die Verwendung von Alternativen, wie beispielsweise Feldeffekttransistoren, Kohlenstoffnanoröhren oder Molekulare Schalter. Anstatt sich auf binäre Operationen zu beschränken, bieten diese Technologien Funktionen, die darüber hinaus gehen.
Dass das Moore'sche Gesetz heute stagniert, zeichnet sich auch schon im Wachstum der Top500 ab, siehe Abbildung 2.1. Bisher verhielt sich das Wachstum der Performance von Supercomputern annähernd exponentiell. Ab etwa 2004-2008 war der Anstieg sogar leicht über der Projektion. Die Weiterentwicklung liegt jedoch seit 2013-2014 sichtbar

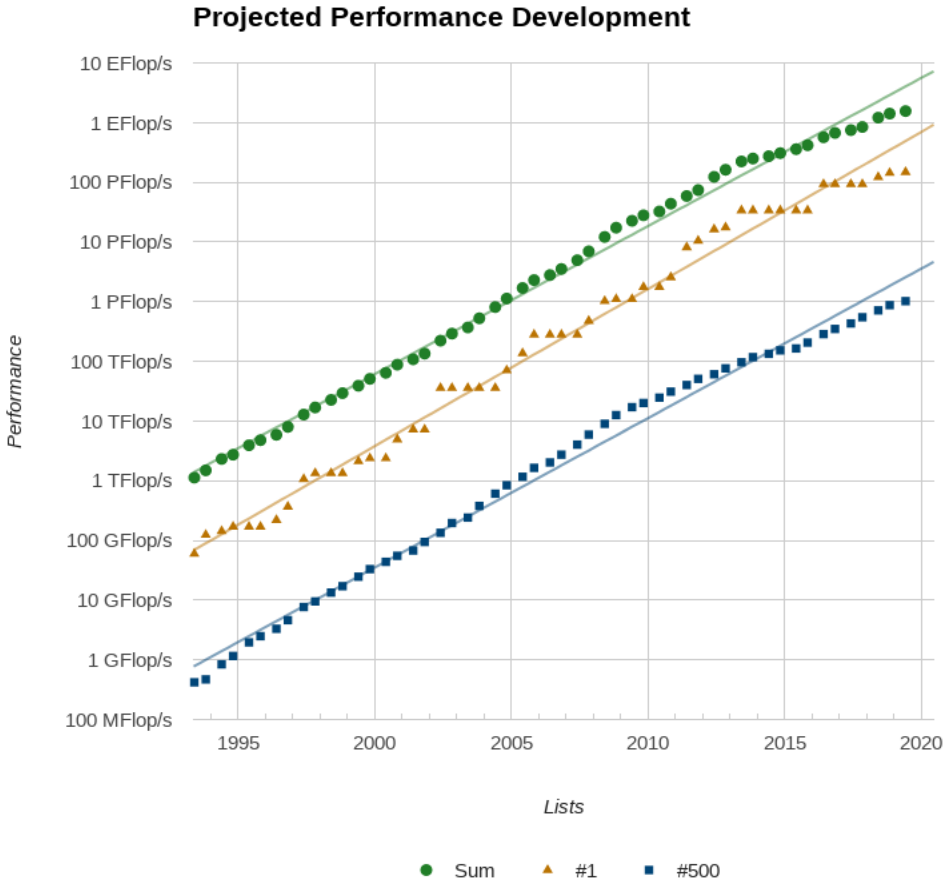

Abb. 2.1: Veranschaulichung der Top500. Abgebildet sind jeweils die Summe aller Plätze (Sum), erster Platz (#1), sowie letzter Platz (#500) zwischen 1993 und 2019 [92].

unterhalb des vorhergesagten Soll-Wertes.

2.1.3 Clock Moore und Dark Silicon

Mitte der 2000er endete das Dennard Scaling und infolgedessen stagnierten die Taktraten bei etwa 3 - 4 GHz. Zu dieser Zeit begann man, sich mit einer dynamischen Taktfrequenz auseinanderzusetzen. Während sich More Moore, More than Moore and Beyond More darauf konzentrierten, immer mehr Rechenleistung auf kleinere Flächen unterzubringen, legt man mit Clock Moore nun den Fokus darauf, mehr Rechenleistung aus der Perspektive der Zeit (gemeint ist hier die Laufzeit) zu ziehen. Dies führt jedoch unumgänglich dazu, dass nicht alle Einheiten, wie beispielsweise die CPU-Kerne, zur gleichen Zeit die volle Leistung entfachen können. Diese Abwärme wäre enorm. Man spricht hier von der Dark Silicon Ära. Vereinfacht gesagt, besagt das “Dark Silicon”, dass manche Komponenten ausgeschaltet oder heruntergefahren sind, während andere die Leistung brauchen. Die hohen Taktfrequenzen von 4 GHz und drüber können nur

noch auf wenigen Kernen gleichzeitig und meistens auch nur über eine kurze Dauer gehalten werden. Intel nennt das bei den eigenen Prozessoren Turbo-Boost. Einen genauen Einblick in das Dark Silicon bekommt man in der Publikation [42].

2.2 Terminologie

Dieses Kapitel führt die wichtigsten Begrifflichkeiten ein, die in dieser Dissertation des öfteren auftauchen und die Grundlage der Forschung bilden. Diese wären folgende:

- DVFS (Dynamic Voltage and Frequency Scaling)
- RAPL (Running Average Power Limit)
- MSR (Model Specific Registers)
- Intels (Power) Domänen

Die Begrifflichkeiten und Ausführungen basieren, wenn nicht anders angegeben, auf der Publikation [53].

2.2.1 DVFS - Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) bezeichnet die Möglichkeit, die Frequenzen und die Spannung zu verändern. Speziell beim Energiemanagement von Prozessoren wird dieses Verfahren eingesetzt, um die Taktfrequenz des Prozessors in Abhängigkeit zur Laufzeit zu verändern. So sinkt bei niedrigen Taktraten zwar die benötigte Spannung, jedoch beeinflusst eine niedrigere Taktrate die Laufzeit negativ, da die benötigte Leistung und der daraus resultierende Energieverbrauch abhängig von der gewählten Anwendung sind.

2.2.2 RAPL - Running Average Power Limit

Mit Einführung der Sandy Bridge Prozessoren war es erstmals möglich, den Energievierbauch von Prozessoren onboard auszulesen. Dabei stellt RAPL Sensoren bereit, die die Leistungsaufnahme verschiedener Komponenten der CPU auslesen. Diese Komponenten sind die beiden Power Planes PP0 und PP1 mit den entsprechenden Sensoren `RAPL_PP0` und `RAPL_PP1`. Weiter gibt es noch Sensoren zum Auslesen der Spannung des DRAM `RAPL_DRAM` und der Package-Spannung `RAPL_PKG`. Eine Übersicht über die Domänen findet sich im Unterkapitel *Domänen*. Nicht alle Sensoren sind auf allen Prozessoren verfügbar.

MSR - Model Specific Registers

Der Zugriff auf die RAPL-Sensoren erfolgt über Kontrollregister, oftmals als Modellspezifische Register (MSR) (engl. model specific registers), bezeichnet. Intel stellt hierfür zwei Instruktionen, `readmsr` und `writemsr` bereit. Linux Benutzer können das MSR -Kernel-Modul laden und dann direkt die Register auslesen oder via virtuellen Filesystem `sysfs` auf die Energiewerte sogar im User-Space ohne Privilegien zugreifen. Die Einheit der Register ist in Mikro-Joule gespeichert. Dadurch sind die Zahlenwerte sehr hoch und es kommt je nach Auslasung der CPU zu Überlaufen der Register, da diese in den meisten Fällen nur 32kb darstellen können. Der Wert aus diesen Registern ist sehr akturat: In [51] werden die MSR Werte mit den realen Messwerten aus einem Messgerät verglichen. In Kapitel 2.3 werden verschiedene Möglichkeiten zur Messung der Energie detailliert aufgezeigt.

Domänen

Ab der Sandy Bridge Architektur werden drei Power-Domänen unterstützt: package (PKG), PowerPlane0 (PP0), PowerPlane1 (PP1) und DRAM. Die Domäne Unified Core (Uncore) kann durch Subtraktion von PP0 und PP1 von PKG berechnet werden, siehe Abbildung 2.2. Als Uncore wird bei Intel all das bezeichnet, was sich zwar auf PP0 aber außerhalb von PP1 befindet. Bezogen auf die eben erwähnte Abbildung wären das die Grafikeinheit und der Last Level Cache samt Memory Controller. Eigene Tests haben gezeigt, dass PP1 nur ausgelesen werden konnte, wenn die interne Grafikeinheit auch verwendet wurde. Eine separate Messung des Caches war bei bei den Tests nicht möglich. Die DRAM Domäne war auf einigen Rechnern auch nicht auslesbar. Bei einem Intel Core i7-6950X Broadwell Prozessor war hingegen PP0 nicht auslesbar, sondern nur PP1.
Somit lässt sich behaupten, man sollte sich vor den eigentlichen Untersuchungen gut informieren, welche Domänen der gewünschte Prozessor bereitstellt.

2.2.3 Dynamische Frequenzen und Pstates

Als Quelle für dieses Kapitel diente hauptsächlich [67] mit den entsprechenden Unterkapiteln. Abweichende Informationsquellen wurden separat angeführt.
Auf aktuellen DVFS Prozessoren wird die Taktfrequenz auf Linux-Betriebssystemen von Kernel-Modulen - auch als Treiber bezeichnet - reguliert. Dabei kommt auf neueren Intel Prozessoren hauptsächlich der `intel_pstate driver` zum Einsatz. Bei AMD Systemen wird alternativ dazu der `acpi-cpufreq driver`, kurz `acpi`-Treiber eingesetzt. Acpi

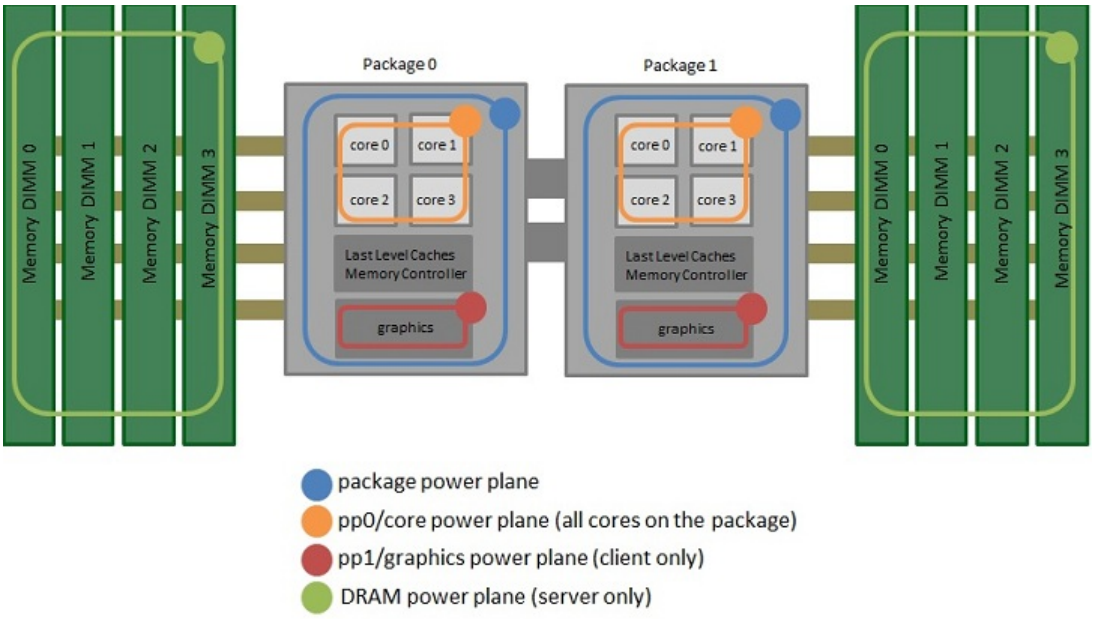

Abb. 2.2: Verfügbare Power Domänen auf 2 Sockets. Zum Ausrechnen der Uncore Domäne müssen PP0 und PP1 von PKG subtrahiert werden. [70]

steht für *Advanced Configuration and Power Interface*. Beide Module unterstützen die Enhanced Intel-SpeedStep-Technologie kurz `EIST`. Diese Energiesparfunktion ermöglicht dynamisches Spannungs- und Taktmanagement. Während diese Technik zu Anfangszeiten lediglich prüfte, ob ein Notebook am Netzteil oder am Akku betrieben wurde und die CPU entsprechend hoch- bzw. runtergetaktet wurde, so erlauben neue Revisionen anwendungsspezifisches `dynamic switching`. Hier wird der Takt beispielsweise im Idle -Zustand auf ein Minimum gesenkt und kann unter Last innerhalb von wenigen Millisekunden erneut angehoben werden[3]. Den aktiven Treiber kann man unter Linux (OpenSuse) mittels `cpupower frequency-info` abfragen. Es existieren neben dem `intel_pstate driver` und `acpi-cpufreq driver` noch weitere Kernel-Module. Aufgrund von Aktualität und Verfügbarkeit beschränkt sich dieser Abschnitt jedoch auf die beiden genannten.
Beide Treiber unterscheiden sich aus Anwendersicht in einigen Punkten. Ein signifikanter Unterschied betrifft die Governor, genauer gesagt die `Scaling Governor`. Diese sind vordefinierte Modi, die die Steuerung der Taktrate der CPU regulieren. Es gibt folgende Govenor zur Auswahl:

Beim intel_pstate-Treiber beschränkt sich die Auswahl auf zwei Governor: Performance und Powersave. Beim acpi-Treiber ist die Auswahl breiter gefächert: Performance,

[3] `https://de.wikipedia.org/wiki/Intel-SpeedStep-Technologie`

Governor	Kurzbeschreibung
Performance	CPU auf höchster Frequenz
Powersave	CPU auf niedrigster Frequenz
Userspace	CPU auf benutzerspezifischer Frequenz
Ondemand	Höchste Frequenz bei Belastung, sonst Minimum
Conservative	Belastungsabhängig - feingranulater als Ondemand
Schedutil	CPU taktet entsprechend eines Schedulers

Ondemand, Conservative, Userspace und Schedutil. Nicht jeder Govenor ist auf jedem Prozessor verfügbar. Standardmäßig ist der Ondemand Governor aktiviert. Der Conservative Governor kommt hauptsächlich auf mobilen, batteriebetriebenen Geräten zum Einsatz. Userspace wird aktiviert, sobald man die Frequenzen manuell setzt. Bei Server-CPUs, wie z.B. dem aktuellen AMD EPYC 7551 stehen nur drei Governor zur Verfügung: Ondemand, Performance und Schedutil.
Viele wichtige Systeminformationen speichert der Linux-Kernel im Unterverzeichnis `/sys/devices/system/cpu/cpu0/cpufreq/` ab. Ein Aufruf von `cpupower frequencyinfo` unter OpenSuse liest dieses Verzeichnis aus und gibt die wichtigsten Eigenschaften aus, siehe Listing 2.1. Unter der Ubuntu Distributionen verwendet man alternativ das Tool `cpufrequtils`.

```
root@epyc:~> # cpupower frequency-info
analysiere CPU 0:
  driver: acpi-cpufreq
  Folgende CPUs laufen mit der gleichen Hardware-Taktfrequenz: Not Available
  Die Taktfrequenz folgender CPUs werden per Software koordiniert: Not Available
  Maximale Dauer eines Taktfrequenzwechsels:  Cannot determine or is not
supported.
  Hardwarebedingte Grenzen der Taktfrequenz: 1.20 GHz - 2.00 GHz
  available frequency steps:  2.00 GHz, 1.60 GHz, 1.20 GHz
  mogliche Regler: ondemand performance schedutil
  momentane Taktik: die Frequenz soll innerhalb 1.20 GHz und 2.00 GHz.
                    liegen. Der Regler "ondemand" kann frei entscheiden,
                    welche Taktfrequenz innerhalb dieser Grenze verwendet wird.
  current CPU frequency: 1.20 GHz (asserted by call to hardware)
  boost state support:
    Supported: yes
    Active: yes
    Boost States: 0
    Total States: 3
    Pstate-P0:  2000MHz
    Pstate-P1:  1600MHz
    Pstate-P2:  1200MHz
```

Listing 2.1: Übersicht der verfügbaren Informationen zum Frequenzmanagement eines AMD EPYC 7551

In Zeile drei von Listing 2.1 sieht man den aktivierten Treiber und in Zeile elf die verfügbaren Governor. Neben dem Setzen der Governor, kann man auch die Frequenz auf ein bestimmtes Niveau festpinnen. Jeder Prozessor besitzt vordefinierte Frequenz-Stufen, sogenannte Power-States, kurz `Pstates`. Im Fall des AMD EPYC aus Listing 2.1 sind es drei Stufen: Pstate-P0, Pstate-P1 und Pstate-P2. Der niedrigste Pstate wird der höchsten Frequenz zugeordnet und der höchste Pstate der niedrigsten Frequenz. Der AMD EPYC besitzt nur drei Pstates, was die Granularität sehr grob hält. Auch auf weiteren getesteten AMD Ryzen CPUs wie dem Ryzen 5 2400G und dem Ryzen 7 2700X waren nur drei Pstates verfügbar. Je mehr Pstates eine CPU besitzt, desto feiner kann die Frequenz und damit das Energieverhalten manipuliert werden. Listing 2.2 zeigt vergleichsweise die 15 verfügbaren Pstates eines Intel i7-4770 Prozessor der Haswell Generation (Anhang A.2). Die verfügbaren Pstates wurden mit dem Tool `likwid` ausgelesen.

```
root@Haswell_acip:~> # likwid-setFrequencies -l
Available frequencies:
0.8 1 1.2 1.4 1.5 1.7 1.9 2.1 2.3 2.5 2.7 2.8 3 3.2 3.4
```

Listing 2.2: Einstellbare Frequenzen Haswell acpi-Treiber.

Aber nicht nur der Prozessor, sondern auch der aktive Treiber haben einen Einfluss auf die Pstates: Listing 2.3 zeigt die Pstates auf identischem Haswell-Prozessor, nur diesmal mit aktiven intel_pstate- Treiber:

```
root@Haswell_pstate:~> # likwid-setFrequencies -l
Available frequencies:
0.8 0.88 0.96 1.04 1.12 1.2 1.28 1.36 1.44 1.52 1.6 1.68 1.76 1.84 1.92 2 2.08
2.16 2.24 2.32 2.4 2.48 2.56 2.64 2.72 2.8 2.88 2.96 3.04 3.12 3.2 3.36
```

Listing 2.3: Einstellbare Frequenzen Haswell Intel Pstate Treiber.

Die niedrigste einstellbare Frequenz ist in beiden Fällen 800 MHz. Im Maximum beginnen schon die Unterschiede. Unter dem acpi-Treiber und deaktiviertem Intel-Turbo-Boost lässt der acpi-Treiber die CPU um 40 MHz höher Takten. Die Stufen steigen in etwa 100 bis 200 MHz Schritten beim acpi-Treiber. Beim intel_pstate-Treiber sind die Stufen nicht ganz so gleichmäßig aufgeteilt und es gibt hier viel mehr Frequenzen zur Auswahl. Es sei jedoch gesagt, das sich die Frequenzen nicht setzen lassen. Kann man beim ascpi-Treiber jede verfügbare (angezeigte) Frequenz festpinnen, funktioniert das Frequenzsetzen bei den verfügbaren Consumer-CPUs nicht. Bei Server-CPUs lies sich die Frequenz bei den meisten Stufen setzen. Manche Stufen wurden als verfügbar angezeigt, jedoch mit einer Fehlermeldung abgebrochen. Da dies von System zu System unterschiedlich ist und es nahezu immer ein paar wenige Frequenzen gibt,

welche nicht setzbar sind, wird empfohlen, den acpi-Treiber zu verwenden, wenn man vorhat die Pstates manuell zu setzen. Um diesen bei aktiven Pstate-Treiber zu aktivieren, muss diese in der GRUB-Commandline hinzugefügt werden. Dies geht wie folgt: Unter `/etc/default/grub` die Zeile `GRUB_CMDLINE_LINUX_DEFAULT` erweitern mit `GRUB_CMDLINE_LINUX_DEFAULT="intel_pstate=disable acpi=force"` und den GRUB mit `sudo update-grub` aktualisieren. Nach einem Neustart ist nun der acpi-Treiber aktiviert. Die ganze Prozedur und weitere Hintergrundinformationen können in [95] nachgelesen werden.
Neben den mit dem Power-States im Zusammenhang stehenden Kern-Frequenzen(engl. `Core-Frequency`), existieren noch die Uncore-Frequenzen (engl. `Uncore-Frequency`). Diese setzen die Frequenzen des auf dem Prozessor befindlichen Memory Controllers, genauer gesagt die des Ringbuses des L3-Caches, siehe Abbildung 2.2. Die Uncore-Frequenz wurde von Intel ab der Nehalem Reihe (mit Ausnahme der Sandy-Bridge Architektur) von derKern-Frequenz getrennt, um Energie zu sparen[4]. Genau wie die Core-Frequency unterliegt die Uncore-Frequency gewissen Grenzen. Das Tool Likwid setzt die minimale Uncore-Frequenz mit der geringsten Uncore-Frequenz gleich. Die Obergrenze ist jedoch höher als bei der Kern-Frequenz. Die Uncore-Frequenz-Stufen können innerhalb dieser Grenzen in 100 MHz Schritten verändert werden. Die aktuellen Frequenzen jedes einzelnen CPU-Kerns kann man sich mit Likwid mit `likwid-setFrequencies -p` anzeigen lassen, siehe Listing 2.4. Zusätzlich werden neben der minimalen, der aktuellen und maximalen Frequenz auch der Governor und Turbomode ausgegeben. Es ist jedoch nicht möglich die aktuellen Uncore-Frequenzen anzeigen zu lassen.

```
root@Haswell_pstate:~> # likwid-setFrequencies -p
Current CPU frequencies:
CPU 0: governor     ondemand min/cur/max 0.8/1.626/3.4 GHz Turbo 0
CPU 1: governor     ondemand min/cur/max 0.8/2.063/3.4 GHz Turbo 0
CPU 2: governor     ondemand min/cur/max 0.8/1.792/3.4 GHz Turbo 0
CPU 3: governor     ondemand min/cur/max 0.8/1.843/3.4 GHz Turbo 0
CPU 4: governor     ondemand min/cur/max 0.8/2.424/3.4 GHz Turbo 0
CPU 5: governor     ondemand min/cur/max 0.8/2.212/3.4 GHz Turbo 0
CPU 6: governor     ondemand min/cur/max 0.8/2.338/3.4 GHz Turbo 0
CPU 7: governor     ondemand min/cur/max 0.8/2.559/3.4 GHz Turbo 0

Current Uncore frequencies:
Socket 0: min/max 0.8/3.9 GHz
```

Listing 2.4: Ausgabe der aktuell eingestellten Core-Frequenzen und verfügbaren Uncore-Frequenzen mit likwid.

Neben den Pstates könnte man an dieser Stelle auch noch auf die C-States und S-States

[4] `https://www.anandtech.com/show/6355/intels-haswell-architecture/10`

eingehen. Erstere schicken den Prozessor in einen Stromsparmodus, wenn gerade nichts gerechnet wird. Je höher der C-State, desto geringer ist die Leisungsaufnahme im IDLE, desto höher ist die Latenz, den Prozessor wieder aufzuwecken. Der enstprechende C-State im Nicht-IDLE-Modus ist C0. Die S-States gehen noch einen Schritt weiter und beschreiben den Zustand des kompletten Systems. Der höchste S-State ist S5, bei dem das System komplett offline ist und somit fast gar keinen Strom benötigt[5]. Sowohl die C-States als auch die S-States spielen aber in dieser Arbeit keine Rolle. Deshalb sei für mehr Informationen auf Fremdliteratur hingewiesen [95].

2.3 Energiemessung unter Linux

In diesem Kapitel werden verschiedene Methoden beschrieben, mit denen man den Energieverbrauch der CPU messen kann. Es gibt natürlich nicht nur diese Möglichkeiten, jedoch sind dies die gängigsten.
Das Konzept der vorgestellten Tools ist jedoch immer das gleiche: Der Energiewert wird bei Intel-Prozessoren in modellspezifischen Hardware-Registern (MSR) gespeichert. Über das RAPL-Interface kann man zur Laufzeit auf diese Register zugreifen. Seit der zweiten Generation der Intel i-Prozessoren (ab Sandy Bridge) führt Intel diese Methode fort. Wie die Abkürzung MSR schon sagt, handelt es sich um modellspezifische Register auf der CPU, welche nicht überall vorhanden sind.

2.3.1 LIKWID

LIKWID [76] (Like I Knew What I'm Doing) ist eine Sammlung von einfach zu nutzenden Kommandozeilenprogrammen für die Performance-orientierte Programmierung im Linux-Umfeld, welche erstmals 2010 vorgestellt wurde. Mit diesen Programmen ist es beispielsweise möglich, die Cache- oder Kern-Topologie der CPU abzufragen, wichtige Performance-Daten einfach auszulesen oder ein Programm nur auf bestimmten CPU-Kernen ausführen zu lassen. Eine Besonderheit von LIKWID ist, dass es keine externen Abhängigkeiten besitzt und alles im Userspace implementiert ist. Dies ermöglicht es, schnell neue Architekturen zu unterstützen, ohne dabei bestimmte Kernel-Versionen vorauszusetzen. Eine Übersicht über die in LIKWID implementierten Tools liefert Tabelle 2.1. Die für diese Arbeit meist genutzten LIKWID-Programme sind

- likwid-setFrequencies

[5] `https://www.technikaffe.de/anleitung-32-c_states_p_states_s_states_energieverwaltung_erklaert/`

Programm	Kurzbeschreibung
likwid-topology	Ausgabe von Thread, Cache and NUMA topology
likwid-perfctr	Konfiguration und Ausgabe der Hardware Performance Counter
likwid-powermeter	Auslesen der RAPL Energie-Werte
likwid-pin	Setzen der Affinität der parallelen Anwendungen
likwid-bench	(Minimales) Benchmarken der Hardware
likwid-features	Ausgeben und Manipulieren der CPU Feaers
likwid-genTopoCfg	Dumpen der Topologie in eine Datei prefetchers
likwid-mpirun	Wrapper für MPI und Hybrid-Parallele Anwendungen
likwid-perfscope	Frontend zum plotten der timeline von likwid-perfctr
likwid-setFrequencies	Tool zum Kontrollieren der CPU und Uncore-Frequenzen
likwid-memsweeper	Leeren des Speichers der NUMA Domänen und Cachelines

Tab. 2.1: Übersicht und kurze Beschreibung der Sammlung an likwid Unterprogrammen.

- likwid-powermeter
- likwid-perfctr.

Diese Tools werden im nachfolgenden Teil kurz erläuert.

likwid-setFrequencies

Das Programm likwid-setFrequencies dient dazu, die CPU-Frequenz von einzelnen Kernen zu setzen oder auszulesen (vgl. Kapitel 2.2.3). Gerade im Zusammenhang mit Benchmarks ist dies wichtig, um eine definierte Umgebung zu erhalten, in der alle CPU-Kerne mit der gleichen Geschwindigkeit arbeiten. Zusätzlich ist es auf bestimmten CPUs von Intel möglich, die Uncore-Frequenz zu verändern. Bei Intel-Prozessoren bezieht sich der Begriff Uncore dabei auf alle Teile des Chips, die nicht Teil des Prozessorkerns sind, wie z. B. gemeinsamer Last-Level-Cache, Ring-Interconnect und Speichercontroller (vgl. Kapitel 2.2.2).

likwid-perfctr

Mit likwid-perfctr ist es möglich, bestimmte Performance-Metriken über die Laufzeit eines Programms zu messen. Hierzu sind auf allen Prozessoren eine Reihe von vordefinierten Performance-Gruppen verfügbar, die mehrere Events für die Berechnung der gewünschten Metrik zusammenfassen.

likwid-powermeters

Mit Hilfe von likwid-powermeter ist es möglich, den Energieverbrauch innerhalb eines CPU-Packages für einen bestimmten Zeitraum abzufragen und den daraus resultierenden Stromverbrauch berechnen zu lassen.

2.3.2 PAPI

Neben LIKWID gibt es auch die PAPI (Performance Application Programming Interface) API [57]. Diese bietet dem Nutzer einen einfachen Zugriff zu Performance-Counter. Diese werden bei PAPI als Events bezeichnet. Nach der Installation kann man mit `papi_avail` alle auf dem System verfügbaren Events auflisten. Eine gute Übersicht über die Möglichkeiten und Limitierungen von Papi 5.0 finden sich in [61].

2.3.3 Sysfs

Neben LIKWID und PAPI kann man den Energieverbrauch auch ohne externe Tools auslesen. In das virtuelle Dateisystem des Linux-Kernel, dem `sysfs`, werden Subsysteme, Hardware und Geräte-Treiber als virtuelle Dateien gespeichert. Auf diese kann der Benutzer im User-Mode zugreifen. So kann man das auch mit dem Energieverbrauch ab Linux-Kernel 3.3 machen. Dazu navigiert man bei einem Intel Prozessor und einem Linux Betriebssystem nach `/sys/class/powercap/intel-rapl/`. Über die Unterordner `intel-rapl:0` bzw. `intel-rapl:1` hat man nun Zugriff auf alle wichtigen Domänen. Abbildung 2.3 zeigt einen Auszug der Dateistruktur. Die ersten Ziffern der Unterordner geben den Socket an. Navigiert man dort hin, hat man Zugriff auf die Domäne `package-0` bzw. `package-1` für den ersten CPU-Sockel bzw. dem zweiten. Über die Datei `name` liest man die Domäne aus. Die Datei `energy_uj` ist das Abbild des Energiewertes der Register. Die Einheit ist `Micro-Joule` und aktualisiert sich etwa alle 10 ms. Da das Register nur 32-bit groß ist, können nur eine beschränkte Anzahl an Ganzzahlen dargestellt werden und so kommt es hin und wieder zu einem Überlauf. Dies muss man berücksichtigen, indem Start und Endwert des Registers verglichen werden. LIKWID registriert so einen Überlauf automatisch. Der im Register größte darstellbare Zahlenwert ist auch in der Datei `max_energy_range_uj` hinterlegt. Dieser Wert ist ausschlaggebend für den Überlauf und sollte berücksichtigt werden.
Weiter beinhaltet die Package-Domäne die Subdomänen: Diese sind in Unterordnern mit selber Namensstruktur organisiert. Diesmal mit, je nach verfügbaren Domänen, einem zweiten Index, beginnend mit `0`. Dort findet man nun wieder die bekannten

Dateien `name` mit entsprechend gespeicherter Domäne und `energy_uj`. Das Beispiel in Abbildung 2.3 hat 4 Domänen: Package, Core, Uncore und Dram. Üblicherweise sind aber nicht immer alle 4 Domänen verfügbar.
Es ist durch die Einführung der RAPL-Sensoren und Kernel 3.3 somit auch ohne Root-Zugriff und zusätzliche Software möglich, schnell und einfach den Energieverbrauch der einzelnen Komponenten am Prozessor zu messen. Es reicht dafür ein Iterieren über die Ordnerstruktur und das Bilden der Differenzen der entsprechenden Energiewerte. Diese und weitere Erklärungen finden sich in [71].

Abb. 2.3: Ordner/Datei-Struktur des Intel-Rapl-Interfaces im Userspace am Beispiel eines Intel Haswell i7-4770 mit 4 verfügbaren Domänen unter der Linux-Distribution Suse 15.1.

2.4 Benchmarks Suites

Zur Datenauswertung in dieser Arbeit dienten vor allem die beiden Benchmarksuites PARSEC und SPLASH-2 als Hauptanwendungen, da diese eine große Variation an Programmcharakteristika bieten. In diesem Kapitel werden beide Suites kurz vorgestellt. Dieses Kapitel geht kurz auf beide Benchmark Suites ein und gibt Hintergrundinformationen zu den einzelnen Benchmarks, die sich dahinter befinden.

2.4.1 PARSEC Benchmark Suite

Die PARSEC Benchmark Suite [19] (Princeton Application Repository for Shared-Memory Computers) ist eine Ansammlung an 13 Benchmarks zur Untersuchung von Thread-Parallelität auf aktuellen Multiprozessoren. Dabei konzentrieren sich die einzelnen Benchmarks auf größeren Workloads, die typisch für heutige Anwendungen sind. Im Gegensatz zu den SPLASH2 Benchmarks, die hauptsächlich wissenschaftliche Berechnungen behandeln, beinhalten die PARSEC Benchmarks eine breitere Front an Anwendungen im kommerziellen Bereich, wie der Finanzwelt, Animation, Data Mining oder Enterprise Storage. Eine genaue Untersuchung zu den einzelnen Benchmarks findet sich in [19, 18] wieder. Ein Auszug der wichtigsten Kennzahlen ist im weiteren Verlauf aufgelistet. Die Benchmarks der Suite im Überblick:

- **blackscholes**: Eine Anwendung zur Bewertung von Finanzoptionen, die mit dem Black-Scholes-Modell Preise für ein europäisches Portfolio berechnet.
- **bodytrack**: Eine Anwendung, die durch maschinelles Sehen die dreidimensionale Figur eines Menschen erfasst. Genutzt werden dabei Aufnahmen von mehreren Kameras.
- **canneal**: Eine Anwendung, die durch „Simulated Annealing“, ein heuristisches Approximationsverfahren, die Routing-Kosten beim Chip Design minimiert.
- **dedup**: Eine Anwendung, die mithilfe von Deduplikation sehr hohe Kompressionsraten von Testdaten erzeugt.
- **facesim**: Eine Anwendung, die auf der Basis eines Modells des menschlichen Gesichts und einer Zeitsequenz aus Muskelkontraktionen eine physikalisch korrekte Animation des Gesichts erzeugt.
- **fluidanimate**: Eine Anwendung, bei der die Methode der geglatteten Teilchen-Hydrodynamik genutzt wird, um eine nicht komprimierbare Flüssigkeit zu simulieren.
- **freqmine**: Eine Anwendung, die auf der Grundlage der Frequent Pattern-growth Methode Data Mining betreibt.

- **streamcluster**: Eine Anwendung, bei der für einen Eingabestrom bestehend aus Punkten eine vorgegebene Anzahl an Medianen gefunden wird, so dass jeder Punkt seinem nächstgelegenen Clustermittelpunkt zugeordnet wird.
- **swaption**: Eine Anwendung, bei der durch Berechnung von partiellen Differentialgleichungen durch Monte Carlo Simulationen die Preise der „Swaptions“ eines Portfolios berechnet werden.
- **vips**: Eine Anwendung, die parallele Bildverarbeitung mit unterschiedlichen Operationen, wie z.B. affine Transformationen oder Konvolutionen betreibt.
- **x264**: Eine Anwendung, die mit dem H.264/AVC (Advanced Video Coding) Standard Videos enkodiert.

2.4.2 SPLASH-2 Benchmarksuite

Die SPLASH-2 Benchmark Suite aus dem Jahr 1995 ist zwar schon etwas älter, jedoch immer noch sehr beliebt. Sie besteht aus 11 Benchmarks. Sie setzt den Fokus ihrer Anwendungen hauptsächlich auf wissenschaftliche Berechnungen und Computergrafik. Darunter fallen beispielsweise die Cholesky-Zerlegung und die LU-Faktorisierung, aber auch Raytracing und Radiosity-Algorithmen. Aufgrund der damaligen Architekturen paralleler Maschinen wurden die Codes auf Multi-Nodes ausgelegt, so dass versucht wurde, die Kommunikation zwischen einzelnen Prozessen auf ein Minimum zu reduzieren oder auf diese ganz zu verzichten. Ein genauer Vergleich mit der PARSEC Benchmark Suite findet sich in [18]. Nachfolgend werden die einzelnen SPLASH-2 Benchmarks kurz beschrieben:

- **barnes**: Ein Benchmark, der den Barnes-Hut Algorithmus implementiert, welches ein Näherungsverfahren für eine effektive Berechnung der Kräfte in einem Mehrkörper-Problem möglich macht. Bei dieser Implemtierung liegt die Problemgröße bei 65536 Partikeln.
- **cholesky**: Ein Benchmark zur Zerlegung einer symmetrisch positiv definiten Matrix in eine Dreiecksmatrix unter Anwendung der Cholesky-Faktorisierung.
- **radix**: Ein Benchmark, der einen parallelen Radix-Sort implementiert der 8388608 Integer sortiert.
- **fmm**: Ein Benchmark als Alternative zu barnes zur schnellen Lösung des N-Körper-Problems mit 65536 Partikeln.
- **fft**: Ein Benchmark, der eine schnelle Fourier-Transformation (engl. fast Fourier transform) implementiert. Dieser Algorithmus dient der effizienten Berechnung der diskreten Fourier-Transformation und kommt aus der Signalverarbeitung. Die

Anzahl an Datenpunkten beträgt 4194304.

- **lu**: Ein Benchmark, der eine 1024×1024 Matrix in 64×64 Blöcke aufteilt und das Produkt aus einer oberen und unteren Dreiecksmatrix berechnet.
- **ocean**: Ein Benchmark, der weitläufige Ozeanbewegungen auf einem Gitter von 514×514 simuliert, die auf Strömungen basieren.
- **raytrace**: Ein Benchmark zum Rendern einer dreidimensionalen Szene auf Bildebene.
- **radiosity**: Ein Benchmark zur Berechnung der Lichtverteilung in einem Raum.
- **volrend**: Ein Benchmark ähnlich zu raytrace, jedoch mit einer anderen zugrundeliegenden Implementierung.
- **water**: Ein Benchmark, der ein N-Köper-Problem für 4096 Wassermoleküle berechnet.

3 | Metriken zur Bewertung der Energieeffizienz

Anwendungsprogramme nutzen die Hardware aufgrund interner Ausführungseigenschaften sehr unterschiedlich aus, was zu einem recht unterschiedlichen Verhalten hinsichtlich ihrer Leistung oder der Energie, die für ihre Ausführung verbraucht wird, führt. Eine Änderung der Frequenz von DVFS-Prozessoren sowie die Nutzung von Thread-Parallelität auf Multicore Systemen führt zu weiteren Variationen in Leistung und Energieverbrauch. Dieses Kapitel kombiniert Frequenzskalierung und Thread-Parallelität und zeigt mehrere Metriken zur Bewertung dieser. Anhand der PARSEC Benchmarksuite und der SPLASH-2 Benchmarksuite werden diese Metriken abschließend evaluiert. Die Benchmarks wurden in den Kapiteln 2.4.1 und 2.4.2 beschrieben.
Als Hardwareplattform dienen eine Haswell (Anhang A.2) und eine Skylake (Anhang A.1) Plattform und zusätzlich eine ARM Architektur (Anhang A.5). Die genauen Daten zu den verwendeten Systemen sind in den verlinkten Anhängen beschrieben. Die PARSEC und SPLASH-2 Benchmarksuites wurden in Kapitel 2.4 genauer eingeführt. Als Grundlage inklusive den Diagrammen und Tabellen dienen, wenn nicht anders angegeben, die eingereichten Publikationen [4, 3].

3.1 Einführung und Motivation

3.1.1 Motivation

Die Leistung und die Energieeffizienz einer laufenden Anwendung zeigen ein unterschiedliches Verhalten in Abhängigkeit von verschiedenen Faktoren: Der Software selbst und der benutzten Hardwareplattform. Auf der Softwareseite umfassen die Faktoren den Algorithmus und wie die spezifische Implementierung die Hardware der Ausführungsplattform nutzt. Auf der Hardwareseite gehören zu den Faktoren die interne Organisation, also die Prozessorarchitektur, sowie die spezifische Einstellung von Hardware-Parametern, die für die Ausführung ausgewählt wurden, wie beispielsweise die Anzahl der Threads

oder die Betriebsfrequenzen der DVFS-Prozessoren. Es ist schwer vorherzusagen, wie sich Parametereinstellungen auf die Leistung oder Energieeffizienz auswirken. Genau das greift dieses Kapitel auf. Es ist auch schwer vorherzusagen, ob die Verwendung von mehreren Threads profitabel ist und dies zu einer kürzeren Ausführungszeit einer Anwendung führt oder nicht. In den meisten Fällen gibt es eine optimale Anzahl an Threads über die hinaus die Ausführungszeit nicht weiter reduziert werden kann und es zu einer Sättigung kommt. Diese gilt es zu finden. Ebenso kann die Verwendung einer geringeren CPU-Frequenz zu einer Reduzierung des Energieverbrauchs führen. Wie bei den Threads gibt es auch hier oft eine optimale Betriebsfrequenz, ab der eine weitere Frequenzreduzierung nicht zu einer weiteren Energieeinsparung führt. Der Idealfall wäre eine lineare Abhängigkeit zwischen Ausführungszeit und Energieverbrauch. Wäre das gegeben, so würde die schnellste Ausführungszeit immer den geringsten Energieverbrauch bedeuten. Wegen des Stromverbrauchs existiert aber eine lineare Abhängigkeit in der Form nicht. Um ein Programm auf Leistung und/oder Energieeffizienz abzustimmen, benötigt es Metriken zur Bewertung, die diese speziellen Eigenschaften erfassen können.

Für die Leistungsbewertung (Performance) zur Lösung eines bestimmten Problems (Time-to-Solution) werden Laufzeit-Speedup, die Anzahl der Gleitkommaoperationen pro Sekunde (Flops/s) oder die insgesamt benötigte Ausführungszeit verwendet. Für die Bewertung der Leistung oder Energieeffizienz beinhalten die Metriken die Performance pro Watt oder die zu benötigte Energiemenge um ein Problem zu lösen (Energy-to-Solution). Ansätze, die Laufzeitleistung und den Energieverbrauch in eine fusionierte Metrik zu verbinden, ist das EDP (engl. Energy-Delay Product), was man mit Energieverzögerungsprodukt übersetzen könnte.
In diesem Kapitel werden die verschiedenen Wechselwirkungen zwischen verschiedenen Threadzahlen und unterschiedliche Frequenzen auf die Laufzeit und die Energie, die mitunter in der Summe der Vielzahl von Anwendungen sehr kompliziert sein können, aufgezeigt und in neue und bekannte Metriken verpackt. Es stellte sich die Frage, ob neue geeignete Metriken definiert werden können, die die Abhängigkeit von den beiden variierenden Parametern auf Performance und Energie erfassen können und somit Verbesserungseffekte quantitativ ausgedrückt werden können. Nach dieser Motivation geht es in dem Kapitel wie folgt weiter:
Kapitel 3.1.2 stellt den Unterschied zwischen Leistungsaufnahme und Energie ausführlich dar. Kapitel 3.2 führt etablierte und neue Metriken zur Performance- und Effizienzbewertung von Programmausführungen ein. Kapitel 3.3 untersucht und evaluiert die Metriken anhand ausgewählter Benchmarks. Das letzte Unterkapitel 3.6 zieht ein Fazit

und fasst den Vorteil der neuen Metriken zusammen.

3.1.2 Energie und Leistung

Energie und Leistung werden oft als Synonyme verwendet. Insbesondere im Englischen ist oft nicht auf den ersten Blick klar, welches der beiden Begriffe gemeint ist. Der englische Ausdruck *Power* kann wörtlich übersetzt sowohl Energie, als auch Leistung(saufnahme) bedeuten. Übersetzt man beispielsweise den englischen Begriff *Power Consumption* aus dem Englischen ins Deutsche, werden die folgenden Übersetzungsvorschläge unterbreitet[1]: Stromverbrauch, Leistungsaufnahme und Energieverbrauch. Man bekommt hier für eine Übersetzung drei unterschiedliche Begriffe mit unterschiedlichen Einheiten. Genauer ist hier die englische Bezeichnung *Power Drain*, welche eindeutig die Leistungsaufnahme meint. Um keine Missverständnisse beim Lesen zu schaffen, werden in diesem Kapitel die Begriffe Energie und Leistung genauer eingeführt.
Die `Energie` E, die während der Ausführung vom ausgeführten Programmcode verbraucht wird, hängt von der Ausführungszeit t auf der zugrundeliegenden Hardware und von deren `Leistungsaufnahme` P ab. P kann während der Ausführung des Codes schwanken. Infolgedessen wird die Energie E als Integral $E = \int_{t=0}^{t_{end}} P(t)dt$ ausgedrückt unter der Annahme, das Programm wurde von Zeitpunkt $t = 0$ bis $t = t_{end}$ ausgeführt. Dementsprechend ist die Leistungsaufnahme P eine Momentaufnahme, wohingegen die Energie E eine periodische Einheit ist, um den Energieverbrauch zu beschreiben. Energie wird in Joule (J) oder Wattsekunden (Ws) angegeben und die Leistungsaufnahme in Watt (W). Nicht zu verwechseln ist die Leistungsaufnahme mit der TDP - der Thermal Design Power. Beide Metriken haben die Einheit Watt. Jedoch handelt es sich bei der TDP um die Verlustleistung, genauer gesagt, die Leistung, die maximal in Wärme umgewandelt wird. Die TDP gibt somit an, wie groß die Kühlleistung des Prozessors dimensioniert werden muss[2].
Weiter wird angenommen, dass DFVS Prozessoren p_{max} Kerne haben und mit der Frequenz f zwischen f_{min} und f_{max} betrieben werden. Da, wie in Kapitel 2.2.3 erklärt, nicht jede beliebige Frequenz zwischen f_{min} und f_{max} eingestellt werden kann, wird noch ein dimensionsloser Skalierungsfaktor $s \geq 1$ eingeführt, welcher eine kleinere Frequenz $f \leq f_{max}$ relativ zum Maximum f_{max} angibt als $f = f_{max}/s$. Der maximale Skalierungsfaktor ist somit $s_{max} = f_{max}/f_{min}$.
Die Leistungsaufnahme P variiert mit der Anzahl der für die Ausführung einer Anwendung eingesetzten Threads (Prozesse) p und der Frequenz f und kann somit als

[1] `https://dict.leo.org/englisch-deutsch/Power%20consumption`
[2] `https://de.wikipedia.org/wiki/Thermal_Design_Power`

Funktion mit p und s ausgedrückt werden als $P = P(p, s)$. Infolgedessen hängt auch die Energie von p und s ab und wird als

$$E(p, s) = \int_{t=0}^{t_{end}(p,s)} P(p, s)(t)dt. \tag{3.1}$$

definiert. Als Vereinfachung wird $P(p, s)$ oft als konstanter Mittelwert während der Laufzeit der Anwendung angenommen. Es handelt sich also um eine Diskretisierung mit festen Schrittweiten, über die dann die Summe gebildet wird. Die Schrittweite P sind nur wenige Millisekunden. Nun kann die Gleichung (3.1) als

$$E(p, s) = P(p, s) \cdot T(p, s) \tag{3.2}$$

umgeschrieben werden mit $T(p, s) = t_{end}(p, s)$ als Gesamtausführungszeit. In der Praxis ist E eine diskrete Funktion über p und s. Die Anzahl an Threads ist eine natürliche Zahl und die Frequenz kann nicht kontinuierlich gewechselt werden.
Da die beiden Entwicklungskriterien für Anwendungen, die Ausführungszeit und der Energieverbrauch, von zwei unabhängigen Variablen p und s abhängen, kann die Auswertungsfunktion $\mathcal{E} = (E, T)$ mit einer zweidimensionalen Funktion

$$(E, T) : [1, p_{max}] \times [1 : s_{max}] \subset \mathbb{R}^2 \to \mathbb{R}^2$$

mit

$$(p, s) \mapsto (E(p, s), T(p, s)).$$

beschrieben werden. Die Performance jeder Anwendung kann somit durch eine Fläche über einem zweidimensionalen Raster an (p, s)-Punkten abgebildet werden. Abbildung 3.1 zeigt beispielhaft die Oberfläche des Energieverbrauchs und der Laufzeit für den Benchmark `freqmine` aus der PARSEC Benchmarksuite auf einem Skylake Prozessor. Die untere Fläche ist dabei die Laufzeit und die obere Fläche repräsentiert den Energieverbrauch. Sehr gut zu erkennen ist hier, dass beide unterschiedlich verlaufen. Beim Untersuchen der Eigenschaften einer Anwendung müssten beide Flächen betrachtet werden, sei es zum Finden des geringsten Energieverbrauchs E oder der geringsten Laufzeit T. Noch deutlicher wird dies durch Betrachtung der Pareto Punkte in Abbildung 3.2. Hier wird die Laufzeit auf der x-Achse und die Energie auf der y-Achse abgebildet. Gut zu erkennen sind die Verläufe der einzelnen Threads. Bei einem Thread bewegt man sich weit rechts und weit oben in der linken Abbildung, wohingegen bei 8 Threads die Werte unten links zu sehen sind. Ein optimaler Kompromiss bewegt sich also bei

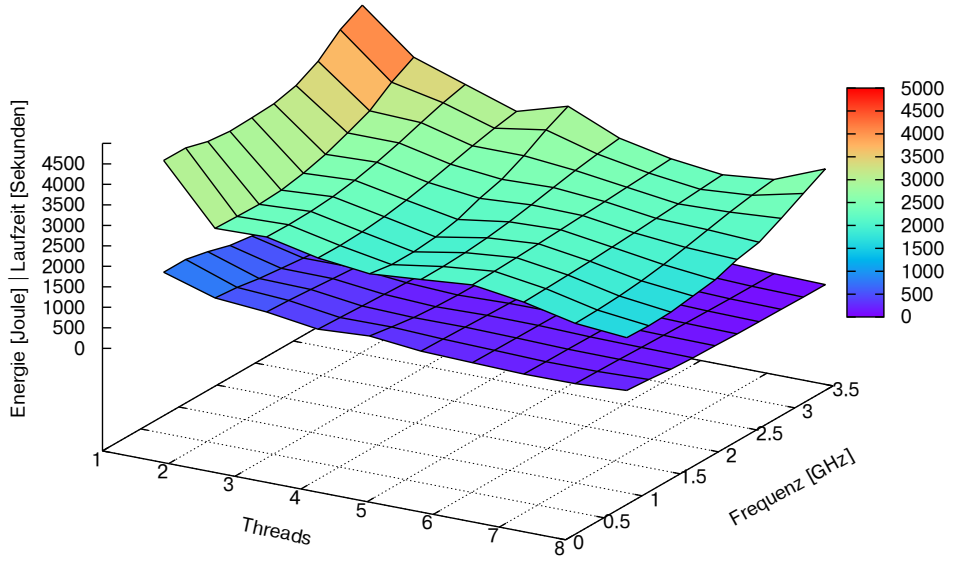

Abb. 3.1: Energieverbrauch (obere Fläche) und Laufzeit (untere Fläche) des Freqmine Benchmarks mit einer unterschiedlichen Anzahl an Threads und verschiedenen Frequenzen auf einem Skylake i7-6700 Prozessor.

niedrigen y-Werten und niedrigen x-Werten. Auf der rechten Seite wurde in die Abbild reingezoomt, wobei die Werte für 1 Thread nicht mehr sichtbar sind, sondern nur noch für die Threadzahlen 2, 4 und 8. Die Pareto-Front könnte man etwa bei 200 Sekunden und 200 Joule erkennen. Hier sind beide Parameter (Frequenz und Threadzahl) sehr gut. Die Frequenzen bewegen sich hier zwischen 1.4 GHz und 2.5 GHz. Eine detailierte Patero-Optimierung findet sich in [10].
Eine Übersicht der Parameter und Symbole findet sich in Tabelle 3.1 wieder.

Abbildung 3.3 vergleicht die Leistungsaufnahme eines Haswell Prozessors und einem

Tab. 3.1: Symbole und Einheiten aus Kapitel 3.1.2.

	Notation	
Symbol	Einheit	Bedeutung
p	Skalar	Anzahl an Threads
f	1/Sekunden	Betriebsfrequenz
f_{max}	1/Sekunden	Maximale Prozessor-spezifische Frequenz
f_{min}	1/Sekunden	Minimale Prozessor-spezifische Frequenz
$s \geq 1$	dimensionslos	Skalierungsfaktor $f = f_{max}/s$
s_{max}	dimensionslos	Maximaler Skalierungsfaktor
$T(p, s)$	Sekunden	Skalierte parallele Ausführungszeit
$P(p, s)$	Watt	Leistung für skalierte prallele Ausführung
$E(p, s)$	Joule	Energie für parallele Ausführung

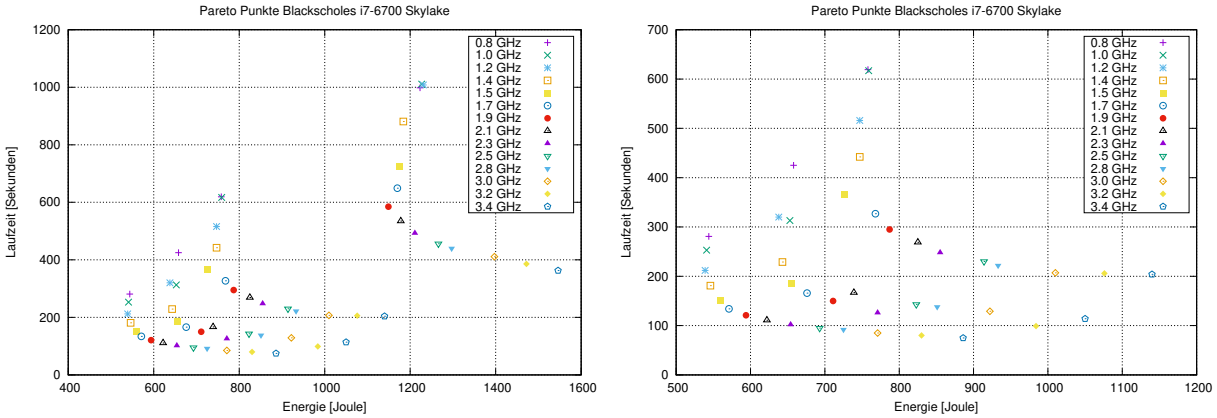

Abb. 3.2: Pareto Punkte aus Laufzeit in Sekunden und Energie in Joule des Benchmarks Blackscholes aus PARSEC auf einem Skylake i7-6700 Prozessor für 1, 2, 4 und 8 Threads (links) und reingezoomt für 2, 4 und 8 Threads (rechts).

Intel i7 Skylake Prozessors bei der Ausführung verschiedener PARSEC und SPLASH-2 Benchmarks für sequenzielle und parallele Ausführungen. Auf der Vertikalen ist die Leistungsaufnahme in Watt aufgetragen und auf der Horizontalen die Kern-Frequenzen. Es ist unschwer zu erkennen, dass die Werte für eine unterschiedliche Anzahl von Threads unterschiedlich sind. Ein niedriger Energieverbrauch bedeutet nicht zwangsweise, dass mit entsprechender Threadanzahl und Frequenz auch die niedrigste Leistungsaufnahme und/oder geringste Laufzeit zu finden ist. Unterschiedliche Benchmarks skalieren verschiedenartig unter Zunahme mehrerer Threads und Erhöhung der Frequenz. Bei der Leistung ist jedoch nicht von der Hand zu weisen, dass sowohl bei der Erhöhung der Frequenz als auch bei der Erhöhung der Threadanzahl die Leistung zunimmt. Die Transistoren auf dem Chip steigern die Anzahl an Zustandswechseln bei steigender Frequenz und bei der Erhöhung der Threadanzahl steigt auch die Anzahl der Transistoren, die aktiv bei den Berechnungen teilnehmen.
Vergleicht man den entsprechenden Energieverbrauch in Abbildung 3.4, erkennt man den architekturbedingten unterschiedlichen Verlauf. Der Skylake Prozessor hat den geringsten Energieverbrauch eher in niedrigeren Frequenzbereichen, wohingegen der Haswell Prozessor vor allem bei sequenzieller Ausführung der Benchmarks im mittleren Frequenzbereich energieoptimal läuft.

Bei dem Diagramm ist bei der Laufzeit zu erkennen, dass eine Steigerung der Frequenz eine niedrigere Laufzeit bedeutet. Dies sollte man jedoch nicht für alle Benchmarks verallgemeinern: Generell hat eine höhere Taktfrequenz eine schnellere Abarbeitung der Berechnung zufolge. Jedoch ist die Abarbeitungsgeschwindigkeit der Core-Frequenz

Tab. 3.2: Performance-, Leistungs- und Energie-Metriken

Symbol	Einheit	Bedeutung
$S(p,s)$	dimensionslos	Speedup für skalierten Fall, Gl. (3.3)
$R(p,s)$	dimensionslos	Laufzeit-Reduktions Fakto , Gl.(3.4)
$EDP(p,s)$	Joule*Sekunden	Energy-Delay Product, Gl.(3.5)
$ES(p,s)$	dimensionslos	Energie-Speedup, Gl. (3.6)
$ER(p,s)$	dimensionslos	Energie-Reduction Factor, Gl. (3.7)
$EPS(p,s)$	Joule	Energie per Speedup, Gl. (3.8)
$PS(p,s)$	dimensionslos	Power Speedup, Gl. (3.9)
$PI(p,s)$	dimensionslos	Power Increase Factor, Gl. (3.11)
$RPI(p,s)$	dimensionslos	Relative Power Increase Factor, Gl. (3.12)

abhängig vom Workload, der vom Hauptspeicher oder Cache in die Register geladen wird. Ist der Memorycontroller zu langsam, um die Workloads zu laden, so wirkt sich auch eine Erhöhung der Core-Frequenz nicht automatisch zugunsten der Laufzeit aus. Jedoch sei auch angemerkt, dass sich die Uncore-Frequenz automatisch erhöht, sobald sich die Kern-Frequenz erhöht - sofern man die Uncore-Frequenzen nicht selbst setzt. Dies ging aus einer Anlayse in [12] hervor. Hier wurde die Gesamtenergie des Prozessors ohne manuelles Setzen der Uncore-Frequenz gemessen und anschließend wurde die Uncore-Frequenz varriert. Entsprach die Uncore-Frequenz der Core-Frequenz, so haben sich die Energieverläufe bis auf geringe Messtoleranzen überlappt.

3.2 Energie- und Performance-Metriken

In diesem Kapitel werden einige Metriken eingeführt, welche verschiedene Leistungsaspekte reflektieren, die zum einen von der Threadzahl p als auch vom Skalierungsfaktor s der entsprechenden Frequenz f abhängen. Neben neuen Metriken zählen hier auch etablierte Metriken wie das Energy-Delay Product EDP (auf deutsch in etwa Energieverzögerungsprodukt). Die Metriken unterscheiden zwischen Performance, Energie und einer Mischung aus beiden. Tabelle 3.2 fasst die in diesem Kapitel eingeführten Metriken zusammen. Eine Erklärung zu den einzelnen Metriken finden sich in den entsprechenden Unterkapiteln. Die Wertetabellen und Diagramme wurden mit der PARSEC Benchmarksuite durchgeführt.

3.2.1 Speedup

Der (Laufzeit-)`Speedup` S wird als Metrik häufig zur Evaluierung der Performance paralleler Programme verwendet. Er drückt aus, wie stark die Ausführungszeit durch

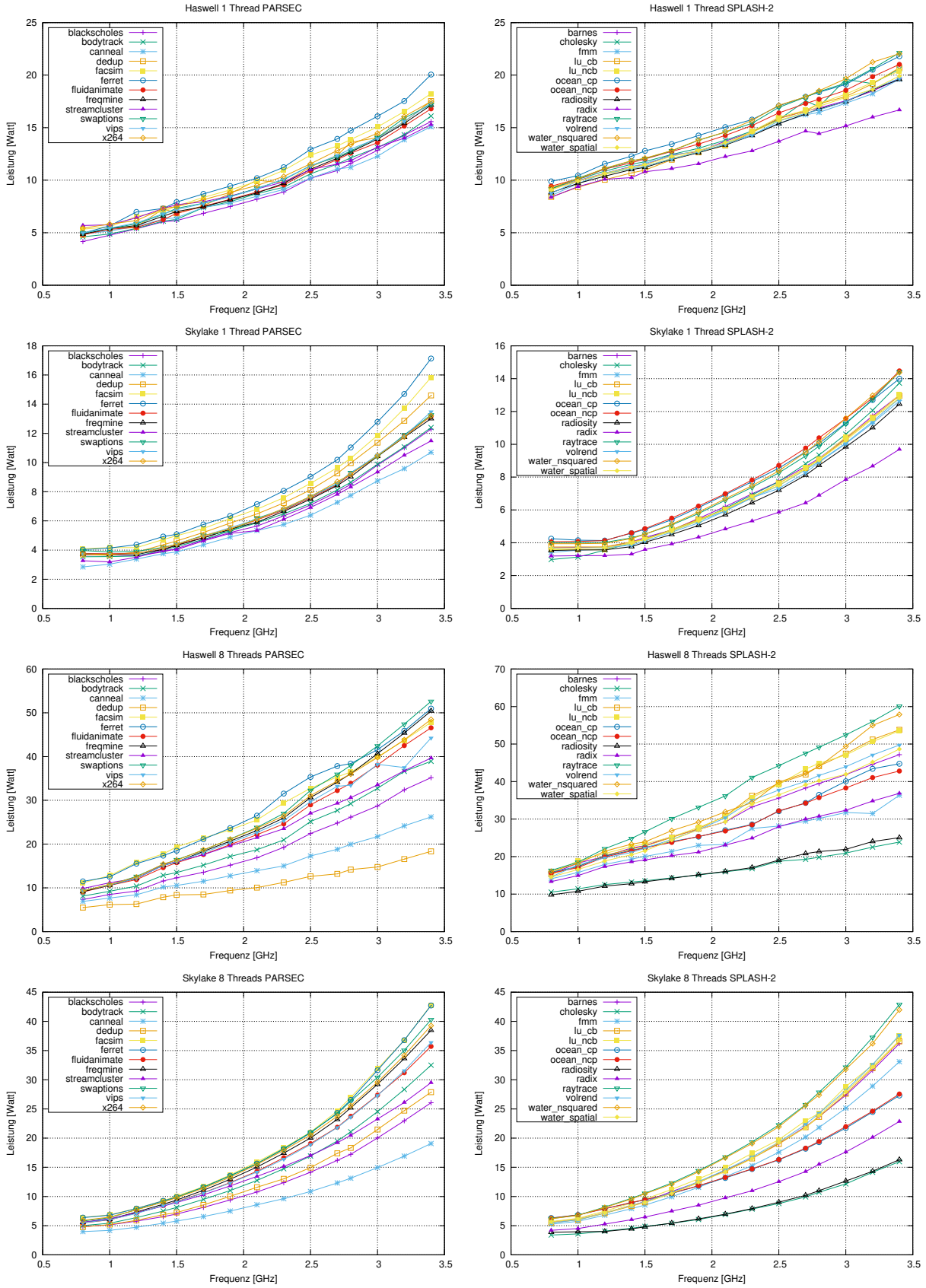

Abb. 3.3: Leistungsaufnahme (in Watt) verschiedener PARSEC (linke Spalte) und SPLASH-2 (rechte Spalte) Benchmarks mit einem (Reihen 1 und 2) bzw. acht (Reihen 3 und 4) Threads für einen Intel i7-6700 Skylake bzw. i7-4770 Haswell Prozessor mit verschiedenen Frequenzen.

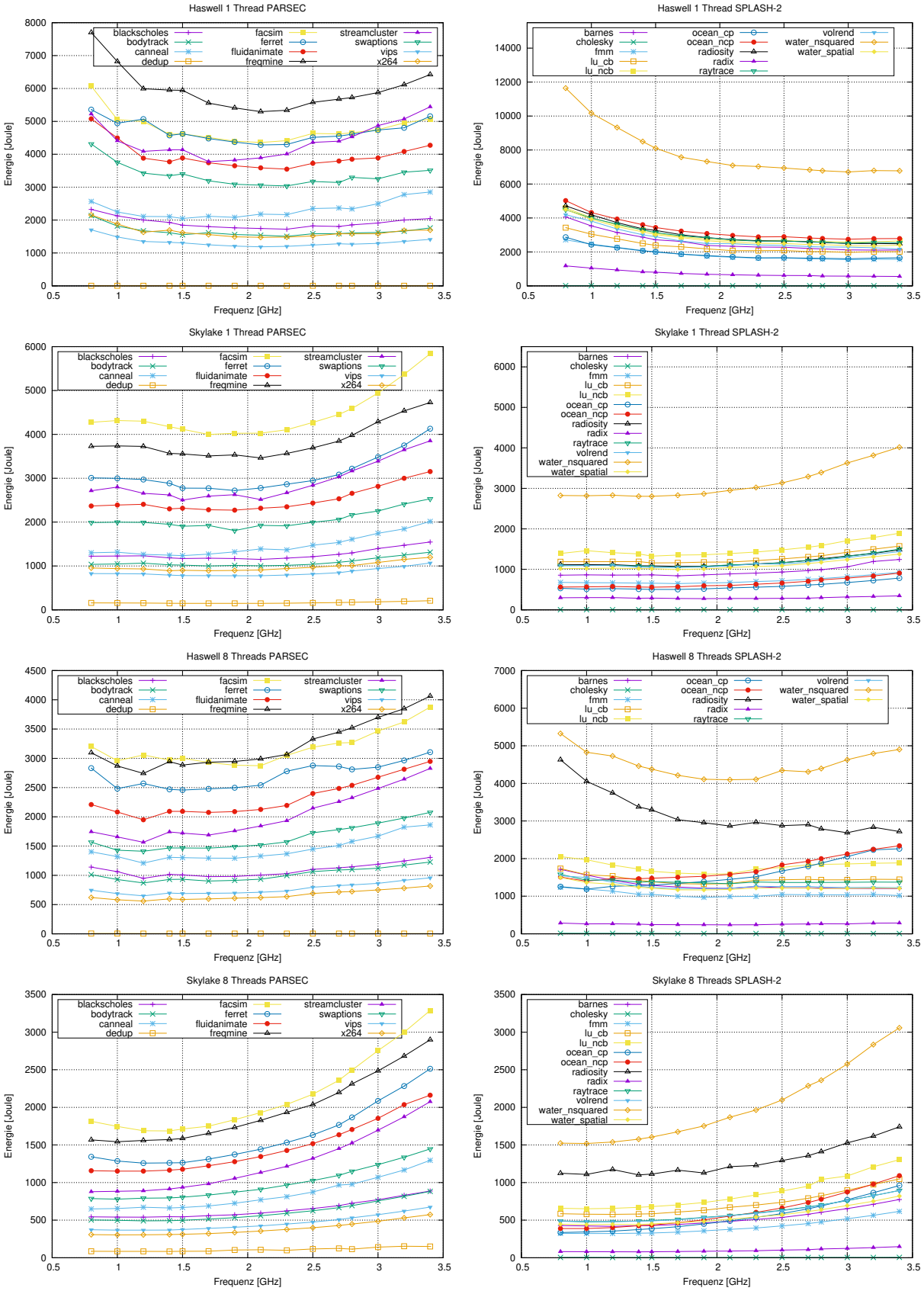

Abb. 3.4: Energieverbrauch in Joule verschiedener PARSEC (linke Spalte) und SPLASH-2 (rechte Spalte) Benchmarks mit einem (Reihen 1 und 2) bzw. acht (Reihen 3 und 4) Threads für einen Intel i7-6700 Skylake bzw. i7-4770 Haswell Prozessor mit verschiedenen Frequenzen.

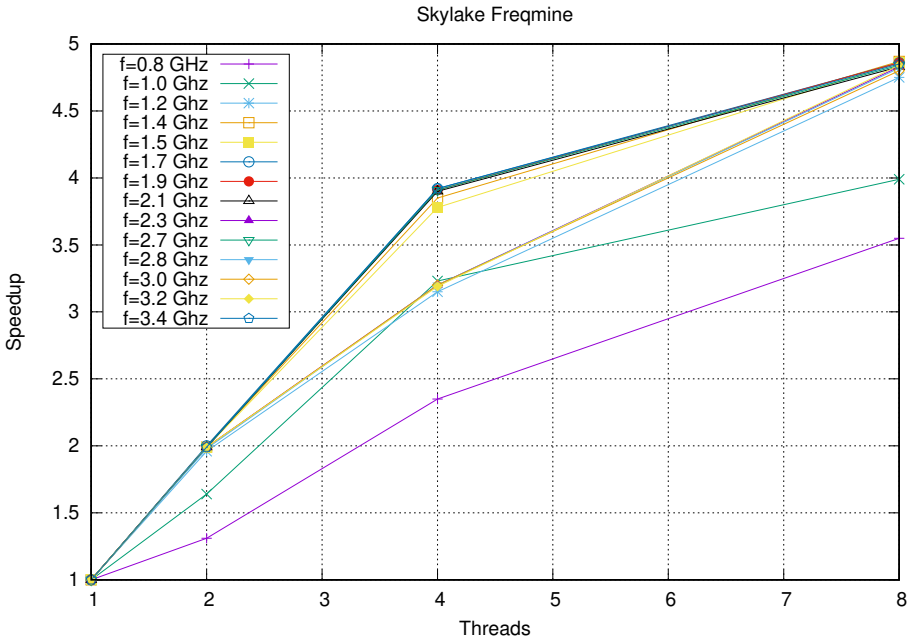

Abb. 3.5: Speedup des Freqmine Benchmarks für verschiedene Frequenzen auf einem Intel i7-6700 Skylake Prozessor

das Verwenden mehrerer Threads p einer CPU relativ zur sequenziellen Ausführungszeit bei gleichbleibender Frequenz gemindert wird. Ausgerechnet wird der Speedup durch eine Division der sequenziellen Laufzeit durch die parallele Laufzeit. Der Speedup variiert bei gleichbleibender Applikation auf unterschiedlichen Systemen beziehungsweise bei unterschiedlichen Applikationen auf gleichen Plattformen. Aber auch bei gleichbleibender Anwendung und Plattform jedoch bei unterschiedlichen Frequenzen variiert der Speedup. Somit ist es sinnvoll, den Speedup als Metrik in Abhängigkeit von zwei Parametern p und s abzubilden, was in einem Tupel von Speedup-Werten für jede Threadzahl $p \in \{1, \ldots, p_{max}\}$ resultiert:

$$S(p,s) = \frac{T_{seq}(s)}{T_{par}(p,s)}, \qquad \text{mit } s \in \{1, \ldots, s_{max}\}. \tag{3.3}$$

mit $T_{seq}(s) = T(1,s)$ und $T_{par}(p,s) = T(p,s)$. Somit ist der Speedup paralleler Programme eine Ansammlung an Kurven, bei der jede Kurve eine feste Frequenz beschreibt. Abbildung 3.5 zeigt als Beispiel diese Speedup-Werte für den Benchmark Freqmine aus der PARSEC Benchmarksuite auf einem Skylake-Prozessor. Andere Benchmark-Programme zeigen hier ein ähnliches Verhalten.

Die Problemgröße (native) ist konstant geblieben. Obwohl der Speedup eine weit verbreitete und angewendete Metrik ist, so ist der Bedarf nach Metriken mit ähnlichem

Ansatz groß, die auch das Energieverhalten berücksichtigen. Aus diesem Grund werden im weiteren Verlauf des Kapitels Metriken gezeigt und eingeführt, welche Performance und Energieeffizienz gleichzeitig evaluieren.

3.2.2 Runtime Reduction Factor

Die nächste Metrik betrachtet die Performance aus einer anderen Sichtweise. Sie erfasst die relativen Ersparnisse der Laufzeit $T(p, s)$ in Abhängigkeit von $s \in \{1, \ldots, s_{max}\}$. Dies entspricht den Werten aus Abbildung 3.5 bei einem fixen p, z.B. $p = 4$, und allen Punkten die von einer vertikalen Linie durch die x-Achse geschnitten werden. Wie bereits erwähnt, variiert die Ausführungszeit auf DVFS-Prozessoren, wenn sich die Frequenz ändert. Üblicherweise reduziert sich die Laufzeit mit sinkender Frequenz. Um zu zeigen, wie intensiv sich die Ausführungszeit ändert, dient eine Metrik, die dem Speedup sehr ähnlich ist, dem `Runtime Reduction Factor` R (auf deutsch in etwa Laufzeit Reduktionsfaktor).

$$R(p, s) = \frac{T_{par}(p, s)}{T_{par}(p, 1)}, \qquad \text{mit } p \in \{1, \ldots, p_{max}\}, \tag{3.4}$$

wobei $T_{par}(p, 1)$ der höchsten Frequenz f_{max} entspricht und $T_{par}(p, s)$ der verminderten Frequenz $f = f_{max}/s \leq f_{max}$, mit $s > 1$.

R beschreibt somit den relativen Laufzeitzuwachs, der entsteht, wenn eine kleinere Frequenz unter einer festen Threadzahl benutzt wird. Gleichung (3.4) kann auch für den sequenziellen Fall $T_{seq}(s) = T_{par}(1, s)$ verwendet werden. Tabelle 3.3 zeigt den typischen Verlauf von R am Beispiel des Blackscholes Algorithmus der PARSEC Benchmark Suite ausgeführt auf einem Skylake Prozessor. Die Frequenz $f_{max} = 3.4$ repräsentiert den nicht skalierten Fall $R(p, 1) = 1.0$. Im Allgemeinen wird $R(p, s) \geq 1$ erwartet, da die runterskalierte Laufzeit $T_{par}(p, s)$ üblicherweise größer ist als die nicht skalierte Laufzeit $T_{par}(p, 1)$. Jedoch wachsen die Werte von $R(p, s)$ nicht mit der selben Rate wie s. Diese Annahmen treffen auf Tabelle 3.3 zu. Bei einer Frequenz von 3.4 GHz liegt R immer bei 1. Eine etwas geringere Frequenz als 3.4 GHz hat kaum Auswirkungen auf R. Vor allem bei 2 Threads ist R sogar leicht unter 1. Ab 2.1 GHz wird R hier deutlich größer, vor allem bei 1, 4 und 8 Threads. Bei der kleinsten Frequenz 0.8 GHz ist R bei 8 Threads mit einem Wert von 3.738 am höchsten. Aber auch bei 4 Threads ist R mit 3.728 deutlich gewachsen und schon sehr nah am Wert von 4.25, was bedeuten würde, dass die Laufzeit und die Frequenz linear skalieren würde.

Tab. 3.3: Runtime Reduction R Factor für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	R Blackscholes			
Freq	1	2	4	8
0.8 GHz	2.748	2.134	3.728	3.738
1.0 GHz	2.782	2.128	2.740	3.362
1.2 GHz	2.774	1.777	2.804	2.821
1.4 GHz	2.426	1.524	2.008	2.403
1.5 GHz	2.259	1.421	1.905	2.245
1.7 GHz	1.998	1.260	1.623	1.985
1.9 GHz	1.788	1.126	1.457	1.776
2.1 GHz	1.612	1.017	1.317	1.612
2.3 GHz	1.474	0.929	1.466	1.474
2.5 GHz	1.357	0.856	1.107	1.352
2.7 GHz	1.256	0.792	1.249	1.258
2.8 GHz	1.212	0.764	1.207	1.219
3.0 GHz	1.132	0.713	1.131	1.130
3.2 GHz	1.063	1.068	0.864	1.059
3.4 GHz	1.000	1.000	1.000	1.000

3.2.3 Energy-Delay Product

Das `Energy-Delay Product` EDP (auf deutsch etwa Energieverzögerungsprodukt) ist definiert als die verbrauchte Energie einer Anwendung multipliziert mit deren Ausführungszeit [38, 52].
In Anlehnung an die Energie und parallele Ausführungszeit ist das EDP eine Funktion, die abhängig von den zwei Variablen p und f ist und beide Funktionen vereint. Verwendet man den Skalierungsfaktor s, so wird das EDP ausgedrückt als

$$EDP(p,s) = E(p,s) \cdot T(p,s)\ [Watt \cdot sec^2]. \tag{3.5}$$

EDP kombiniert Effekte der Ausführungszeit und des Energieverbrauchs für unterschiedliche Konfigurationen einer Anwendung und stellt die Umwandlung der Energie in Arbeit dar.
Die Bedeutung des EDP wird in der Betrachtung der Energieeffizienz sichtbar. Die Energieeffizienz einer Anwendung wird definiert als Performance (flops/sec) pro Energieeinheit ($W \cdot sec$) und wird gemessen in $flop/(W \cdot sec^2)$. Sind zwei EDP Werte EDP_1 und EDP_2 mit $EDP_1 < EDP_2$, respektive $1/EDP_1 > 1/EDP_2$, beide gemessen in $1/(W \cdot sec^2)$, bedeutet dies gleichzeitig eine bessere Energieeffizienz für kleinereEDP Werte. Ist ein EDP Wert also kleiner als ein anderer bedeutet dies mehr Leistung

pro Energieeinheit für diesen. Tabelle 3.4 zeigt die EDP-Werte für den Blackscholes Benchmark aus der PARSEC Benchmarksuite. Auffällig ist, dass die Frequenz für das Optimum, also der niedrigste Wert des EDP, mit zunehmender Anzahl an Threads abnimmt. So ist bei einem Thread das niedrigste EDP bei 3.4 GHZ mit 561945 Watt$\cdot$ sec^2 gemessen worden. Bei 2 und 4 Threads wird der niedrigste Wert bei 2.8 GHz gemessen. Eine noch niedrigere Frequenz für das niedrigste EDP zeigt sich bei 8 Threads bei 2.7 GHz und einem Wert von 17756. Über die Threads hinweg nimmt das EDP auch stark ab. So sinkt der optimale Wert bei einem Thread um das 31-fache durch Einsatz von 8 Threads. Zusätzlich zeigt Tabelle, dass das optimale EDP nicht unbedingt bei der höchsten Frequenz liegt. Ähnliche Resulatate zeigten auch die anderen Benchmarks.
Eine Variation des EDP ist das Energy-Delay^2 Product $EDDP$. Hier wird die Laufzeit noch einmal auf das EDP multipliziert[3]. Tabelle 3.4 zeigt das EDDP für den Blackscholes Benchmark. Hier erwartet man, dass die Laufzeit der bestimmende Faktor ist, und somit die beste Laufzeit auch gleichzeitig der niedrigste EDDP-Wert der entsprechenden Threadzahl ist. Dies bestätigt sich großteils auch. Bei einem und acht Threads ist der geringste EDDP-Wert jeweils bei 3.4 GHz und bei zwei und vier Threads bei 3.0 GHz und 3.2 GHz. Je höher die Potzenz, also der Anteil der Laufzeit, beim Energy-Delay^x ist, desto weniger Einfluss hat die Leistungsaufnahme darauf.
Das EDP und das EDDP überführen schlussendlich eine multikreterielle Bewertung (Die Energie und die Laufzeit) in eine 1-dimensional gewichtete Metrik.

3.2.4 Energy Speedup

Messungen zeigen, dass sich der Energieverbrauch für die Ausführung einer Anwendung für verschiedene Frequenzen und einer steigenden Anzahl an Rechenkernen unterschiedlich ändert. Der Grund sind unterschiedliche Laufzeiten und unterschiedliche Leistungsaufnahmen der verschiedenartigen Anwendungen. Deswegen ist es sinnvoll, die Quantität dieser Veränderung im Energieverbrauch $E(p, s)$ in einer Metrik festzuhalten. Analog zum (Laufzeit-)Speedup $S(p, s)$ wird der `Energy Speedup` ES (etwa Energie-Beschleunigung) definiert als

$$ES(p, s) = \frac{E(1, s)}{E(p, s)}, \qquad \text{mit } s \in \{1, \ldots, s_{max}\}. \tag{3.6}$$

ES drückt den relativen Unterschied des Energieverbrauchs aus, der entsteht, wenn p Threads anstelle von nur einem Thread für eine feste Frequenz verwendet werden. Somit wird die Einsparung der Energie mit einer steigenden Zahl an Kernen eingefangen. Der

[3] `http://www.eecs.harvard.edu/~dbrooks/cs246/cs246-lecture2.pdf`

Tab. 3.4: EDP für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	EDDP Blackscholes			
Freq	1	2	4	8
0.8 GHz	1222261	323467	121711	51578
1.0 GHz	1241180	325292	85888	43550
1.2 GHz	1241738	222844	70656	31351
1.4 GHz	1044422	184940	50171	27070
1.5 GHz	961496	172906	47952	25134
1.7 GHz	854602	148725	41892	22991
1.9 GHz	760345	140633	38234	20675
2.1 GHz	673449	130005	36400	19525
2.3 GHz	631358	124509	42055	18711
2.5 GHz	597488	119051	33246	18026
2.7 GHz	578022	117847	39900	17756
2.8 GHz	571383	115965	39842	18387
3.0 GHz	575058	117164	40515	17764
3.2 GHz	568696	123868	33010	17913
3.4 GHz	561945	130401	40707	18056

Tab. 3.5: EDDP für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	EDP Blackscholes			
Freq	1	2	4	8
0.8 GHz	1221038206	200188034	51760823	14518665
1.0 GHz	1255019707	200788819	26850153	11024799
1.2 GHz	1252156638	114883015	22604434	6661506
1.4 GHz	920861115	81757175	11494678	4898533
1.5 GHz	789495492	71267215	10422438	4248394
1.7 GHz	620667905	54357086	7758874	3437701
1.9 GHz	494123771	45931048	6356033	2765162
2.1 GHz	394546636	38345095	5470860	2370720
2.3 GHz	338219584	33551319	7033363	2077038
2.5 GHz	294673497	29566232	4197314	1835150
2.7 GHz	263878942	27074067	5687634	1682022
2.8 GHz	251716259	25712127	5486946	1687966
3.0 GHz	236618709	24235686	5226447	1511273
3.2 GHz	219701691	25473601	3255306	1427983
3.4 GHz	204256041	26609633	4644188	1359775

Tab. 3.6: ES für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	ES Blackscholes			
Freq	1	2	4	8
0.8	1.00	1.61	1.86	2.25
1.0	1.00	1.62	1.88	2.27
1.2	1.00	1.65	1.93	2.28
1.4	1.00	1.59	1.84	2.17
1.5	1.00	1.56	1.80	2.15
1.7	1.00	1.62	1.80	2.10
1.9	1.00	1.52	1.73	2.05
2.1	1.00	1.46	1.61	1.93
2.3	1.00	1.43	1.59	1.89
2.5	1.00	1.42	1.57	1.85
2.7	1.00	1.38	1.54	1.83
2.8	1.00	1.39	1.52	1.79
3.0	1.00	1.38	1.52	1.81
3.2	1.00	1.37	1.50	1.77
3.4	1.00	1.36	1.47	1.74

kleinste Wert für $ES(p, s)$ für ein festes s zeigt die Anzahl p für die der Energieverbrauch am geringsten zunimmt im Vergleich zur sequenziellen Ausführung. Wenn man die gemessenen Werte anschaut, wird deutlich, dass der Energieverbrauch sinkt, wenn die Anzahl an Threads steigt. Das gilt zumindest solange das Programm einen Nutzen aus mehr Ressourcen erzielt. Tabelle 3.6 zeigt den ES für den Benchmark Blackscholes auf einem Skylake Prozessor. Hier bestätigen sich die Annahmen. Bei einem Thread ist der Wert immer eins, da dies die Referenz ist. Der Energieverbrauch wird immer größer mit steigenden Threads und so steigert sich auch ES bis hin zu acht Threads. Auffällig ist, dass die Werte für ES bei geringeren Frequenzen höher sind. Bei niedrigen Frequenzen ist aber auch der Energieverbrauch am höchsten und so profitiert die Anwendung sehr stark von einer Laufzeitreduzierung. Der höchste Wert (2.25) ist bei niedrigster Frequenz und höchster Threadzahl erreicht und der niedrigste Wert (1.36) bei höchster Frequenz und niedrigster Threadzahl (ausgenommen aller Werte bei einem Thread).

3.2.5 Energy Reduction Factor

Betrachtet man eine sich verändernde Frequenz und den Einfluss auf den Energieverbrauch, beschreibt der `Energy Reduction Factor` ER (deutsch: Energie Reduktionsfaktor) die relative Differenz zwischen den Energieverbräuchen einer niedrigeren Frequenz $f = f_{max}/s$ und f_{max} für eine feste Anzahl an Threads. Definiert wird ER als

Tab. 3.7: ER für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	ER Blackscholes			
Freq	1	2	4	8
0.8	1.000	1.000	1.000	1.000
1.0	0.997	0.999	1.009	1.005
1.2	0.994	1.015	1.032	1.009
1.4	1.033	1.016	1.023	0.996
1.5	1.045	1.013	1.015	0.999
1.7	1.040	1.044	1.006	0.971
1.9	1.012	0.972	0.952	0.956
2.1	1,064	0.964	0.925	0.916
2.3	1.038	0.919	0.890	0.874
2.5	1.010	0.887	0.854	0.831
2.7	0.966	0.829	0.800	0.785
2.8	0.943	0.813	0.774	0.750
3.0	0.875	0.751	0.714	0.705
3.2	0.831	0.705	0.669	0.655
3.4	0.791	0.665	0.627	0.614

$$ER(p,s) = \frac{E(p,s)}{E(p,1)}, \qquad \text{mit } p \in \{1,\ldots,p_{max}\}\ . \tag{3.7}$$

Dementsprechend ist ER vergleichbar mit ES, mit dem Unterschied, dass ER die Frequenz variiert und die Threadanzahl fest bleibt. Für $s = 1$ ist der Wert für ER immer 1, also $ER(p,1) = 1$. Wenn für größere s der Wert für ER kleiner als 1 ist, also $ER(p,s) < 1$, bedeutet dies, dass weniger Energie verbraucht wurde als bei der maximalen Frequenz f_{max}. Für eine gegebene Anwendung und einem festen p bedeutet ein Minimum in $ER(p,s)$ ein Minimum des Energieverbrauchs für p. Tabelle 3.7 zeigt beispielhaft die Werte für ES für den Blackscholes Benchmark auf einem Skylake Prozessor. Bei 800 Mhz ist dieser immer bei 1, denn dieser Wert gilt als Referenz für die jeweiligen Threads. Bei niedrigen Frequenzen ist der Wert immer nah an 1 und nimmt bei höheren Frequenzen ab und fällt deutlich unter 1, oft sogar auf 0.6. Hier macht sich wieder die typische flache U-Form des Energieverbrauchs bemerkbar. Sind die ER-Werte kleiner als 1, dann sinkt der Energieverbrauch, sind sie größer, dann steigt der Energieverbrauch. Beim Skylake-Prozessor hat man bei niedrigeren Frequenzen den niedrigsten Energieverbrauch, der dann exponenziell zunimmt.

3.2.6 Energy per Speedup

Die Metrik `Energy per Speedup` EPS (Energie pro Speedup) stellt den Energieverbrauch in Relation zum (Laufzeit-) Speedup für eine Anwendung dar. Er wird definiert

als

$$EPS(p,s) = \frac{E(p,s)}{S(p,s)} \qquad [J]. \tag{3.8}$$

EPS zeigt, wie viel Energie man investieren muss, um einen bestimmten Speedup zu erreichen. Genauer gesagt den Speedup, den das Programm durch Einsatz zusätzlicher Threads erreicht. Das EPS bei einem Thread ist entsprechend der Energieverbrauch bei einem Thread selbst. Kleine Werte von EPS deuten auf eine effiziente Nutzung der Energie für parallele Programme hin, wenn mehr Ressourcen aufgrund einer schnelleren Ausführungszeit genutzt werden. Interessanterweise ist das EPS stark verwandt mit dem EDP aus Gleichung 3.5. Denn

$$EPS(p,s) = \frac{EDP(p,s)}{T_{seq}(s)}$$

normalisiert das EDP in Hinblick auf die anwendungsspezifische sequenzielle Laufzeit für selbige Frequenz. Tabelle 3.8 zeigt die EPS-Werte für den Blackscholes Benchmark auf einen Intel Skylake Prozessor. Bei einem Thread entspricht der EPS-Wert immer den Energiewert, da man nur durch eins teilt. Danach halbiert sich der Wert für das EPS in etwa von einem auf zwei Threads und von zwei auf vier Threads. Von vier auf acht Threads gibt es nochmal eine deutliche Reduktion des EPS. Die Abnahme des EPS von einem auf zwei Threads ist am höchsten, denn der Speedup der Laufzeit macht sich hier deutlich bemerkbar. Durch die zusätzliche Abnahme der Energie sinkt das EPS von einem auf zwei Threads um das 2.3 bis 2.4-fache. Von zwei auf vier Threads liegt der Faktor immer noch bei etwa 1.8 über alle Frequenzen. Von vier auf acht Threads hat man immer noch eine Senkung des EPS um etwa das 1.5-fache. Bei steigenden Frequenzen wird das EPS prozentual immer größer bei niedrigeren Threadzahlen. Abbildung 3.6 vergleicht den Blackscholes Benchmark auf der Skylake mit der Haswell Architektur. Beide Systeme verhalten sich hier sehr ähnlich. Der Haswell Prozessor braucht mehr Energie, zeigt aber über die Threads hinweg einen ähnlichen Verlauf. Einzig bei niedrigen Frequenzen zeigt der Haswell ein erhöhtes EPS. Wie schon bei der Analyse der Energieverläufe beobachtet, ist der Haswell Prozessor bei niedrigen Frequenzen energiehunglastiger als der Skylake Prozessor.

3.2.7 Power Speedup

Der `Power Speedup` $PS(p,s)$, also der Speedup für die Leistungsaufnahme, ist abgeleitet von ES und dem (Laufzeit-) Speedup. Definiert wird dieser über S und ES

Tab. 3.8: EPS für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	EPS Blackscholes			
Freq	1	2	4	8
0.8 GHz	1223.49	522.66	286.19	183.24
1.0 GHz	1227.49	527.00	274.74	172.04
1.2 GHz	1231.41	432.26	220.86	147.55
1.4 GHz	1184.56	418.35	218.99	149.59
1.5 GHz	1170.97	419.50	220.63	148.70
1.7 GHz	1176.71	406.92	226.19	153.76
1.9 GHz	1170.00	430.59	230.00	154.60
2.1 GHz	1149.51	440.77	242.19	160.82
2.3 GHz	1178.57	462.05	251.47	168.56
2.5 GHz	1211.48	479.37	263.34	177.07
2.7 GHz	1266.15	512.96	279.91	187.46
2.8 GHz	1297.01	523.02	289.31	200.30
3.0 GHz	1397.57	566.42	314.07	208.82
3.2 GHz	1472.07	602.33	334.74	224.72
3.4 GHz	1546.02	639.04	356.82	239.76

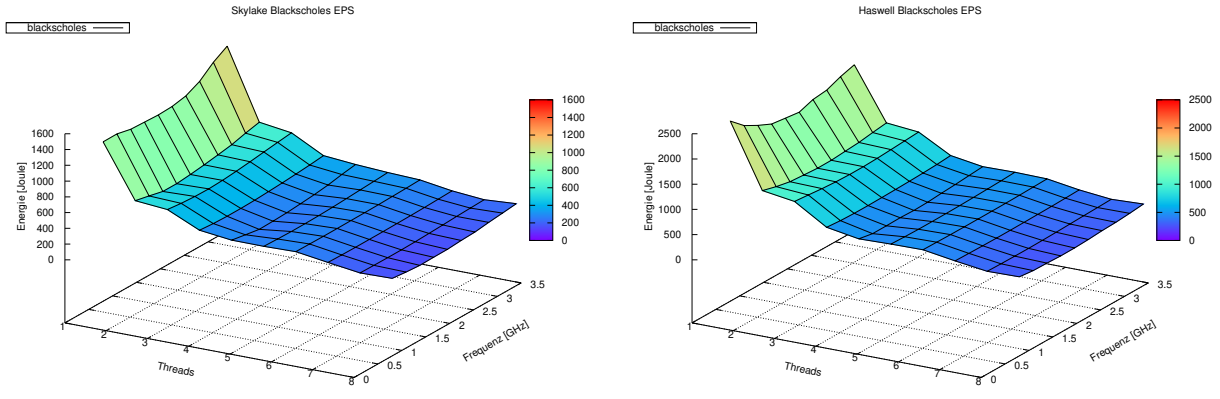

Abb. 3.6: Energy per Speedup EPS für Blackscholes (PARSEC) für unterschiedliche Frequenzen und Threads auf einem Intel i7-6700 Skylake Prozessor (links) und einem Intel i7-4770 Haswell Prozessor (rechts).

folgendermaßen:

$$PS(p,s) = \frac{ES(p,s)}{S(p,s)}. \tag{3.9}$$

Beim Umformen der Formel 3.9 wird PS auf

$$\begin{aligned} PS(p,s) &= \tfrac{E(1,s)}{E(p,s)} \cdot \tfrac{T_{par}(p,s)}{T_{seq}(s)} \\ &= \tfrac{E(1,s)}{T_{seq}(s)} \cdot \left(\tfrac{E(p,s)}{T_{par}(p,s)}\right)^{-1} \\ &= \tfrac{P(1,s)}{P(p,s)}, \end{aligned} \tag{3.10}$$

reduziert und somit wird die Laufzeit gekürzt. $PS(p,s)$ ist dimensionslos und stellt die relative Differenz der Leistungsaufnahme einer Anwendung zwischen paralleler p und sequenzieller Ausführung bei fester Frequenz dar. Da es einen Zuwachs in der Leistungsaufnahme bei allen Anwendungen bei steigender Threadzahl gibt, ist PS immer kleiner als 1. Abbildung 3.7 (linke Seite) zeigt PS für den Blackscholes Benchmark aus der PARSEC Suite. Auffällig ist hier die stufenweise Reduzierung des PS bis vier Threads. Ab vier Threads ist PS relativ konstant. Der Benchmark verwendet zuerst die vier physischen Kerne des Prozessors und ab fünf Threads auch die restlichen logischen Kerne (Hyperthreads). Bei den Messungen fällt auf, dass die logischen Kerne unter Volllast nur eine minimal höhere Leistungsaufnahme zeigen. Trotzdem profitiert die Laufzeit davon. Bei einem Volllasttest wurde auch eine solche Stufenbildung beobachtet. Da die Skala sehr fein ist, sind die Verläufe auf der Achse der Frequenzen fast vernachlässigbar klein. Der kleinste PS-Wert ist bei vier Threads zu finden.

3.2.8 Power Increase Factor

Für eine gemeinsame Bewertung von Ausführungszeit und Energieverbrauch eignet sich nachfolgende Metrik sehr gut. Betrachtet man die Leistungsaufnahme einer Anwendung bei fester Frequenz und variiert die Anzahl der Threads zur Berechnung, so nimmt die Leistungsaufnahme typischerweise zu. Um diesen Effekt quantitativ festzuhalten, eignet sich der Power Increase Factor $PI(p,s)$ (zu deutsch etwa Leistungsaufnahmezuwachs Faktor), der als

$$PI(p,s) = \frac{P(p,s)}{P(1,s)}, \qquad s \in \{1,\ldots,s_{max}\}\ , \tag{3.11}$$

definiert ist. In der Tat ist PI die Inverse zum Power Speedup PS. Abbildung 3.7 (rechte Seite) zeigt PI für den Blackscholes Benchmark auf einem Skylake Prozessor. Man sieht, dass $PI(p,s)$ keine lineare Funktion von p ist, was bedeutet, dass PI nicht

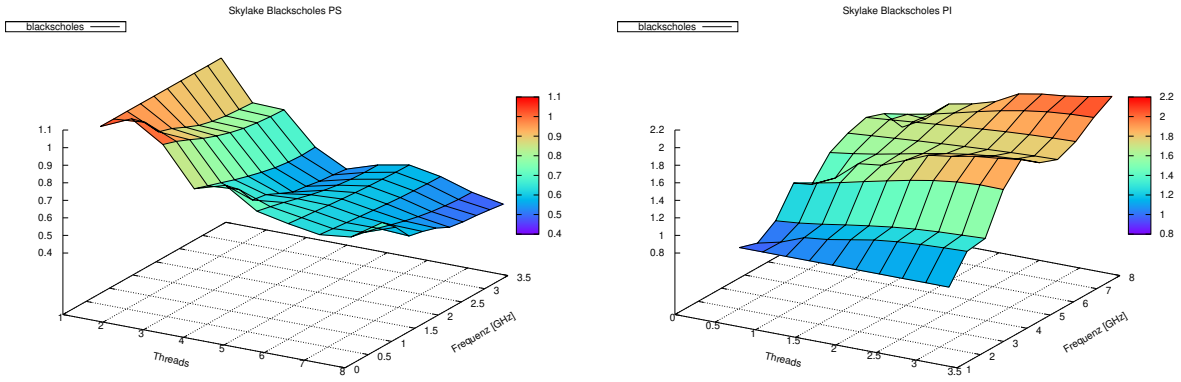

Abb. 3.7: Power Speedup PS (links) und Power Increase Faktor PI (rechts) für Blackscholes für unterschiedliche Frequenzen und Threads auf Skylake Architektur.

proportional zur Anzahl der Threads ist. Jedoch nimmt der Wert von PI mit p für eine feste Frequenz f zu. Vergleicht man unterschiedliche Frequenzen, variiert dieser Wert mit der Tendenz zu größeren PI Werten bei höheren Frequenzen.
Ein Vergleich zwischen PI und dem (Laufzeit-) Speedup zeigt eine direkte Korrelation zwischen Laufzeit und Leistungsaufnahme. Somit ist der (Laufzeit-) Speedup ein guter Indikator dafür, ob für eine gegebene Anwendung einen signifikanter Zuwachs in der Leistungsaufnahme für eine größere Anzahl an Threads erwartet wird. Dies führt direkt zur nächsten Metrik, dem RPI.

3.2.9 Relative Power Increase Factor

Um genauer die Korrelationen zwischen dem (Laufzeit-) Speedup und dem Zuwachs der Leistungsaufnahme zu untersuchen, behilft man sich mit dem `Relative Power Increase Factor` RPI (Relativen Leistungszuwachs Faktor). Die qualitative Abhängigkeit beider wird als

$$RPI(p,s) = \frac{PI(p,s)}{S(p,s)} \tag{3.12}$$

definiert. RPI beschreibt, um wie viel die Leistungsaufnahme einer Anwendung pro (Laufzeit-)Speedup zunimmt, wenn die Anzahl an Threads entsprechend steigt. Kleine RPI Werte implizieren eine effizientere Umsetzung von Leistungsaufnahme. Das bedeutet, dass die erhöhte Stromzufuhr im Vergleich zum Performancezuwachs gering ist, wenn die Anzahl an ausführenden Threads gesteigert wird. Würde sich die Leistungsaufnahme linear zum (Laufzeit-)Speedup verhalten, wäre der RPI Wert um etwa 1 schwankt. Für die meisten gemessenen Benchmarks war $RPI < 1$, was bedeutet, dass der Leistungsmehraufwand kleiner ist als der Speedup. Der $RPI(p,s)$ fängt die Effekte von den

Tab. 3.9: RPI für Blackscholes (PARSEC) auf einem Intel i7-6700 Skylake Prozessor

	RPI Blackscholes			
Freq	1	2	4	8
0.8	1.00	0.62	0.54	0.44
1.0	1.00	0.62	0.53	0.44
1.2	1.00	0.61	0.52	0.44
1.4	1.00	0.63	0.54	0.46
1.5	1.00	0.64	0.55	0.47
1.7	1.00	0.62	0.56	0.48
1.9	1.00	0.66	0.58	0.49
2.1	1.00	0.68	0.62	0.52
2.3	1.00	0.70	0.63	0.53
2.5	1.00	0.71	0.64	0.54
2.7	1.00	0.72	0.65	0.55
2.8	1.00	0.72	0.66	0.56
3.0	1.00	0.72	0.66	0.55
3.2	1.00	0.73	0.67	0.56
3.4	1.00	0.74	0.68	0.57

unabhängigen Variablen p und s auf und zeigt den Einfluss beider. Der Energieverbrauch und die parallele Ausführungszeit werden hier zusammen ausgewertet. Tabelle 3.9 zeigt den Verlauf des RPI am Beispiel des Backscholes Benchmarks aus der PARSEC Suite. Bei einem Kern ist dieser immer bei 1. Bei zunehmender Threadzahl nimmt dieser etwas ab. Aber auch bei steigender Frequenz nimmt der RPI zu, jedoch deutlich weniger als durch Steigerung der Threadzahl. Relativ gesehen steigt die Leistungsaufnahme somit stärker als der Speedup. Dies ist auf die Leistungsaufnahme zurückzuführen: Diese hat eine polynomiale Steigung, wohingegen die Laufzeit bei optimaler Parallelität höchstens linear ist. Auch auffällig ist, dass der RPI bei niedrigen Frequenzen kleiner sind als bei höheren.

3.3 Auswertung der Ergebnisse

Verschiedene Messungen wurden auf einem Skylake und Haswell Prozessor erstellt. Details der beiden Systeme finden sich in Anhang A.1 und A.2. Die wichtigsten Ergebnisse sind in den Tabellen 3.11 bis 3.12 zusammengefasst. Sowohl für die PARSEC als auch für die SPLASH-2 Benchmarks sieht man eine große Variation in der Laufzeit (Spalten 1 und 2) und im Energieverbrauch (Spalten 5 und 6) bei unterschiedlichen Frequenzen und einer unterschiedlichen Anzahl an Threads. Der maximal gemessene Energieverbrauch bei den PARSEC Benchmarks liegt bei dem Haswell System etwa 1.5- bis 1.8-fach höher als beim Skylake System. Bei den SPALSH-2 Bechnmarks ist der Unterschied

noch größer. Hier verbraucht der Haswell Prozessor fast doppelt, manchmal sogar bis zu dem dreifachen (siehe $ocean_{npc}$) an Energie als der Skylake Prozessor. Diese beiden Parameter (Anzahl an Threads und die CPU-Frquenz) besitzen also ein großes Potenzial den Energieverbrauch zu senken, wenn eine gute Kombination aus beiden gefunden wird. Beim Haswell System bewegt sich das Energieminimum zwischen 1.2 GHz und 2.3 Ghz für die PARSEC Benchmarks (Tabelle 3.10) und zwischen 0.8 GHz und 3.2 GHz für die SPLASH-2 Benchmarks (Tabelle 3.11). Bei 8 Threads liegen die Frequenz für den geringsten Energieverbrauch bis auf eine Ausnahme (Cholesky, SPLASH-2) deutlich niedriger als bei einem Thread, siehe Spalten 6 und 7. Der Cholesky Benchmark profitiert durch den Einsatz mehrerer Threads auch nicht und hat eine sehr niedrige absolute Laufzeit, was den Energieverbrauch sehr klein macht. Bei den SPLASH-2 Benchmarks, also den speziell für HPC Anwendugen angepasste Benchmarks, zeigte sich eine höhere Variation im Energieverhalten für den Haswell (Tabelle 3.12). Für die Skylake Architektur liegen entsprechende Werte zwischen 0.8 GHz und 2.1 GHz für die PARSEC Benchmarks (Tabelle 3.11) vor. Ähnlich sieht es für die SPLASH-2 Benchmarks (Tabelle 3.13) aus. Auch hier sind niedrigere Frequenzen für eine höhere Anzahl an Threads für einen niedrigen Energieverbrauch zu verbuchen.
Für die meisten Anwendungen kann durch den Einsatz mehrerer Threads ein signifikanter Speedup erreicht werden. Der erreichte Speedup unterscheidet sich nur geringfügig bei unterschiedlichen Frequenzen. Er ist meistens etwas höher bei höheren Frequenzen, was in den Spalten 3 und 4 der jeweiligen Tabellen 3.10 bis 3.13 notiert ist. Bei dem Skylake System ist der Speedup geringfügiger höher. Bei den SPLASH-2 Benchmarks kommt es öfter vor, dass bei einer geringeren Frequenz ein höherer Speedup erreicht wird.

Betrachtet man den Energy Speedup ES in den Spalten 9 und 10, bestätigt sich, dass der Wert für die meisten Anwendungen größer als 1 ist. Somit sinkt der Energieverbrauch bei einer höheren Anzahl an Threads und die Benchmarks steigen durch den Einsatz mehrer Threads in ihrer Effizienz, da der Laufzeitgewinn höher ist, als die zusätzliche Leistungsaufnahme die Energiebilanz verschlechtert. Es gibt durchaus Ausnahmen: Diese haben aber generell einen niedrigen Energieverbrauch und einen kaum nennenswerten Speedup wie der bereits erwähnte Cholesky Benchmark aus der SPLASH-2 Sammlung. Für die meisten Anwendungen nimmt ES mit zunehmender Frequenz ab. Das bedeutet, dass mit zunehmender Frequenz die Energieeinsparung durch Zunahme der Threads kleiner ist als für niedrigere Frequenzen. Hier gibt es weder zwischen Haswell und Skylake noch zwischen PARSEC und SPLASH-2 einen nennenswerten Unterschied.
Die RPI Werte in den letzten beiden Spalten decken sich nahezu auf beiden Prozessoren.

Tab. 3.10: Auswertung verschiedener PARSEC Benchmarks auf einem Intel i7-4770 Haswell Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads

	Laufzeit		**L-Speedup**		**Energie**		**Opti**		**ES (p=8)**		**EPS**		**RPI (p=4)**	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	f=0.8	f=3.4	min	max	min	max
blackscholes	37.12	558.67	3.60	3.54	948.56	2323.36	2.3	1.2	2.04	1.57	260.22	2323.36	0.52	0.68
bodytrack	31.52	463.37	3.72	3.47	868.69	2134.53	2.3	1.2	2.11	1.44	234.26	2134.53	0.56	0.77
canneal	71.00	527.51	2.59	2.66	1209.09	2851.98	1.5	1.2	1.83	1.53	460.53	2851.98	0.64	0.76
dedup	6.25	51.30	2.77	2.09	182.23	334.58	2.3	2.1	1.69	1.19	68.04	397.87	0.68	0.84
fluidanimate	63.30	1054.09	4.32	4.02	1949.83	5077.73	2.3	1.2	2.30	1.45	453.43	5077.73	0.50	0.77
freqmine	80.63	1581.32	4.67	4.65	2742.39	7700.87	2.1	1.2	2.49	1.58	586.20	7700.87	0.46	0.71
streamcluster	71.25	918.56	5.19	4.99	1564.56	5222.48	1.7	1.2	3.00	1.93	315.75	5442.64	0.50	0.81
swaptions	39.46	867.48	5.18	5.18	1411.24	4313.39	2.3	1.2	2.76	1.69	269.41	4313.39	0.44	0.69
vips	21.63	341.41	4.30	3.76	648.34	1703.58	2.1	1.2	2.28	1.48	151.61	1703.58	0.48	0.70
x264	16.87	396.76	5.95	5.75	560.11	2155.81	2.1	1.2	3.48	2.08	94.92	2155.81	0.33	0.53

Tab. 3.11: Auswertung verschiedener PARSEC Benchmarks auf einem Intel i7-6700 Skylake Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads

	Laufzeit		**L-Speedup**		**Energie**		**Opti**		**ES (p=8)**		**EPS**		**RPI(p=4)**	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	f=0.8	f=3.4	min	max	min	max
blackscholes	34.02	333.86	2.97	3.70	539.53	1546.02	2.1	1.2	2.25	1.74	147.55	1546.02	0.52	0.68
bodytrack	27.15	294.10	2.91	3.91	490.99	1314.92	1.7	1.2	2.06	1.49	126.06	1314.92	0.53	0.73
canneal	67.94	456.88	2.80	2.77	650.01	2015.27	1.5	0.8	2.00	1.55	232.10	2015.27	0.58	0.72
dedup	5.38	43.14	2.36	2.65	84.86	207.75	1.7	1.0	1.83	1.39	30.00	207.75	0.59	0.76
fluidanimate	60.51	636.68	2.97	3.95	1151.72	3151.76	1.9	1.0	2.04	1.46	282.69	3151.76	0.53	0.77
freqmine	75.31	1011.15	3.55	4.83	1543.42	4726.40	2.1	1.0	2.38	1.63	322.56	4726.40	0.48	0.69
streamcluster	70.36	878.20	5.29	4.77	878.56	3851.71	1.5	0.8	3.09	1.86	144.19	3851.71	0.44	0.69
swaptions	35.91	516.00	3.68	5.31	779.57	2529.71	1.9	1.0	2.53	1.75	149.48	2529.71	0.47	0.69
vips	18.54	221.89	3.28	4.27	367.38	1067.31	2.1	1.0	2.22	1.58	84.57	1067.31	0.48	0.68
x264	14.61	255.89	4.92	6.20	304.57	1197.61	1.7	1.0	3.11	2.09	48.37	1197.61	0.37	0.53

Einzig die Anwendungen mit einer sehr geringen Laufzeit haben RPI Werte höher als 0.5. Nur der x264 PARSEC Benchmark fällt mit Werten zwischen 0.33 und 0.37 aus diesem Raster. Gleicher Benchmark erzieht auch den höchsten Speedup.

3.4 Vergleich verschiedener Architekturen

Vergleicht man den den Haswell Prozessor und den Skylake Prozessor, so erreicht der Skylake Prozessor aufgrund der Aktualität kürzere Ausführungszeiten, was sich in den minimalen und maximalen Laufzeiten in Tabellen 3.11 bis 3.12 in den ersten beiden Spalten niederschlägt. Unabhängig davon war die Skylake Architektur bei allen Frequenzen und Threads auch sonst schneller, was hier nicht gezeigt ist. Noch signifikanter ist die deutlich geringere Leistungsaufnahme während der Berechnung des Skylake Prozessors, was zu einer besseren Energieeffizienz führt. Der Skylake Prozessor hat sein Optimum in Hinsicht auf minimalen Energieverbrauch bei niedrigeren Frequenzen als die Haswell

Tab. 3.12: Auswertung verschiedener SPLASH-2 Benchmarks auf einem Intel i7-4770 Haswell Prozessor: Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads

	Laufzeit		**L-Speedup**		**Energie**		**Opti**		**ES (p=8)**		**EPS**		**RPI (p=4)**	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	f=0.8	f=3.4	min	max	min	max
barnes	25.29	444.98	4.34	4.21	1067.37	2477.43	2.3	2.1	1.99	1.61	251.33	2477.43	0.51	0.72
cholesky	0.35	1.51	1.12	1.13	6.18	9.68	2.3	3.0	0.90	0.92	5.71	8.68	1.00	1.05
fft	148.98	595.58	2.66	2.29	1809.84	4830.14	0.8	0.8	1.82	1.54	373.47	4830.14	0.69	0.83
fmm	24.75	317.68	3.54	2.52	831.06	1670.93	2.3	1.4	1.76	1.32	241.21	1670.93	0.55	0.74
lu_cb	26.32	408.42	3.61	3.61	1061.13	2254.42	3.0	1.7	1.65	1.28	286.33	2254.42	0.56	0.74
lu_ncb	34.70	502.74	3.88	3.69	1407.10	2693.82	2.1	1.9	1.63	1.27	375.23	2675.30	0.70	1.09
ocean_cp	49.72	289.87	3.64	1.51	953.77	2106.09	2.3	1.0	1.98	0.66	269.88	1943.71	0.51	1.43
ocean_ncp	54.00	537.91	5.62	2.50	1113.04	3426.67	2.7	1.0	2.92	1.12	208.33	3426.67	0.48	1.02
radiosity	102.44	534.97	1.12	1.16	2065.73	3002.57	3.0	2.1	1.00	0.91	395.28	3002.57	0.95	1.17
radix	7.94	139.99	6.50	4.21	203.03	779.66	3.2	1.7	3.47	1.75	32.87	779.66	0.48	0.70
raytrace	22.90	489.52	5.07	5.01	1158.39	2938.06	2.3	1.4	2.33	1.66	229.44	2938.06	0.52	0.71
volrend	24.73	480.42	4.55	4.55	1031.52	2826.36	2.3	1.7	2.37	1.66	224.74	2826.36	0.50	0.70
w_nsquared	82.97	1252.57	3.77	3.64	3604.37	8221.01	2.3	1.9	1.91	1.26	446.20	8221.01	0.55	0.88
w_spatial	24.75	490.37	4.72	4.85	992.49	2889.17	3.2	1.7	2.53	1.79	208.22	2889.17	0.46	0.64

Tab. 3.13: Auswertung verschiedener SPLASH-2 Benchmarks auf einem Intel i7-6700 Skylake Prozessor:Laufzeit in Sekunden, (Laufzeit-)Speedup für 0.8 und 3.4 GHz, Energieverbrauch in Joule (alle Threads), Optimale Frequenz für einen und acht Threads, Energy Speedup ES, Energy per Speedup EPS und Relativ Power Increase RPI für 4 Threads

	Laufzeit		**L-Speedup**		**Energie**		**Opti**		**ES (p=8)**		**EPS**		**RPI (p=4)**	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	f=0.8	f=3.4	min	max	min	max
barnes	21.39	232.84	3.05	4.48	417.23	1244.07	1.7	1.2	1.55	1.61	100.79	1244.07	0.52	0.68
cholesky	0.32	1.32	1.10	1.13	3.39	5.14	1.7	1.5	0.96	0.96	3.10	4.95	0.99	1.02
fft	23.55	90.83	1.50	1.65	258.05	580.93	1.5	1.5	1.23	1.15	163.02	580.93	0.78	0.87
fmm	18.64	198.87	3.21	3.88	322.22	926.49	1.7	1.0	2.09	1.50	81.17	926.49	0.53	0.77
lu_cb	28.65	317.63	2.97	4.24	576.70	1573.95	1.5	1.2	2.03	1.50	137.53	1573.95	0.51	0.70
lu_ncb	34.78	407.64	3.36	4.18	652.01	1892.70	1.5	1.0	2.11	1.45	145.82	1892.70	0.53	0.96
ocean_cp	34.98	126.45	2.36	1.59	335.16	784.12	1.5	0.8	1.59	0.82	125.37	784.12	0.66	1.19
ocean_ncp	39.60	139.62	2.24	1.58	388.21	1103.04	0.8	1.0	1.45	0.83	146.66	907.70	0.71	1.22
radiosity	106.78	317.35	1.13	1.21	1073.33	1740.68	1.7	1.4	1.00	0.85	943.07	1559.80	0.99	1.11
radix	6.48	94.27	4.77	5.52	78.92	346.79	1.9	1.4	3.63	2.34	11.90	346.79	0.42	0.61
raytrace	20.88	273.69	3.46	4.98	477.11	1498.99	1.7	1.0	2.24	1.67	97.71	1498.99	0.52	0.70
volrend	23.71	298.27	3.33	4.80	480.75	1437.68	1.7	1.0	2.23	1.61	102.06	1437.68	0.51	0.69
w_nsquared	72.83	715.78	2.91	3.84	1519.57	4015.32	1.5	1.0	1.85	1.31	394.11	4015.32	0.59	0.85
w_spatial	22.35	279.36	3.56	4.80	439.87	1376.12	1.7	1.0	2.29	1.68	89.69	1376.12	0.47	0.63

Architektur. Betrachtet man die Leistungsaufnahmen beider Systeme in Tabelle 3.3, sieht man den geringeren Leistungshunger bei zunehmender Frequenz. Somit hat eine Reduktion der Frequenz eine größere Auswirkung auf den Skylake als auf den Haswell, was den Energieverbrauch enorm beeinflusst.

Neben der x86-Architekur beider Intel System wurden einige PARSEC Benchmarks auch auf einer ARM Architektur ausgeführt. Es handelt sich dabei um ein Odroid XU-4 Board mit einem Prozessor der big.LITTLE Architektur. Als Prozessor arbeitet hier ein Samsung Exynos 5422 Octa-Core, der aus einem Cortex-A15 mit 2 Ghz und 4 Kernen und einem Cortex-A7 mit 1,4 GHz und auch 4 Kernen besteht. Durch die beiden unterschiedlichen Prozessorkerne sind trotz insgesamt 8 Kernen ab einer gewissen Last keine großen Speedups zu erwarten. Zu beobachten war jedoch, dass bei den Benchmarks die leistungsstärkeren A15 Kerne bevorzugt wurden. Eine genaue Beschreibung des XU4 befindet sich im Anhang A.5. Die Messung der Energie wurde mit einem zweiten XU4-Board durchgeführt, da die Abtastung der Leistung und gleichzeitige Berechnung der Benchmarks auf einem Board zu stärkeren Leistungseinbußen geführt haben. Als Beispiel dient hier der Blackscholes Benchmark. Die Daten für P, PI, RPI und EPS finden sich in den Tabellen 3.14 bis 3.17. Folgende Beobachtungen wurden dabei gemacht:

- Die Leistungsaufnahme steigt mit zunehmender Frequenz. Verglichen mit dem Skylake ist dieser Anstieg jedoch deutlich flacher. Abbildung 3.14 zeigt die Leistungsaufnahme auf dem ARM-Prozessor am Blackscholes Benchmark für verschiedene Frequenzen und Threads. Auch die maximale Leistungsaufnahme ist viel kleiner als auf den i7 Prozessoren. PI nimmt nur schleichend zu mit der Anzahl der Threads, wie Tabelle 3.16 zeigt und ist auch kleiner als PI auf den x86 Prozessoren. Bei allen Benchmarks blieb die Leistungsaufnahme deutlich unter 10 Watt. Somit hat die Leistungsaufnahme deutlich weniger Einfluss auf den Energieverbrauch als bei den x86 Systemen.
- Die Laufzeit der Benchmarks ist auf dem Odroid Board deutlich höher als auf den Intel Systemen. Dies führt in diesem Fall auch zu höheren Energieverbräuchen. Dies hat auch Einfluss auf EDP, und EPS: Diese sind deutlich höher als auf den i7 Systemen, siehe Tabelle 3.17 und 3.8.
- Der Runtime Reduction Factor R ist ähnlich auf allen Systemen bei gleichen Skalierungsfaktoren. Größere Abweichungen gibt es bei steigender Threadzahl.
- RPI ist relativ ähnlich zu den Intel Systemen, was man in Tabelle 3.15 ablesen kann. Die Werte werden kleiner, wenn die Betriebsfrequenz gemindert wird und die Anzahl der Threads zunimmt.

Alle diese Beobachtungen suggerieren, dass sowohl die etablierten, als auch die neu eingeführten Metriken gut geeignet sind, Ausführungen von multi-threaded Anwendungen auf verschiedenen Systemen auf verschiedenen Architekturen zu evaluieren und deren Verhalten in Hinsicht auf Frequenzskalierung und Parallelität zu studieren und grundlegende Unterschiede in deren Energie-Sensitivität einzufangen.

3.5 Verwandte Arbeiten

Energieeinsparung steht seit Jahren im Mittelpunkt der Forschung. Verschiedene Features wurden in Computersysteme eingebettet: Vom Smartphone bis hin zu großen Serversystemen [54, 27, 28]. Eine weit verbreitete und angenommene Technik ist das DVFS, ein Feature, das es ermöglicht Frequenzen eines Prozessors dynamisch zur Laufzeit, zu ändern, wie in [64] dokumentiert wird. Das Energy-Delay Product (EDP) wurde zum ersten Mal in [34] eingeführt um sowohl den Energieverbrauch, als auch die Ausführungszeit gleichzeitig darzustellen. Jedoch wurde in [15] diskutiert, dass es mit EDP nicht möglich ist, vorherzusagen, ob eine kleinere Frequenz oder eine bessere Energieeffizienz zu kleineren EDPs führt, wenn man verschiedene Varianten eines Programms betrachtet. Effekte des EDP und entsprechend der Varianten ED^n für ein positives n werden zusätzlich in [39] diskutiert. Um Energie und Performance separat zu betrachten, wenn verschiedene Programmversionen miteinander verglichen werden sollen, wurde in [15] das *powerup* und das *greenup* als Metrik eingeführt. Diese beiden Metriken sind ähnlich zu den in diesem Kapitel neu eingeführten Metriken *Power Speedup* und *Energy Speedup*. Diese beiden Metriken werden zusammen mit dem Speedup benutzt, um in unterschiedliche Kategorien des Energieverbrauchs eingeordnet werden zu können. Die hier eingeführten Metriken bewegensich in eine andere Richtung, indem die Beziehungen zwischen Speedup, Energie und Leistung genauer untersucht werden. Im Zusammenhang mit Frequenzskalierungen definiert [30] den *power-aware Speedup*, welcher die Frequenz in den (Laufzeit-)Speedup miteinbezieht. Die Energie wird hier jedoch nicht betrachtet. Im Zusammenhang mit Scheduling-Regeln hat [46] die Metriken *Speedup per Watt (SPW)*, *Power per Speedup (PPS)* und *Energie per Target (EPT)* vorgestellt und das Task-Scheduling mit diesen Metriken evaluiert.

3.6 Schlussfolgerungen

In diesem Kapitel wurden verschiedene Metriken definiert: T, E, P, S, R, ES, ER, EDP, EPS, PS, PI und RPI. Zum Abschluss folgt eine kurze Beschreibung, wann

	P Blackscholes			
Freq	1	2	4	8
0.2	2.44	3.06	2.52	2.60
0.3	2.51	2.61	2.62	2.71
0.4	2.57	2.70	2.66	2.81
0.5	2.63	2.80	2.80	2.93
0.6	2.69	2.91	2.89	3.07
0.7	2.75	3.00	2.98	3.17
0.8	2.84	3.13	3.12	3.34
0.9	2.96	3.27	3.25	3.53
1.0	3.06	3.41	3.30	3.71
1.1	3.17	3.58	3.54	4.03
1.2	3.27	3.75	3.75	4.15
1.3	3.40	3.93	3.95	4.40
1.4	3.52	4.10	4.10	4.63
1.5	3.63	4.29	4.23	4.81
1.6	3.81	4.59	4.17	5.06
1.7	3.99	4.87	4.35	5.33
1.8	4.23	5.21	4.50	5.62
1.9	4.51	5.70	5.17	6.09
2.0	4.93	6.41	5.12	6.54

Tab. 3.14: *P* für Blackscholes auf XU4 für verschiedene Frequenzen und Threads.

	PI Blackscholes			
Freq	1	2	4	8
0.2	1.00	1.25	1.03	1.06
0.3	1.00	1.04	1.04	1.08
0.4	1.00	1.05	1.04	1.10
0.5	1.00	1.07	1.07	1.11
0.6	1.00	1.08	1.08	1.14
0.7	1.00	1.09	1.09	1.15
0.8	1.00	1.10	1.10	1.18
0.9	1.00	1.11	1.10	1.19
1.0	1.00	1.11	1.08	1.21
1.1	1.00	1.13	1.11	1.27
1.2	1.00	1.15	1.15	1.27
1.3	1.00	1.16	1.16	1.29
1.4	1.00	1.17	1.17	1.32
1.5	1.00	1.18	1.17	1.33
1.6	1.00	1.21	1.10	1.33
1.7	1.00	1.22	1.09	1.34
1.8	1.00	1.23	1.06	1.33
1.9	1.00	1.26	1.14	1.35
2.0	1.00	1.30	1.04	1.33

Tab. 3.16: *PI* für Blackscholes auf XU4 für verschiedene Frequenzen und Threads.

	RPI Blackscholes			
Freq	1	2	4	8
0.2	1.00	0.71	0.50	0.33
0.3	1.00	0.59	0.51	0.34
0.4	1.00	0.60	0.51	0.34
0.5	1.00	0.60	0.52	0.34
0.6	1.00	0.61	0.52	0.35
0.7	1.00	0.62	0.53	0.36
0.8	1.00	0.62	0.54	0.37
0.9	1.00	0.62	0.54	0.37
1.0	1.00	0.63	0.54	0.39
1.1	1.00	0.64	0.52	0.39
1.2	1.00	0.65	0.57	0.40
1.3	1.00	0.66	0.58	0.41
1.4	1.00	0.66	0.58	0.42
1.5	1.00	0.67	0.61	0.44
1.6	1.00	0.69	0.60	0.46
1.7	1.00	0.69	0.63	0.48
1.8	1.00	0.70	0.64	0.50
1.9	1.00	0.72	0.72	0.53
2.0	1.00	0.75	0.68	0.54

Tab. 3.15: *RPI* für Blackscholes auf XU4 für verschiedene Frequenzen und Threads.

	EPS Blackscholes			
Freq	1	2	4	8
0.2	11561.67	4665.48	2788.56	1175.03
0.3	7844.16	2629.16	1940.31	828.50
0.4	6028.86	2042.55	1484.84	627.25
0.5	4938.38	1680.23	1262.86	525.31
0.6	4240.47	1468.31	1085.29	467.08
0.7	3706.72	1292.05	970.65	415.42
0.8	3363.54	1188.27	885.25	385.86
0.9	3125.81	1100.47	825.38	365.54
1.0	2887.67	1042.74	770.40	353.00
1.1	2749.21	1004.97	669.07	322.09
1.2	2602.31	961.47	728.44	336.13
1.3	2496.77	951.37	719.34	329.37
1.4	2413.06	910.50	698.76	327.92
1.5	2320.11	892.90	750.75	344.11
1.6	2292.03	901.43	758.42	368.54
1.7	2259.22	893.10	822.14	392.32
1.8	2270.24	908.24	874.67	424.81
1.9	2300.43	954.72	1036.87	471.12
2.0	2376.84	1021.98	1065.69	518.81

Tab. 3.17: *EPS* für Blackscholes auf XU4 für verschiedene Frequenzen und Threads.

und wie diese Metriken angewendet werden können.
Die Laufzeit T, Energie E und Leistung P einer Anwendung können für verschiedene Frequenzen und Threads gemessen werden und für einen Vergleich verschiedener Implementierungsvarianten einer Anwendung herangezogen werden. Diese Daten stellen jedoch keine Information hinsichtlich Energie- und Leistungs-Sensitivität dar. Der (Laufzeit-)Speedup S ist die traditionelle Metrik um die Skalierung der Laufzeit paralleler Anwendungen zu parametrisieren. Diese Metrik ignoriert jedoch die Betriebsfrequenz. Der Laufzeitreduktionsfaktor R einer parallelen Anwendung ist eine Modifikation des (Laufzeit-) Speedups, der den Schwerpunkt auf steigende/sinkende Frequenzen setzt, anstatt auf eine steigende/sinkende Anzahl an Threads. Somit eignet sich R gut, um das DVFS-Verhalten von Anwendungen zu untersuchen und den Einfluss der Frequenz auf die Laufzeit darzustellen. Der Energiereduktionsfaktor ER hat denselben Zweck wie R mit dem Unterschied, dass ER den Einfluss der Frequenz auf den Energieverbrauch widerspiegelt.
Die Energie-Sensitivität oder Energieeffizienz kann mit den Metriken ES, EDP und EPS zugänglich gemacht werden. Der Energy Speedup ES kann als energieorientierte Version des (Laufzeit-)Speedup gesehen werden, da dieser den Energieverbrauch von Anwendungen für eine feste Betriebsfrequenzen für parallel ausgeführte Anwendungen darstellt. Ein Wert $ES(p, s) < 1$ bedeutet einen höheren Energieverbrauch durch parallele Ausführung anstatt sequenzieller Ausführung, wohingegen $ES(p, s) > 1$ bedeutet, dass mehrere Threads zu einer Reduzierung der Energie führen. Die Tabellen 3.10 bis 3.13 zeigen, dass der Energieverbrauch bei nahezu allen PARSEC und SPLASH-2 Benchmarks von mehreren Threads profitiert, da ES größer als 1 ist. Soll die Energie einer Anwendung gering bleiben, so wählt man die Implementierungsvariante mit einem höheren $ES(p, s)$.
Die Energy per Speedup EPS ist stark verwandt mit dem EDP: Die Normalisierung mit der sequenziellen Laufzeit erlaubt einen Vergleich verschiedener Anwendungen oder Implementierungen der gleichen Anwendung. Dies geht mit dem EDP nicht. Kleinere EPS Werte deuten auf eine energieeffizientere Ausnutzung paralleler Ausführungen hin.
Das Verhalten der Leistungsaufnahme bezüglich variabler Frequenzen und Threads steht im Fokus der Metriken PS, PI und RPI. Der Power Speedup PS untersucht die Entwicklung der Leistungsaufnahme bei variabler Anzahl der Threads. Typischerweise nimmt diese mit steigender Zahl zu. Die Metrik kann verwendet werden, um die Leistungs-Sensitivität hinsichtlich der parallelen Threads abzubilden. Der Leistungszuwachs PI ist die Inverse des PS. Setzt man PS in Relation zum (Laufzeit-)Speedup, so erhält man den relativen Leistungszuwachs RPI. Dieser zeigt, inwiefern eine Steigerung

der Leistungsaufnahme einer Anwendung zu einer Steigerung der Laufzeit führt. Ein kleiner RPI bedeutet eine gute und effiziente Leistungsaufnahme. Somit sind solche Implementierungsvarianten mit kleinerem RPI zu bevorzugen.

4 | Modellbasierte Optimierung des Energieverbrauchs paralleler Anwendungen

Für moderne Hardware gilt die Energieeffizienz als kritisches Anliegen. Es wurde eine Vielzahl an Hardware-Features entwickelt, um die Energiebilanz zu verbessern [54, 28], unter anderem DVFS. Jedoch wirken sich verschiedene Frequenzen unterschiedlich auf Laufzeit und Energieverbrauch aus.
Wurden in Kapitel 3 Metriken verwendet, um Performance und Energieeffekte einzufangen, beschäftigt sich dieses Kapitel damit, mit analytischen Modellen die Energieeffizienz abzubilden. Zudem wird analysiert, ob ein analytisches Modell in der Lage ist, eine Vorauswahl für eine CPU-Frequenz zu treffen, welche zu einem nahezu optimalen Energieverbrauch einer Anwendung führt. Auch das Energy-Delay-Produkt EDP, welches in Kapitel 3.2.3 schon eingeführt wurde, wird in diesem Zusammenhang wieder verwendet, um daraus ein weiteres Energiemodell zu generieren. Das EDP gewichtet die Laufzeit stärker gegenüber der Leistungsaufnahme. Die Evaluation erfolgt am Ende mit den beiden Benchmarksuites PARSEC und SPLASH-2, welche in Kapitel 2.4 eingeführt wurden. Als Vorlage für dieses Kapitel dienten, wenn nicht anders angegeben, die eingereichten Publikationen [1, 2].

4.1 Einführung

Es ist schwer, vorab einzuschätzen, ob eine kleinere Frequenz auch zu einem geringeren Energieverbrauch führt. Eine geringe Frequenz bedeutet eine längere Laufzeit und eine hohe Frequenz bedeutet eine höhere Leistungsaufnahme. Das Optimum liegt irgendwo dazwischen, selten an einem der beiden Extrempunkte. Bei multi-threaded Anwendungen ist es zudem nicht leicht vorherzusagen, ob der Einsatz von zusätzlichen Threads zu einer Verbesserung der Laufzeit und/oder zu einem geringeren Energieverbrauch führt. Zudem gibt es bei multi-threaded Anwendungen meistens einen Sättigungspunkt, an dem die Laufzeit nicht mehr durch den Einsatz von einer höheren Threadzahl verbessert werden kann. Die zusätzlichen Threads würden nur unnötig die Leistungsaufnahme und

damit den Energieverbrauch erhöhen. Ähnlich verhält es sich mit der Frequenz. Auch hier gibt es meistens ein Optimum, bei dem der Energievierbauch am geringsten ist.

Um den Gebrauch von DVFS sinnvoll einsetzen zu können, ist es wichtig, die Wechselwirkungen zwischen der Frequenz, den Charakteristika der Anwendung und dem resultieren Energieverbrauch und der Performance zu verstehen. In der idealen Umgebung hätte man ein analytisches Modell, das exakt diese Effekte beeinflussende Parameter einfängt. Trotzdem existiert ein solches Modell nicht, da es sehr schwierig ist, alle Einflussfaktoren und deren quantitative Effekte zu parametrisieren, da das Verhalten zu komplex und schwer analytisch zu beschreiben ist.
In diesem Kapitel werden zwei Modelle entwickelt, die zumindest ansatzweise in diese Richtung gehen. Genauer wird von einem Leistungsmodell ausgegangen, welches die Leistungsaufnahme von CMOS-Chips modelliert. Von diesem Modell aus werden zwei weitere Modelle abgeleitet, eins auf Basis der Energie und eins auf Basis des EDP. Beide Modelle werden dahingehend untersucht, ob sie geeignet sind, eine Vorauswahl an Frequenzen zu treffen, welche sich nahe am Energie- oder EDP-Optimum befinden.

Die Evaluierung erfolgt auf der Basis der PARSEC und SPLASH-2 Benchmarks. Die parallelen Workloads und unterschiedlichen Charakteristika eignen sich gut für Untersuchungen hinsichtlich der Energieverläufe. Als Hardware werden wieder das Haswell System (Anhang A.2) und das Skylake System (Anhang A.1) verwendet. Der Rest des Kapitels ist wie folgt strukturiert: In Kapitel 4.2 wird ein neues Energiemodell hergeleitet. In Kapitel 4.3 wird dieses Modell für parallele Programme erweitert und gezeigt, wie man daraus die optimale Frequenz berechnen kann. Im nächsten Kapitel 4.4 wird das Energiemodell so verändert, dass daraus ein Modell für das EDP entsteht. Kapitel 4.5 geht auf die Beobachtungen bei den Messungen ein. Das nächste Kapitel 4.6 stellt die real gemessenen Optima mit den modellierten Optima aus dem Modell gegenüber. Das wird sowohl für das Energie- als auch für das EDP-Modell gemacht. Dort wird auch die Unterscheidung zwischen anwendungsspezifischen und anwendungsunabhängigen Optima eingeführt. Kapitel 4.7 gibt einen Überblick über verwandte Arbeiten bevor das Schlusskapitel 4.8 die Ergebnisse zusammenfasst.

4.2 Leistungs- und Energiemodell für Frequenzskalierung

Wie schon bei den verschiedenen Metriken aus Kapitel 3.1.1, ist es nützlich, auch bei den Energiemodellen Skalierungsfaktoren zu verwenden (eingeführt in Kapitel 3.1.2). Leistungs-Modelle (gemeint ist hier die Leistungsaufnahme) für DVFS-Prozessoren unterscheiden zwischen statischer Leistung und dynamischer Leistung [64]. Die dynamische

Leistung $P_{dyn}(f)$ wird der Versorgungsspannung und der Schaltaktivität während der Berechnungen des Prozessors zugeteilt und kann als

$$P_{dyn}(f) = \alpha \cdot C_L \cdot V^2 \cdot f$$

ausgedrückt werden, wobei α die Schaltwahrscheinlichkeit der Transistoren, C_L die Kapazität und V die Versorgungsspannung ist. Unter der Annahme, dass die Frequenz f linear von der Versorgungsspannung abhängt, gilt $V = \beta \cdot f$ mit einem konstanten β. Somit kann die Abhängigkeit der dynamischen Leistungsaufnahme auf die Frequenz f als $P_{dyn}(f) = \gamma \cdot f^3$ mit $\gamma = \alpha \cdot C_L \cdot \beta^2$ ausgedrückt werden. Bei Substitution der Frequenz mit entsprechendem Skalierungsfaktor s folgt

$$P_{dyn}(s) = s^{-3} \cdot P_{dyn}(1) \tag{4.1}$$

mit $P_{dyn}(1)$ als Leistungsaufnahme im nicht skalierten Fall, also der höchsten Frequenz. Die statische Leistung $P_{stat}(f)$ bildet die Leckleistung ab und kann als $P_{stat}(f) = V \cdot N \cdot k_{\text{design}} \cdot I_{\text{leak}}$, mit N als Anzahl der Transistoren, k_{design} als Design-abhängiger Parameter und I_{leak} als Technologie-abhängiger Parameter (Stromstärke), abgebildet werden [21]. Mit $V = \beta \cdot f$ erhält man wieder eine lineare Abhängigkeit der statischen Leistung und der Frequenz f: $P_{stat}(f) = \delta \cdot f$ mit $\delta = N \cdot k_{\text{design}} \cdot I_{\text{leak}} \cdot \beta$ oder $P_{stat}(s) = s \cdot P_{stat}(1)$ mit $P_{stat}(1)$ als statische Leistung für den nicht skalierten Fall, also wieder der maximalen Frequenz. Andere Publikationen definieren zur Vereinfachung P_{static} als unabhängig von der Spannung oder Frequenz [64], was zu

$$P_{stat}(s) = P_{stat}(1) = P_{static}, \tag{4.2}$$

führt. Für die gesamte Leistungsaufnahme werden nun beide Komponenten addiert:

$$P_{total}(s) = s^{-3} P_{dyn}(1) + P_{static}. \tag{4.3}$$

Eine Reduzierung der Betriebsfrequenz eines Prozessors um den Skalierungsfaktor s senkt üblicherweise die Leistungsaufnahme. Allerdings wird jedoch gleichzeitig die Ausführungszeit T einer Anwendung um den gleichen Faktor im Vergleich zum unskalierten Fall erhöht. Somit gilt $T(s) = s \cdot T(1)$. Nimmt man die Abhängigkeit der Leistung und der Ausführungszeit vom Skalierungsfaktor s so wie eben beschrieben an, führt dies

zum ersten in Abhängigkeit von s definierten Energiemodell $E(s)$:

$$\begin{aligned} E(s) &= (P_{dyn}(s) + P_{static}) \cdot T(s) \\ &= (s^{-3} \cdot P_{dyn}(1) + P_{static}) \cdot s \cdot T(1) \\ &= (s^{-2} \cdot P_{dyn}(1) + s \cdot P_{static}) \cdot T(1). \end{aligned} \tag{4.4}$$

Der Energieverbrauch ist somit das Produkt der Ausführungszeit im nicht skalierten Fall und der Leistungsaufnahme, ebenfalls im nicht skalierten Fall. Die Leistungsaufnahme ist wiederum die Summe aus dynamischer und statischer Leistungsaufnahme.

4.3 Entwicklung eines Energiemodells für parallele Programme

Bei Ausführung paralleler Programme wird eine neue Variable p ein, welche die Anzahl an Threads erfasst. Üblicherweise sinkt mit steigender Threadzahl die Laufzeit paralleler Programme typischerweise nicht linear bis zu einem Sättigungspunkt. Dies ist stark abhängig von der Anwendung. Auf der anderen Seite steigt auch die Leistungsaufnahme mit steigender Threadzahl. Nimmt man die Abhängigkeit zwischen Ausführungszeit und Leistungsaufnahme, Frequenzskalierung und die Anzahl der Threads zusammen, so erhält man ein Energiemodell mit zwei Parametern. Gleichung (4.4) wird erweitert und nimmt die sich veränderliche Anzahl an Threads p zum Skalierungsfaktor s hinzu, was zu

$$E(p, s) = (s^{-2} \cdot P_{dyn}(p, 1) + s \cdot P_{stat}(p, 1)) \cdot T_{par}(p, 1) \tag{4.5}$$

führt. $T_{par}(p, 1)$ ist die parallele Ausführungszeit mit p Threads für den Skalierungsfaktor $s = 1$. $P_{dyn}(p, 1)$ und $P_{stat}(p, 1)$ sind die dynamische und statische Leistungsaufnahme bei p Threads für s=1.

Das Energiemodell in Gleichung (4.5) ist eine differenzierbare Funktion in s. Es wird hier die Annahme getroffen, dass s beliebige Werte annehmen kann. In der Praxis kann s jedoch nur diskrete Werte annehmen, was das Modell etwas ungenau macht. Dies ermöglicht es, analytisch den optimalen Wert für s abzuleiten, wenn p konstant ist. Der optimale Skalierungsfaktor $s_{opt}(p)$ welcher $E(p, s)$ minimiert, kann nun bestimmt werden, indem von $E(p, s)$ die Ableitung gebildet wird und die Nullstelle bestimmt wird. Die Ableitung von $E(p, s)$ ist

$$\frac{\partial E(p, s)}{\partial s} = \left(-2s^{-3} P_{dyn}(p, 1) + P_{static}(p, 1)\right) \cdot T_{par}(p, 1) = 0.$$

Die Funktion $E(p, s)$ in Gleichung (4.5) ist konvex, da die zweite Ableitung $\partial^2 E(p, s)/\partial^2 s$

existiert und $\partial^2 E(p,s)/\partial^2 s \geq 0$ ist. Für das resultierende Optimum gilt

$$s_{opt}(p) = \left(\frac{2 \cdot P_{dyn}(p,1)}{P_{static}(p,1)} \right)^{1/3}, \tag{4.6}$$

unter der Annahme, dass sich p und s während der Ausführung der Anwendung nicht ändern. In Gleichung (4.6) erkennt man, dass der Wert von $s_{opt}(p)$ unabhängig von der aktuellen Laufzeit einer Anwendung ist und so scheint $s_{opt}(p)$ auf den ersten Blick konstant zu sein. Jedoch haben unterschiedliche Anwendungen unterschiedliche Werte in $P_{dyn}(p,1)$ und $P_{static}(p,1)$ für ein konstantes p aufgrund der unterschiedlichen Ausnutzung der Hardware auf dem Prozessor. Weiter führt eine unterschiedliche Anzahl an Threads zu unterschiedlichen $s_{opt}(p)$.

4.4 Entwicklung eines Energiemodells für das EDP

In diesem Kapitel wird das in Kapitel 4.3 eingeführte Energiemodell auf das Energy-Delay-Produkt (EDP) ausgeweitet. Das EDP ist definiert als die Energie, die eine Anwendung verbraucht, multipliziert mit der Ausführungszeit [52]. Da die Energie bereits die Laufzeit berücksichtigt, siehe Gleichung (4.5), ist das EDP somit das Produkt der Leistungsaufnahme mit der quadratischen Laufzeit. Beachtet man den Skalierungsfaktor s, kann das EDP als

$$EDP(p,s) = E(p,s) \cdot (T(p,1) \cdot s) \qquad [Watt \cdot s^2] \tag{4.7}$$

$$= (s^{-1} \cdot P_{dyn}(p,1) + s^2 \cdot P_{static}(p)) \cdot T(p,1)^2 \tag{4.8}$$

geschrieben werden. Bildet man die Ableitung

$$\frac{\partial EDP(p,s)}{\partial s} = \left(-s^{-2} P_{dyn}(p,1) + 2s \cdot P_{static}(p,1) \right) \cdot T(p,1)^2$$

und verfolgt den gleichen Ansatz wie für den Energieverbrauch, führt dies zum optimalen Skalierungsfaktor

$$s_{opt}^{EDP}(p) = \left(\frac{P_{dyn}(p,1)}{2 \cdot P_{static}(p,1)} \right)^{1/3} \tag{4.9}$$

für das EDP mit einem gegebenen p. Der optimale Skalierungsfaktor aus Gleichungen (4.6) und (4.9) haben die Eigenschaft, dass $s_{opt}^{EDP}(p) < s_{opt}(p)$ gilt, da auch $(1/2)^{1/3} < 2^{1/3}$ gilt. Das bedeutet, dass zum Minimieren des EDP ein kleinerer Skalierungsfaktor verwendet wird als für die Minimierung des Energieverbrauchs. Abschließend fasst Tabelle

Tab. 4.1: Zusammenfassung der Notation der Kapitel 4.2 bis 4.4.

Notation		
Symbol	Einheit	Beschreibung
E	Joule	Energieverbrauch
T	Sekunden	Ausführungszeit
P	Watt	Leistungsaufnahme
p	Skalar	Anzahl an Threads
f	1/Sekunden	Betriebsfrequenz
f_{max}	1/Sekunden	Maximale Prozessor-spezifische Frequenz
$s \geq 1$	dimensionslos	Skalierungsfaktor $f = f_{max}/s$
$P_{dyn}(f)$	Watt	dynamische Leistung, abhängig von f
$P_{dyn}(s)$	Watt	dynamische Leistung, abhängig von s
$P_{stat}(f)$	Watt	statische Leistung, abhängig von f
P_{static}	Watt	dynamische Leistung nicht skaliert $s = 1$
$T(1)$	Sekunden	nicht sklaierte sequentielle Ausführungszeit, $s = 1$
$T(s)$	Sekunden	skalierte sequentielle Ausführungszeit $T(s) = s \cdot T(1)$
$E(s)$	Joule	Energie abhängig vom Skalierungsfaktor s
$E(p, s)$	Joule	Energie für parallele Ausführung
$s_{opt}(p)$	dimensionslos	optimaler Skalierungsfaktor für parallele Ausführung
$P_{dyn}(p, 1)$	Watt	dynamische Leistung, nicht skalierte parallele Ausführung
$P_{static}(p, 1)$	Watt	statische Leistung, nicht skalierte parallele Ausführung
$T(p, 1)$	Sekunden	parallele Ausführungszeit für nicht skalierten Fall
$EDP(p, s)$	Joule · Sekunden	Energy-Delay Produkt EDP parallel und skaliert
$s_{opt}^{EDP}(p)$	dimensionslos	optimaler Skalierungsfaktor des EDP für parallele Ausführung

4.1 nochmal alle Symbole und Einheiten der Kapitel 4.2 bis 4.4 zu den Energiemodellen zusammen.

4.5 Messungen und Beobachtungen der Leistung, Energie und EDP

Die Leistungsaufnahme und der Energieverbrauch sind von vielen Faktoren wie der Anzahl der Threads und der Hardware abhängig. Die Tabellen und Diagramme im Folgenden zeigen Messungen der Leistungsaufnahme und des Energieverbauchs für einen, respektive acht Threads der SPLASH-2 Benchmarks auf dem Skylake Haswell Prozessor. Auch Benchmarks aus der PARSEC Benchmarksuite werden hier aufgeführt. Die genauen Beobachtungen werden in diesem Abschnitt festgehalten.

Tab. 4.2: Leisungsaufnahme in Watt des PARSEC Benchmarks Freqmine ausgeführt auf einem Intel i7-6700 Skylake Prozessors für verschiedene Frequenzen und Threads.

	Leistungsaufnahme			
Freq	$p = 1$	$p = 2$	$p = 4$	$p = 8$
0.8 GHz	3.73	3.59	4.56	5.57
1.0 GHz	3.69	3.60	5.71	6.10
1.2 GHz	3.70	4.25	5.91	7.34
1.4 GHz	4.04	4.95	7.96	8.69
1.5 GHz	4.33	5.28	8.48	9.38
1.7 GHz	4.83	5.99	10.21	11.06
1.9 GHz	5.43	6.94	11.94	12.96
2.1 GHz	5.91	8.03	13.78	15.05
2.3 GHz	6.65	9.23	13.51	17.42
2.5 GHz	7.49	10.59	18.40	20.01
2.7 GHz	8.42	12.31	17.72	23.20
2.8 GHz	9.03	13.18	19.20	25.18
3.0 GHz	10.42	15.21	22.20	29.22
3.2 GHz	11.74	11.93	30.95	33.64
3.4 GHz	13.00	13.26	28.63	38.49

Tabelle 4.2 zeigt beispielhaft die Leistungsaufnahme des Benchmarks Freqmine aus der PARSEC Benchmarksuite. Für eine feste Anzahl an Threads und steigender Frequenz nimmt die Leistungsaufnahme zu. Andersherum, also bei einer festen Frequenz und steigender Threadzahl, steigt die Leistungsaufnahme auch kontinuierlich bis 4 Threads. Bei dem Sprung auf 8 Threads steigt die Leistungsaufnahme in den meisten Fällen auch, jedoch manchmal nicht mehr ganz so stark. Abbildung 3.3 (rechts) zeigt die Leistungsaufnahme aller SPLASH-2 Anwendungen auf der Haswell und Skylake Architektur für $p = 1$ und $p = 8$. Auch hier zeigt sich ein ähnliches Verhalten: Mit steigenden Frequenzen respektive steigender Threadzahlen nimmt die Leistung zu. Jedoch kann beobachtet werden, dass quantitativ der Zuwachs der Leistung sehr unterschiedlich zwischen den einzelnen Benchmarks ist. Bei der höchsten Frequenz von 3.4 GHz gehen die Werte weit auseinander. Die SPLASH-2 Benchmarks aus Abbildung 3.3 (rechts) zeigen folgende Leistungswerte in Watt auf:

- Skylake (oben links): für $p = 1$: zwischen 10 und 15 Watt
- Haswell (unten links): für $p = 1$: zwischen 16 und 23 Watt
- Skylake (open rechts): für $p = 8$: zwischen 15 und 43 Watt
- Haswell (unten rechts): für $p = 8$: zwischen 22 und 60 Watt

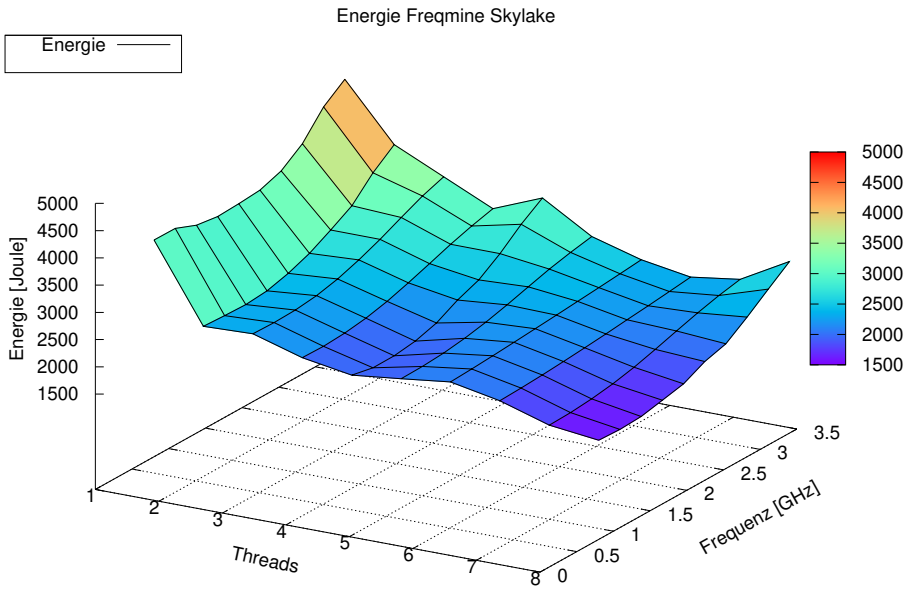

Abb. 4.1: Energieverbrauch des Benchmarks Freqmine aus der PARSEC Benchmarksuite über verschiedene Frequenzen und Threads.

Der Skylake Prozessor ist sowohl bei dem maximalen als auch bei den minimalen Werten deutlich weniger stromsparender als der Haswell Rechner, obwohl beide die gleiche TDP aufweisen. Das zeigt die deutlich bessere Energieeffizienz beim Skylake Prozessor, da bei mehr Leistung die gleiche Abwärme abfällt (vgl. TDPs).

Geht man zum Energieverbrauch über, ist der Vergleich zwischen Haswell und Skylake noch deutlicher. Abbildung 3.4 (rechts) zeigt den Energieverbrauch der SPLASH2-Benchmarks. Für jede Anwendung ist eine Betriebsfrequenz erkennbar, bei der der Energieverbrauch minimal ist. Die typische glattgebügelte U-Form des Energieverbrauchs ist vor allem beim Haswell deutlich ausgeprägter. Die Verläufe bestätigen das Energiemodell aus Abschnitt 4.3, welches eine solchen energieminimalen Punkt in Gleichung (4.6) vorhersagt. Abbildung 4.1 zeigt eine komplette Messung für den PARSEC Benchmark Freqmine auf dem Skylake Prozessor. Die Threads und Frequenzen sind jeweils auf der x-Achse beziehungsweise y-Achse aufgetragen. Die meisten Anwendungen haben einen ähnlichen Energie-Verlauf.

Da das EDP qualitativ den Effekt der Leistung und Energie kombiniert, werden bei steigender Frequenz typischerweise kleinere EDP-Werte erwartet, da die Laufzeit quadratisch einbezogen wird, siehe dazu auch Kapitel 3.2.3. Tabelle 4.3 zeigt den EDP-Verlauf

Tab. 4.3: EDP für den PARSEC Freqmine Benchmark auf einem i7 Skylake.

	EDP			
Freq	$p = 1$	$p = 2$	$p = 4$	$p = 8$
0.8 GHz	3723095.16	1373633.25	824922.42	441310.04
1.0 GHz	3777602.27	1371858.56	557796.35	390715.00
1.2 GHz	3757455.40	1128545.10	604685.25	331431.44
1.4 GHz	3144239.67	966765.90	417609.50	284399.55
1.5 GHz	2918132.72	897168.38	400445.90	268099.84
1.7 GHz	2548585.34	799809.38	350197.07	247316.78
1.9 GHz	2295347.88	740150.73	329960.00	231725.31
2.1 GHz	2028699.78	698448.61	311274.87	221811.76
2.3 GHz	1909394.77	669977.56	377899.42	214698.15
2.5 GHz	1821256.42	653056.97	**293312.49**	**207365.11**
2.7 GHz	1755470.91	649779.96	360036.75	208173.11
2.8 GHz	1751893.85	**647770.16**	364071.61	212228.70
3.0 GHz	1764447.10	650671.30	369381.94	211446.83
3.2 GHz	1751533.01	1143415.82	300938.57	213749.20
3.4 GHz	**1717953.01**	1115712.35	372650.08	218278.50

des Freqmine Benchmarks auf dem Skylake Prozessor. Die Minima (bezogen auf den EDP-Wert) sind für die jeweiligen Frequenzen rot markiert. Es kann beobachtet werden, dass die Frequenz für die Minima der jeweiligen Threads mit steigender Threadzahl sinkt. Die anderen Benchmarks zeigen ähnliche Verläufe.

Im nächsten Abschnitt werden diese Beobachtungen analysiert und mit den eingeführten Energiemodellen verglichen.

4.6 Energie- und Performanceanalyse

In diesem Kapitel wird untersucht, ob die optimalen Frequenzen bezüglich des geringsten Energieverbrauchs denen des analytischen Models aus Kapitel 4.3 in Gleichung (4.6) naha kommen. Ebenso wird das gemessene EDP-Optimum mit dem aus dem analytischen EDP-Modell abgeleiteten (Gleichung (4.9)) Optimum verglichen.. Hierfür wird das Kapitel in mehrere Unterkapitel aufgeteilt. Es gibt zwei Hauptgruppen: Die anwendungsspezifische Modellierung und die anwendungsunabhängige Modellierung. Bei der anwendungsspezifischen Modellierung wird jeder Benchmark für sich betrachtet und einzeln modelliert. Andere Anwendungen haben keinen Einfluss auf die Modellierung. Für jede Prozessor Architektur erhält man eine Vielzahl von optimalen Skalierungsfaktoren, welche sich aber typischerweise in einem gewissen Frequenzbereich bewegen.
Anders bei der anwendungsunabhängigen Modellierung: Dort wird nur eine einzige

optimale Frequenz global für die zugrundeliegende Hardware gesucht. Hierfür wird ein Test-Set an Benchmarks verwendet und dann ein Mittelwert gebildet. Beide Modellierungsansätze können sowohl bei der EDP-Modellelierung als auch bei der Energiemodellierung angewendet werden. In diesem Kapitel werden diese vier Modellierungen am Beispiel der PARSEC und SPLASH-2 Benchmarks gezeigt. Die Modellierung der Leistungsaufnahme nach Gleichung (4.3) kann nur anwendungsspezifisch modelliert werden, da man hier keinen einzelnen Faktor modelliert, sondern die Verläufe der einzelnen Benchmarks.

4.6.1 Bestimmung der anwendungsspezifischen Leistungs-Parameter

Die anwendungsspezifische Leistungs-Modellierung liefert anwendungsspezifische Werte für die Parameter $P_{dyn}(1)$ und P_{static} aus Gleichung (4.3). Für die Werte wurde die Methode der kleinsten Fehlerquadrate auf den gemessenen Werten aus Kapitel 4.5 angewendet. Für die SPLASH-2 Benchmarks sieht man die Ergebnisse in den Tabellen 4.4 und 4.5 für $p = 1$ und $p = 8$ und $f = 0.8$ und $f = 3.4$. Die Daten zeigen folgende Charakteristika: Die Werte für P_{static} sind über alle Benchmarks für $p = 1$ recht ähnlich. Auf dem Skylake Prozessor sind sie kleiner als auf dem Haswell Prozessor. Andererseits weisen die anwendungsspezifischen Werte für $P_{dyn}(f)$ eine breitere Variation auf, was auf die unterschiedliche Ausnutzung der Hardware durch die Benchmarks hindeutet. Ein Vergleich von $P_{dyn}(f)$ auf der Haswell CPU und auf der Skylake CPU zeigt jedoch wieder ähnliche Verläufe auf. Die unterschiedlich hohe Leistungsaufnahme ist also hauptsächlich auf $P_{dyn}(f)$ zurückzuführen. Bei acht Threads gibt es eine geringfügig größere Variation der statischen Leistungsaufnahme über alle Benchmarks. Die größere Variation ist jedoch wieder bei $P_{dyn}(f)$ zu finden, wobei diese bei höheren Frequenzen auch größer ist, als bei kleineren Frequenzen. Wie in Abbildung 3.3 zu beobachtet ist, gibt es eine starke Abhängigkeit zwischen $P_{dyn}(f)$ und der eingestellten Frequenz. Eine Abhängigkeit zwischen der Performance-Skalierung der einzelnen Benchmarks konnte auch beobachtet werden. Genauer gesagt zeigen Benchmarks mit einer guten Performance-Skalierung eine größere dynamische Leistungsaufnahme als Benchmarks mit schwächerer Skalierung. Trotzdem gibt es auch signifikante Variationen zwischen Benchmarks mit guter Skalierbarkeit. Vergleicht man in den Tabellen 4.4 und 4.5 die gesamte Leistungsaufnahme, so sind die modellierten Werte sehr nah an den gemessenen. Die gesamte Leistungsaufnahme kann man aus den Tabellen mithilfe von Gleichung 4.3 errechnen. Die linke grüne Spalte vergleicht man mit der Summe der beiden anderen grünen Spalten für $p = 1$. Für $p = 8$ vergleicht man die zweite Spalte (orange) mit der Summe der anderen beiden orangen Spalten. Addiert man die statische und dynamische

Tab. 4.4: Gemessene und modellierte Leistungsaufnahme nach Gleichung (4.3) der SPLASH-2 Benchmarks auf einem Intel i7-4700 Haswell Prozessor.

		Leistung		***P_{dyn}***		***P_{stat}***	
Benchmark	**Freq**	**p=1**	**p=8**	**p=1**	**p=8**	**p=1**	**p=8**
barnes	0.8	9.18	15.40	0.13	0.40	10.66	20.30
cholesky	0.8	9.21	10.53	0.15	0.17	10.69	12.28
fmm	0.8	8.54	14.09	0.13	0.26	10.49	17.96
lu_cb	0.8	8.40	15.38	0.15	0.51	9.99	19.81
lu_ncb	0.8	9.14	15.62	0.14	0.50	10.42	20.38
ocean_cp	0.8	9.89	15.84	0.15	0.37	11.52	19.19
ocean_ncp	0.8	9.43	15.62	0.15	0.35	10.94	19.33
radiosity	0.8	8.82	9.80	0.14	0.20	10.31	11.82
radix	0.8	8.38	13.33	0.10	0.30	9.87	16.69
raytrace	0.8	9.16	16.33	0.16	0.56	10.93	22.77
volrend	0.8	8.75	14.57	0.13	0.45	10.41	19.73
water_nsquared	0.8	9.29	16.07	0.16	0.54	10.96	20.26
water_spatial	0.8	9.09	14.55	0.14	0.44	10.55	19.21
barnes	3.4	19.66	47.17	10.08	30.82	10.66	20.30
cholesky	3.4	20.59	23.86	11.19	12.70	10.69	12.28
fmm	3.4	19.61	36.36	9.99	19.74	10.49	17.96
lu_cb	3.4	20.69	53.79	11.74	39.09	9.99	19.81
lu_ncb	3.4	20.39	53.52	11.13	38.17	10.42	20.38
ocean_cp	3.4	21.79	44.74	11.31	28.71	11.52	19.19
ocean_ncp	3.4	21.02	42.80	11.17	26.64	10.94	19.33
radiosity	3.4	19.59	25.05	10.42	15.01	10.31	11.82
radix	3.4	16.69	36.86	7.73	22.69	9.87	16.69
raytrace	3.4	22.12	60.09	12.26	42.70	10.93	22.77
volrend	3.4	19.69	49.72	10.35	34.82	10.41	19.73
water_nsquared	3.4	22.03	57.88	12.53	41.33	10.96	20.26
water_spatial	3.4	19.92	48.66	10.42	33.56	10.55	19.21
		gemessen	**gemessen**	**modelliert**	**modelliert**	**modelliert**	**modelliert**

Leistungsaufnahme der entsprechenden Threads und vergleicht sie mit dem gemessenen Wert, so kommt man in Tabelle 4.4 für den Haswell Prozessor auf eine durchschnittliche Abweichung von 1.4 Watt für einen Thread und 4.9 Watt für acht Threads. Beim Skylake Prozessor sind die Abweichungen nahezu vernachlässigbar. Berechnet man die durchschnittliche Abweichung in Tabelle 4.5, ergibt sich für einen Thread eine durchschnittliche Abweichung von nur 0.18 Watt und nur 0.84 Watt für acht Threads.

In Abbildung 4.2 wird der Verlauf der Leistung des Barnes Benchmark aus der SPLASH-2 Suite in seine Bestandteile P_{static} und $P_{dyn}(f)$ für jeweils $p = 1$ und p=8 aufgeteilt. Auf der x-Achse sind die verschiedenen Frequenzen aufgetragen. Addiert man die modellierten Werte für P_{static} und $P_{dyn}(f)$ für p=1, welche blau abgebildet sind, so erhält man die gemessene Leistung, hier violett abgebildet. Dasselbe kann man für $p = 8$ beobachten. Gut erkennbar ist, dass P_{static} die dynamische Leistung $P_{dyn}(f)$ nur horizontal nach oben verschiebt. Bei einer hohen Threadzahl ist diese Verschiebung

Tab. 4.5: Gemessene und modellierte Leistungsaufnahme nach Gleichung (4.3) der SPLASH-2 Benchmarks auf einem Intel i7-6700 Skylake Prozessor.

		Leistung		***P_{dyn}***		***P_{stat}***	
Benchmark	**Freq**	**p=1**	**p=8**	**p=1**	**p=8**	**p=1**	**p=8**
barnes	0.8	3.70	5.63	0.13	0.40	3.61	6.43
cholesky	0.8	2.98	3.40	0.14	0.16	3.28	3.67
fmm	0.8	3.41	5.24	0.12	0.36	3.47	6.05
lu_cb	0.8	3.74	5.49	0.13	0.40	3.58	6.24
lu_ncb	0.8	3.58	5.69	0.13	0.41	3.49	6.69
ocean_cp	0.8	4.25	6.32	0.13	0.27	4.13	7.48
ocean_ncp	0.8	4.07	6.28	0.14	0.27	4.10	7.46
radiosity	0.8	3.53	3.88	0.12	0.17	3.36	3.76
radix	0.8	3.19	4.20	0.09	0.24	3.07	4.79
raytrace	0.8	4.01	6.21	0.14	0.47	3.79	7.19
volrend	0.8	3.66	5.47	0.12	0.42	3.49	6.22
water_nsquared	0.8	3.94	6.19	0.14	0.46	3.79	7.19
water_spatial	0.8	3.65	5.66	0.13	0.41	3.54	6.27
barnes	3.4	12.99	36.19	9.72	30.40	3.61	6.43
cholesky	3.4	13.73	15.96	10.68	12.48	3.28	3.67
fmm	3.4	12.80	33.09	9.58	27.66	3.47	6.05
lu_cb	3.4	12.96	36.58	9.64	30.92	3.58	6.24
lu_ncb	3.4	13.02	37.51	9.83	31.52	3.49	6.69
ocean_cp	3.4	14.00	27.30	10.31	20.63	4.13	7.48
ocean_ncp	3.4	14.47	27.56	10.75	20.89	4.10	7.46
radiosity	3.4	12.45	16.30	9.27	12.73	3.36	3.76
radix	3.4	9.69	22.84	6.75	18.57	3.07	4.79
raytrace	3.4	14.40	42.90	10.78	36.36	3.79	7.19
volrend	3.4	12.64	37.66	9.39	31.90	3.49	6.22
water_nsquared	3.4	14.37	41.97	11.03	35.49	3.79	7.19
water_spatial	3.4	12.82	36.74	9.60	31.12	3.54	6.27
		gemessen	**gemessen**	**modelliert**	**modelliert**	**modelliert**	**modelliert**

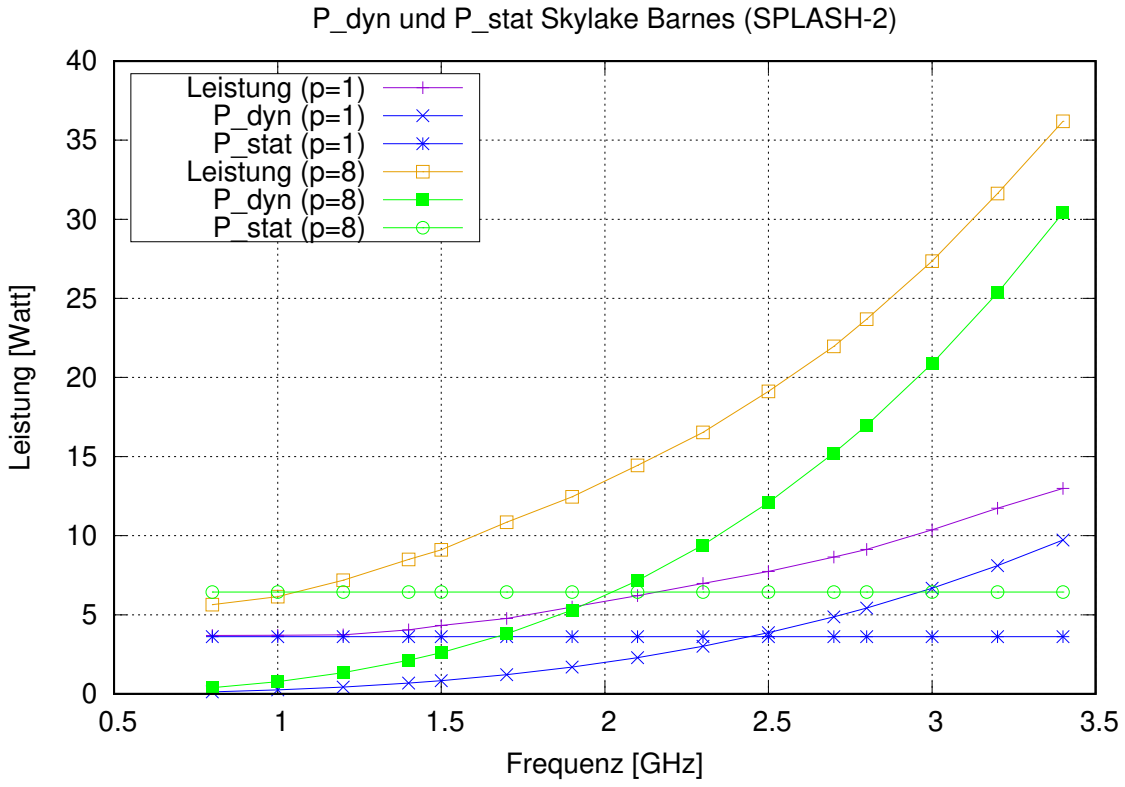

Abb. 4.2: Gemessene und modellierte Leistungsaufname des SPLASH-2 Benchmark Barnes für einen Thread (p=1) und acht Threads (p=8) auf einem Intel i7-6700 Skylake Prozessor.

deutlich größer. Auch die dynamische Leistung verläuft deutlich steiler, als bei p=1. Hier arbeiten mehr Threads mit einer bestimmten Frequenz, was die Spannungsversorgung erhöht und folglich eine höhere Stromaufnahme zur Folge hat.

4.6.2 Anwendungsspezifische Modellierung des optimalen Skalierungsfaktors

Tabellen 4.6 und 4.7 fassen die gemessenen und modellierten Performance- und Energie-Charakteristika der PARSEC Bechmarksuite für die Haswell (Tabelle 4.6) und Skylake (Tabelle 4.7) Architektur zusammen. Die gemessenen Daten in den Spalten 1 und 2 sind die kürzeste und längste Laufzeit in Sekunden für unterschiedliche Frequenzen und eine unterschiedliche Anzahl an Threads. In den Spalten 3 und 4 steht der gemessene Laufzeit-Speedup für f=0.8 GHz und f=3.4 GHz für acht Threads. Bis auf die Benchmarks facesim und ferret wurde bei acht Threads immer der höchste Speedup erreicht. Bei den beiden genannten Benchmarks lag der Speedup um den Wert von 1, was bedeutet, dass diese Benchmarks nicht mit den Threads skalieren. Spalten 5 und 6 zeigen den niedrigsten und höchsten Energieverbrauch über alle Frequenzen hinweg in Joule. In den Spalten 7 und 8 wird die Frequenz gelistet, bei der der jeweilige Benchmark den niedrigsten Energieverbrauch für p=1 und p=8 hatte.
Die modellierten Werte für $s_{opt}(p)$ sind aus Gleichung (4.6) aus Kapitel 4.2 bestimmt und das Ergebnis, also die optimalen Skalierungsfaktoren, in den Spalten 9 und 10 für p=1 and p=8 eingetragen. In Klammern wird die nächstgelegene Frequenz gezeigt, da für die Prozessoren nicht jede beliebige Frequenz manuell eingestellt werden kann. Die Spalten 11 und 12 zeigen die prozentuale Abweichung der Energie der modellierten Skalierungsfaktoren vom jeweils gemessenen Optimum. Bei einer Abweichung von 0 entspricht das abgeleitete Frequenz-Optimum dem gemessenen Optimum. Die letzten beiden Spalten zeigen die Resultate (Abweichungen) für die anwendungsunabhängige Optimierung. Das anwendungsunabhängige Optimum wird später in Kapitel 4.6.4 näher erklärt.
Es gibt eine starke Variation für jede Anwendung was Laufzeiten und Energievierbräuche angeht. Auffällig ist, dass für p=8 Threads eine kleinere Frequenz benötigt wird, um das Energieoptimum zu erreichen, als für p=1, siehe Spalten 7 und 8. Dies führt zu höheren Skalierungsfaktoren in den Spalten 9 und 10.
Eine wichtiges Resultat stellen Spalten 11 und 12 dar, welche die prozentuale Abweichung zwischen gemessenen und vorhergesagten Energie-Werten zeigen. Der Wert 0 indiziert einen perfekten Fit. Bis auf einen Benchmark liegt dieser Wert immer unter 10%. Die höchste Abweichung liegt auf dem Haswell Prozessor bei 12.4% (Streamcluster für

Tab. 4.6: Evaluation verschiedener **PARSEC** Benchmarks auf einem i7-4770 Haswell Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. **E-Diff** und **E-Diff-glob** zeigen prozentuale Unterschiede in der Energie für anwendungsspezifische und anwendungsunabhängige Modelierung.

	Laufzeit		**Speedup**		**Energieverbr.**		**Beste Freq**		s_{opt} (f)		**E-Diff** [%]		**E-Diff-glob** [%]	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
blackscholes	37.12	558.67	3.60	3.54	948.56	2323.36	2.3	1.2	1.83 (1.9)	2.13 (1.5)	2.3	6.2	2.3	3.0
bodytrack	31.52	463.37	3.72	3.47	868.69	2134.53	2.3	1.2	1.80 (1.9)	2.15 (1.5)	3.5	7.2	3.5	3.8
canneal	71.00	527.51	2.59	2.66	1209.09	2851.98	1.5	1.2	1.68 (2.1)	1.86 (1.9)	6.3	6.9	1.6	7.0
facesim	0.27	1.11	1.00	0.93	3.84	6.07	2.3	2.3	1.89 (1.9)	1.78 (1.9)	2.9	5.5	2.9	4.3
ferret	0.26	1.07	1.00	0.94	3.63	5.89	2.3	2.3	1.93 (1.7)	1.74 (1.9)	9.9	7.8	11.3	6.1
fluidanimate	63.30	1054.09	4.32	4.02	1949.83	5077.73	2.3	1.2	1.80 (1.9)	2.20 (1.5)	2.9	7.3	2.9	6.4
freqmine	80.63	1581.32	4.67	4.65	2742.39	7700.87	2.1	1.2	1.81 (1.9)	2.29 (1.5)	2.1	5.1	2.1	6.9
streamcluster	71.25	918.56	5.19	4.99	1564.56	5222.48	1.7	1.2	1.55 (2.1)	1.89 (1.9)	3.0	12.4	1.1	7.8
swaptions	39.46	867.48	5.18	5.18	1411.24	4313.39	2.3	1.2	1.78 (1.9)	2.35 (1.4)	1.7	4.0	1.7	3.8
vips	21.63	341.41	4.30	3.76	648.34	1703.58	2.1	1.2	1.78 (1.9)	2.03 (1.7)	1.8	6.9	1.7	6.8
x264	16.87	396.76	5.95	5.75	560.11	2155.81	2.1	1.2	1.73 (1.9)	2.18 (1.5)	1.0	5.0	0.9	6.5
	gemessen		**gemessen**		**gemessen**		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Tab. 4.7: Evaluation verschiedener **PARSEC** Benchmarks auf einem i7-6700 Skylake Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz. **E-Diff** und **E-Diff-glob** zeigen prozentuale Unterschiede in der Energie für anwendungsspezifische und anwendungsunabhängige Modelierung.

	Laufzeit		**Speedup**		**Energieverbr.**		**Beste Freq**		s_{opt} (f)		**E-Diff** [%]		**E-Diff-glob** [%]	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
blackscholes	34.02	333.86	2.97	3.70	539.53	1546.02	2.1	1.2	1.97 (1.7)	2.53 (1.4)	2.3	1.3	2.3	1.3
bodytrack	27.15	294.10	2.91	3.91	490.99	1314.92	1.7	1.2	1.97 (1.7)	2.85 (1.2)	0	0	0	0.5
canneal	67.94	456.88	2.80	2.77	650.01	2015.27	1.5	0.8	1.92 (1.7)	2.28 (1.5)	2.8	2.8	2.8	1.8
facesim	0.26	0.99	1.01	1.00	2.33	3.47	1.7	1.7	2.17 (1.5)	2.22 (1.5)	2.8	1.0	0	1.8
ferret	0.49	1.72	1.03	0.99	4.53	7.17	2.1	1.5	2.08 (1.7)	2.19 (1.5)	3.3	0.1	3.3	2.3
fluidanimate	60.51	636.68	2.97	3.95	1151.72	3151.76	1.9	1.0	2.02 (1.7)	2.79 (1.2)	0.4	0	0.4	1.1
freqmine	75.31	1011.15	3.55	4.83	1543.42	4726.40	2.1	1.0	2.02 (1.7)	2.97 (1.2)	1.3	1.0	1.3	1.8
streamcluster	70.36	878.20	5.29	4.77	878.56	3851.71	1.5	0.8	1.96 (1.7)	2.31 (1.4)	3.5	4.1	3.5	1.8
swaptions	35.91	516.00	3.68	5.31	779.57	2529.71	1.9	1.0	1.98 (1.7)	2.96 (1.2)	6.3	1.6	6.3	2.0
vips	18.54	221.89	3.28	4.27	367.38	1067.31	2.1	1.0	2.03 (1.7)	2.86 (1.2)	0.2	0.1	0.2	0.4
x264	14.61	255.89	4.92	6.20	304.57	1197.61	1.7	1.0	2.01 (1.7)	2.86 (1.2)	0	0.1	0	1.4
	gemessen		**gemessen**		**gemessen**		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

p=8) und 6.3% auf dem Skylake Prozessor (Swaptions für p=1). Im Schnitt ist die Abweichung bei 3.4% für p=1 und 6.2% für p=8 für die Haswell Architektur. Für die Skylake Architektur liegen die Abweichungen bei 1.9% und 1.0% für p=1 und p=8. Somit ist das Energiemodell etwas besser auf der Skylake Architektur.

Ein Vergleich zwischen Haswell und Skylake Architektur zeigt, dass beim Skylake Prozessor die optimalen Frequenzen typischerweise niedriger sind als für den Haswell Prozessor. Beim Haswell Prozessor sind die Frequenzen für den geringsten Energieverbrauch zwischen 1.2 GHz und 2.3 GHz mit niedrigeren Frequenzen mit zunehmender Threadzahl zu finden. Beim Skylake Prozessor liegen diese Frequenzen zwischen 0.8 GHz und 2.1 GHz - auch hier mit kleineren Frequenzen mit zunehmender Threadzahl. Die Anwendungen profitieren also mehr durch eine gesteigerte Threadzahl als von höheren Frequenzen wenn es um einen geringeren Energieverbrauch geht.

Tab. 4.8: Evaluation der **SPLASH-2** Benchmarks auf einem Intel i7-4700 Haswell Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz.

	Laufzeit		**Speedup**		**Energieverbr.**		**Beste Freq**		s_{opt} (f)		**E-Diff** [%]		**E-Diff-glob** [%]	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
barnes	25.43	442.72	3.97	4.22	1199.80	4063.08	3.2	3.4	1.23 (2.7)	1.44 (2.3)	4.4	4.7	0.2	2.0
cholesky	0.35	1.52	1.11	1.85	7.71	14.47	3.2	3.0	1.14 (3.0)	1.27 (2.7)	8.0	2.4	8.0	5.2
fmm	27.98	317.54	3.61	2.85	969.55	2710.94	3.0	1.9	1.23 (2.7)	1.30 (2.7)	3.1	7.2	0.0	7.9
lu_cb	26.88	408.14	3.61	3.61	1323.18	3430.18	3.0	1.9	1.32 (2.5)	1.58 (2.1)	5.8	0.9	0.0	8.6
lu_ncb	35.20	502.01	3.81	3.69	1588.19	4588.54	3.0	1.9	1.28 (2.7)	1.55 (2.1)	0.9	0.3	0.0	11.2
ocean_cp	50.57	289.78	3.66	1.50	1188.19	2865.26	3.0	1.0	1.25 (2.7)	1.44 (2.3)	2.1	27.8	0.0	40.6
ocean_ncp	54.66	532.34	5.50	2.42	1400.14	5020.83	3.0	1.0	1.26 (2.7)	1.40 (2.5)	2.6	17.9	0.0	31.0
radiosity	108.5	534.57	1.13	1.16	2478.76	4713.34	3.4	3.0	1.26 (2.7)	1.36 (2.5)	4.9	7.0	0.7	7.0
radix	7.82	140.55	6.55	4.27	238.103	1177.16	3.4	2.1	1.16 (3.0)	1.39 (2.5)	2.5	7.1	2.6	7.3
raytrace	22.86	489.27	5.04	5.01	1332.35	4482.98	3.0	2.1	1.30 (2.5)	1.55 (2.1)	5.5	0.0	0.0	2.4
volrend	24.68	479.36	4.53	4.46	1201.91	4192.04	3.4	1.9	1.25 (2.7)	1.52 (2.3)	6.5	5.3	5.2	4.4
water_nsquared	84.67	1253.1	3.78	3.63	4097.42	11647.5	3.0	2.1	1.31 (2.5)	1.59 (2.1)	3.4	0.0	0.0	5.6
water_spatial	24.84	495.01	4.81	4.83	1166.13	4497.56	3.2	1.9	1.25 (2.7)	1.51 (2.3)	3.6	4.5	1.1	4.3
	gemessen		**gemessen**		**gemessen**		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Tab. 4.9: Evaluation der **SPLASH-2** Benchmarks auf einem Intel i7-6700 Skylake Prozessor mit Laufzeit in Sekunden, Energie in Joule und Frequenz in GHz.

	Laufzeit		**Speedup**		**Energieverbr.**		**Beste Freq**		s_{opt} (f)		**E-Diff** [%]		**E-Diff-glob** [%]	
Benchmark	min	max	f=0.8	f=3.4	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
barnes	21.39	232.84	3.05	4.48	417.23	1244.07	1.7	1.2	2.01 (1.7)	2.79 (1.2)	0.0	0.0	0.0	2.2
cholesky	0.32	1.32	1.10	1.13	3.39	5.14	1.7	1.5	2.22 (1.5)	2.26 (1.5)	0.2	0.0	0.0	2.9
fmm	18.64	198.87	3.21	3.88	322.22	926.49	1.7	1.0	2.03 (1.7)	2.73 (1.2)	0.0	0.2	0.0	1.4
lu_cb	28.65	317.63	2.97	4.24	576.70	1573.95	1.5	1.2	2.00 (1.7)	2.89 (1.2)	0.4	0.0	0.4	0.8
lu_ncb	34.78	407.64	3.36	4.18	652.01	1892.70	1.5	1.0	2.05 (1.7)	2.79 (1.2)	1.8	0.8	1.8	2.9
ocean_cp	34.98	125.07	2.36	1.59	335.16	784.12	1.5	0.8	1.94 (1.7)	2.04 (1.7)	0.2	23.2	0.2	11.6
ocean_ncp	39.60	139.62	2.24	1.58	388.21	1103.04	1.5	1.0	1.99 (1.7)	2.06 (1.7)	1.4	20.2	1.4	9.9
radiosity	98.38	317.35	1.13	1.21	1073.33	1740.68	1.7	1.5	2.03 (1.7)	2.25 (1.5)	0.0	0.9	0.0	0.0
radix	6.48	94.27	4.77	5.52	78.92	346.79	1.9	1.5	1.82 (1.9)	2.45 (1.4)	0.0	0.0	1.4	0.0
raytrace	20.88	273.69	3.46	4.98	477.11	1498.99	1.7	1.0	2.06 (1.7)	2.94 (1.2)	0.0	0.9	0.0	2.3
volrend	23.71	298.27	3.33	4.80	480.75	1437.68	1.7	1.2	2.00 (1.7)	2.97 (1.2)	0.0	0.0	0.0	1.1
water_nsquared	72.83	715.78	2.91	3.84	1519.57	4015.32	1.5	1.0	2.09 (1.7)	2.89 (1.2)	0.9	1.2	0.9	3.7
water_spatial	22.35	279.36	3.56	4.80	439.87	1376.12	1.7	1.0	2.01 (1.7)	2.89 (1.2)	0.0	0.8	0.0	0.5
	gemessen		**gemessen**		**gemessen**		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Die Tabellen 4.8 und 4.9 enthalten entsprechende Informationen für die SPLASH-2 Benchmarks für den Haswell Prozessor und den Skylake Prozessor. Die anwendungsspezifische Modellierung in den Spalten 11 und 12 ist meistens recht ähnlich für p=1 und p=8 mit Ausnahme der beiden Ocean Benchmarks für p=8. Diese Unstimmigkeit entsteht durch das irreguläre Verhalten der parallelen Ausführungszeit beider Benchmarks für $p = 8$ mit variablen Frequenzen: Mit der Erhöhung der Frequenz nimmt die Laufzeit viel weniger ab als vom Model erwartet und ist auch generell viel kleiner als es bei allen PARSEC und SPLASH-2 Benchmarks beobachtet werden kann. Andererseits ist der Zuwachs in der Leistungsaufnahme erwartungsgemäß. Diese Anomalien können von den Modellen nicht eingefangen werden. Dementsprechend liegt die Abweichung vom Modell hier sehr groß. Für p=1 kann dieser Effekt nicht beobachtet werden. Dieser Effekt wird durch die anwendungsunabhängige Modellierung abgefangen, wie sich später in Kapitel 4.6.4 zeigt.

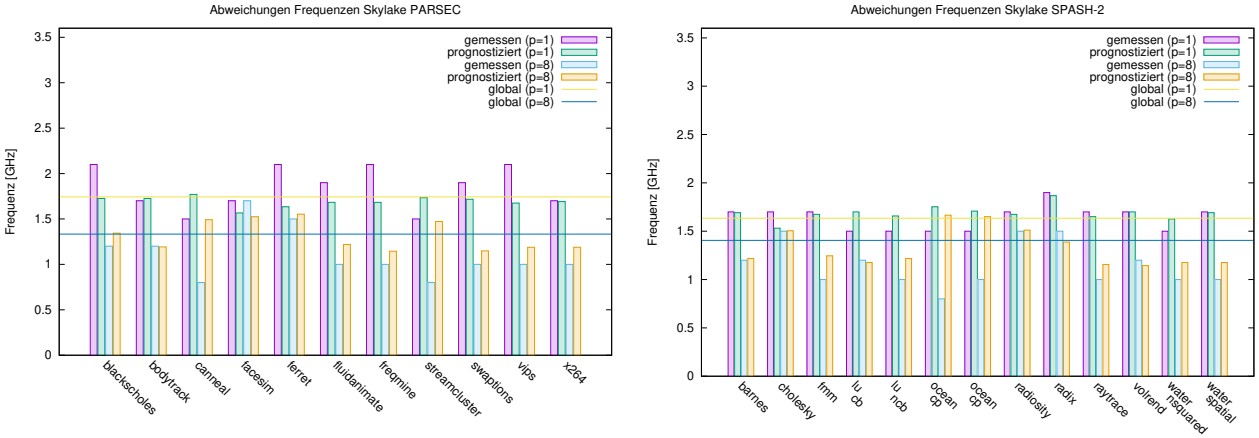

Abb. 4.3: Vergleich der gemessenen und vorhergesagten besten Frequenzen für PARSEC (links) and SPLASH-2 (rechts) Benchmark Suites auf einem i7-6700 Skylake Prozessor.

Abbildung 4.3 zeigt die optimalen Frequenz bezüglich des geringsten Energieverbrauchs für p=1 und p=8. Hier wird die gemessene optimale Frequenz mit der nach Gleichung 4.6 modellierten Frequenz auf dem Skylake System verglichen. Das linke Diagramm zeigt die PARSEC Benchmarks und das rechte Diagramm zeigt die SPLASH-2 Benchmarks. Die anwendungsunabhängigen (im Diagramm als global bezeichnet) modellierten Frequenzen sind als blaue beziehungsweise gelbe Linie aufgetragen. Für einige Benchmarks unterscheiden sich die modellierten von den gemessenen Frequenzen stark. Die Unterschiede zwischen diesen Frequenzen im Energieverbrauch sind jedoch nicht hoch, wie man in den Tabellen 4.6 bis 4.9 sehen kann. Eine erhöhte Abweichung in der Frequenz führt also nicht zwingend zu erhöhter Abweichung im realen Energieverbrauch. Abbildung 4.4 zeigt entsprechende Resultate für die Haswell Architektur.
Vergleicht man die Auswertungen der PARSEC Benchmarks in den Tabellen 4.6 und 4.7 mit den Ergebnisse der SPLASH-2 Benchmarks in den Tabellen 4.8 und 4.9, so treffen die Vohersagen des Energiemodells am besten auf das Skylake System auf den SPLASH-2 Benchmarks zu. Auch bei den PARSEC Benchmarks sind die Vorhersagen beim Skylake System etwas besser. Beim Haswell System sind die Abweichungen bei $E - Diff$ bei beiden Benchmark-Suites sehr ähnlich. Bei irregulärem Verhalten, wie beispielsweise bei den Ocean-Benchmarks, eignet sich das Modell nicht. Das Modell basiert auf einer grundsätzlichen Skalierbarkeit der Anwendungen. Dementsprechend kann es dieses Verhalten nicht einfangen.

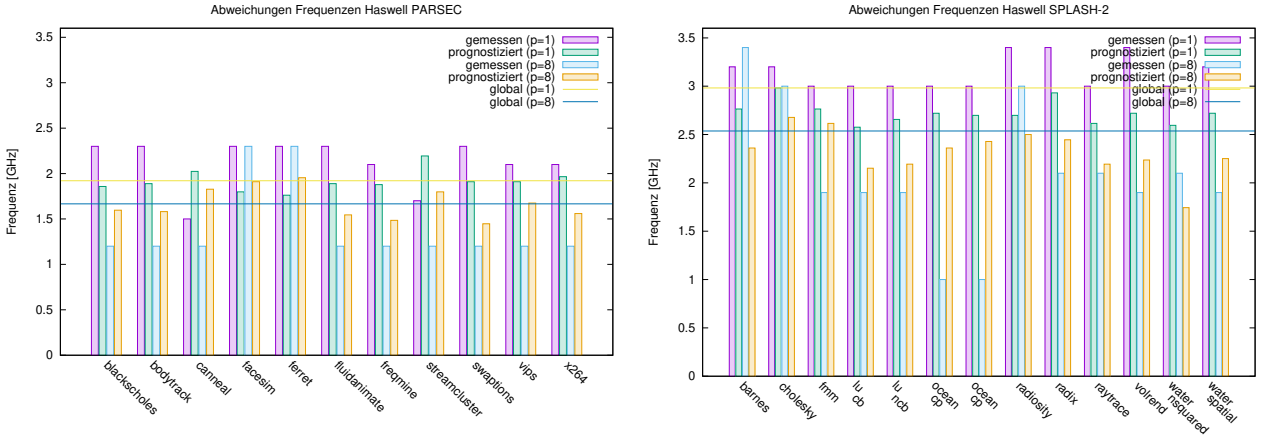

Abb. 4.4: Vergleich der gemessenen und vorhergesagten besten Frequenzen für PARSEC (links) and SPLASH-2 (rechts) Benchmark Suites auf einem i7-4770 Haswell Prozessor.

4.6.3 Anwendungsspezifische optimale EDP Frequenzen

Die Modellierung der optimalen EDP Frequenzen sind in den Tabellen 4.10 und 4.11 für die PARSEC Benchmarks und in den Tabellen 4.12 und 4.13 für die SPLASH-2 Benchmarks auf jeweils dem Haswell Prozessor und dem Skylake Prozessor abgebildet. Beide Tabellen enthalten sowohl das Minimum als auch das Maximum des gemessenen EDP Werts in der Einheit Joule · Sekunden und das jeweils für verschiedene Frequenzen und Threadzahlen, siehe Spalten 1 und 2. Weiter wird die Frequenz für das kleinste gemessene EDP für p=1 und p=8 Threads in den Spalten 3 und 4 gezeigt. Der optimale EDP Skalierungsfaktor aus Gleichung (4.9) befindet sich für einen Thread in Spalte 5 und für acht Threads in Spalte 6. Spalten 7 und 8 zeigen die prozentuale Abweichung des durch das entsprechende EDP-Modell bestimmten EDPs verglichen mit dem durch eine Messung bestimmten Optimums.

In den Tabellen 4.10 und 4.11 stechen wieder die beiden Benchmarks facesim und ferret aufgrund der sehr kurzen Laufzeit und der nicht vorhandenen Skalierung heraus. Kleine EDP Werte benötigen höhere Frequenzen, da die Laufzeit eine größere Rolle spielt, wie man in den Tabellen 4.10 bis 4.13 erkennt. Dies bestätigt das EDP-Modell aus Abschnitt 4.4 und die optimalen Skalierungsfaktoren, die in den Spalten 5 und 6 gezeigt werden. Die prozentuale Abweichung zwischen den minimalen gemessenen EDP Werten und den EDP Werten, die aus dem Modell abgeleitet wurden, sind etwas höher als beim Energiemodell: Für die PARSEC Benchmarks lagen die anwendungsspezifischen Abweichungen zwischen 6.8 % und 8.5 % auf dem Haswell Prozessor und zwischen 3.8 % und 9.3 % auf dem Skylake Prozessor für p=1 and p=8 (Spalten 7 und 8). Für p=8

Tab. 4.10: Evaluation und Modellierung des EDP für **PARSEC** Benchmarks auf einem Intel i7-4770 Haswell Prozessor.

	EDP		**Beste Freq**		s_{opt}^{EDP} (f)		**EDP-Diff** [%]		**EDP-Diff-glob** [%]	
Benchmark	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
blackscholes	47745.13	1297994.89	3.4	3.2	1.15 (3.0)	1.34 (2.5)	5.4	11.9	5.7	7.3
bodytrack	37647.80	989076.15	3.4	3.2	1.13 (3.0)	1.35 (2.5)	4.1	16.2	4.2	14.2
canneal	120885.86	1354092.07	2.8	2.7	1.06 (3.2)	1.17 (3.0)	14.1	5.9	4.3	0
facesim	1.20	6.69	3.4	3.2	1.19 (2.8)	1.12 (3.0)	13.6	4.8	7.6	9.1
ferret	1.13	6.35	3.4	3.4	1.21 (2.8)	1.10 (3.0)	0.8	8.4	2.3	16.1
fluidanimate	535238.18	186174.95	3.4	3.2	1.13 (3.0)	1.38 (2.5)	2.4	6.2	2.5	2.8
freqmine	326402.63	121775.17	3.4	3.2	1.14 (3.0)	1.44 (2.3)	3.1	9.7	3.1	6.6
streamcluster	157654.12	4797148.67	2.3	1.9	0.97 (3.4)	1.19 (2.8)	18.5	12.1	10.4	10.4
swaptions	81911.89	3741768.23	3.4	3.4	1.12 (3.0)	1.48 (2.3)	4.8	10.4	5.0	7.77
vips	19470.45	581622.91	3.4	3.0	1.12 (3.0)	1.28 (2.7)	4.1	4.9	4.1	5.1
x264	13792.44	855348.67	3.4	3.4	1.09 (3.2)	1.37 (2.5)	7.6	11.1	5.3	7.4
	gemessen		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Tab. 4.11: Evaluation und Modellierung des EDP für **PARSEC** Benchmarks auf Intel i7-6700 Skylake Prozessor.

	EDP		**Best Freq**		s_{opt}^{EDP} (f)		**EDP-Diff** [%]		**EDP-Diff-glob** [%]	
Benchmark	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
blackscholes	29693.18	408471.95	3.4	2.7	1.24 (2.7)	1.59 (2.1)	3.0	9.3	1.9	10.3
bodytrack	22756.40	314972.63	3.4	2.7	1.24 (2.7)	1.79 (1.9)	3.5	12.0	2.6	7.7
canneal	68628.75	569316.84	2.7	2.3	1.21 (2.8)	1.43 (2.3)	2.9	0	2.9	1.0
facesim	0.86	2.99	2.8	3.0	1.36 (2.5)	1.40 (2.5)	7.5	6.1	0	21.2
ferret	2.80	9.11	2.8	3.2	1.31 (2.5)	1.38 (2.5)	10.9	3.8	0	21.5
fluidanimate	120781.00	1529633.80	3.4	2.5	1.27 (2.7)	1.76 (1.9)	0.3	8.5	1.4	4.7
freqmine	207365.11	3777602.27	3.4	2.5	1.27 (2.7)	1.87 (1.9)	2.1	10.5	1.9	6.9
streamcluster	94014.90	2456987.16	2.3	1.7	1.23 (2.7)	1.45 (2.3)	0.8	3.7	3.3	1.8
swaptions	49561.53	1031320.00	3.4	2.7	1.25 (2.7)	1.86 (1.9)	2.9	11.8	4.3	7.0
vips	11839.69	183456.93	3.2	2.7	1.28 (2.7)	1.80 (1.9)	0.4	11.2	1.3	7.0
x264	7760.78	244215.62	3.4	2.5	1.26 (2.7)	1.80 (1.9)	6.5	8.7	1.8	5.0
	gemessen		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

auf dem Skylake System zeigen sich die größten Abweichungen zwischen den beiden Modellen. Der höhere Einfluss der Laufzeit zeigt hier seine Wirkung. Beide Modelle nehmen an, dass eine Verdopplung der Frequenz die Laufzeit halbiert, was in der Realität nicht ganz zutrifft. Beim EDP erzeugt diese Ungenauigkeit die höheren Abweichungen als das Energiemodell. Bei den SPLASH-2 Benchmarks liefert das EDP-Modell sehr gute Resultate. Die Abweichungen für p=1 sind bei der Haswell Architektur nicht vorhanden. Beim Skylake Prozessor ist die Abweichung mit Ausnahme beider Ocean Benchmarks immer unter 2 %. Lässt man bei p=8 die beiden Ocean Benchmarks, die sich bereits beim Modellieren der Energie als problematischen erwiesen haben, weg, so ist beim Haswell Prozessor bis auf den Cholesky Benchmark (Abweichung 2.39 %) und dem Radixsort (Abweichung 4.4 %) auch keine Abweichung zu sehen. Beim Skylake Prozessor zeigt sich ein ähnliches Bild. Die Abweichungen sind bis auf die beiden Ocean-Benchmarks immer deutlich unter 2 %.

Tab. 4.12: Evaluation und Modellierung des EDP für **SPLASH-2** Benchmarks auf einem Intel i7-4770 Haswell Prozessor.

	EDP		**Beste Freq**		s_{opt}^{EDP} (f)		**EDP-Diff** [%]		**EDP-Diff-glob** [%]	
Benchmark	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
barnes	30518.1	1798840	3.4	3.4	0.78 (3.4)	0.91 (3.4)	0.0	0.0	0.0	0.0
cholesky	2.70503	21.4811	3.4	3.2	0.71 (3.4)	0.80 (3.4)	0.0	2.39	0.0	2.39
fmm	28483.4	860843	3.4	3.4	0.78 (3.4)	0.81 (3.4)	0.0	0.0	0.0	0.0
lu_cb	38890.3	1400000	3.4	3.4	0.83 (3.4)	0.99 (3.4)	0.0	0.0	0.0	0.0
lu_ncb	66348.6	2303530	3.4	3.4	0.81 (3.4)	0.97 (3.4)	0.0	0.0	0.0	0.0
ocean_cp	74499.6	830306	3.4	1.5	0.78 (3.4)	0.90 (3.4)	0.0	53.58	0.0	53.58
ocean_ncp	92290.7	2672790	3.4	1.9	0.80 (3.4)	0.80 (3.4)	0.0	38.58	0.0	38.58
radiosity	295224	2519620	3.4	3.4	0.79 (3.4)	0.85 (3.4)	0.0	0.0	0.0	0.0
radix	2158.91	165450	3.4	3.0	0.73 (3.4)	0.87 (3.4)	0.0	4.4	0.0	4.4
raytrace	31409.5	2193410	3.4	3.4	0.82 (3.4)	0.97 (3.4)	0.0	0.0	0.0	0.0
volrend	30308.2	2009500	3.4	3.4	0.79 (3.4)	0.95 (3.4)	0.0	0.0	0.0	0.0
water_nsquared	415024	14596300	3.4	3.4	0.83 (3.4)	1.00 (3.4)	0.0	0.0	0.0	0.0
water_spatial	30029	2226340	3.4	3.4	0.79 (3.4)	0.95 (3.4)	0.0	0.0	0.0	0.0
	gemessen		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Tab. 4.13: Evaluation und Modellierung des EDP für **SPLASH-2** Benchmarks auf Intel i7-6700 Skylake Prozessor.

	EDP		**Beste Freq**		s_{opt}^{EDP} (f)		**EDP-Diff** [%]		**EDP-Diff-glob** [%]	
Benchmark	min	max	p=1	p=8	p=1	p=8	p=1	p=8	p=1	p=8
barnes	14989.6	201077	2.8	2.5	1.10 (3.0)	1.33 (2.5)	1.33	0.0	1.33	1.7
cholesky	1.56215	5.20819	2.8	3.2	1.17 (2.8)	1.19 (2.8)	0.0	1.75	1.83	3.24
fmm	10192.8	134910	3.0	2.5	1.11 (3.0)	1.31 (2.5)	0.0	0.0	0.0	0.7
lu_cb	28632.6	378218	3.4	2.5	1.10 (3.0)	1.35 (2.5)	2.2	0.0	2.2	0.03
lu_ncb	39448	595360	3.2	2.7	1.12 (3.0)	1.33 (2.5)	1.6	1.15	1.6	0.0
ocean_cp	15691.6	66498.7	2.5	1.4	1.07 (3.2)	1.11 (3.0)	5.65	74.3	2.45	48.42
ocean_ncp	20022.8	80603.1	2.5	1.5	1.09 (3.2)	1.11 (3.0)	7.93	74.08	5.09	47.51
radiosity	176397	355392	3.2	2.7	1.11 (3.0)	1.19 (2.8)	1.82	0.31	1.82	0.0
radix	801.203	28531.2	3.4	2.3	1.03 (3.2)	1.24 (2.7)	0.86	0.87	3.52	0.87
raytrace	17713.1	299872	3.2	2.8	1.12 (3.0)	1.36 (2.5)	0.43	0.98	0.43	0.18
volrend	20138.8	327530	3.4	2.7	1.10 (3.0)	1.37 (2.5)	1.63	0.76	1.63	0.0
water_nsquared	200446	2021020	3.2	2.5	1.13 (3.0)	1.35 (2.5)	1.34	0.0	1.25	1.82
water_spatial	16879.3	285822	3.4	2.5	1.10 (3.0)	1.35 (2.5)	1.58	0.0	1.58	2.97
	gemessen		**gemessen**		**modelliert**	**modelliert**	**modelliert**		**modelliert**	

Tab. 4.14: Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob} und korrespondierende Frequenzen für PARSEC Benchmarks.

	Optimaler Skalierungsfaktor Energie PARSEC	
Prozessor	$p = 1$	$p = 8$
Haswell	s_{glob}=1.77 (1.9 GHz)	s_{glob}=2.04 (1.7 GHz)
Skylake	s_{glob}=1.95 (1.7 GHz)	s_{glob}=2.55 (1.4 GHz)

Tab. 4.15: Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob} und korrespondierende Frequenzen für SPLASH-2 Benchmarks.

	Optimaler Skalierungsfaktor Energie SPLASH-2	
Prozessor	$p = 1$	$p = 8$
Haswell	s_{glob}=1.14 (3.0 GHz)	s_{glob}=1.34 (2.5 GHz)
Skylake	s_{glob}=2.08 (1.7 GHz)	s_{glob}=2.42 (1.4 GHz)

4.6.4 Anwendungsunabhängige optimale Frequenzen

Das Ziel der anwendungsunabhängigen Modellierung ist es, nur einen optimalen Skalierungsfaktor s_{glob} abzuleiten, der auch gut für alle anderen Benchmarks passt. Der Ablauf zum Bestimmen eines solchen Faktors ist die Aufteilung der PARSEC und SPALSH-2 Benchmarks in jeweils ein Test-Set, bestehend aus drei Benchmarks und einem Validierungs-Set, das die übrigen Benchmarks enthält. Für das Test-Set können beispielsweise die ersten drei Benchmarks hergenommen werden. In diesem Fall wären die Test-Stets die Benchmarks Blackscholes, Bodytrack und Canneal für die PARSECs und Barnes, Cholesky und Fmm für die SPLASH-2 Benchmarks. Wie bereits bei der anwendungsspezifischen Modellierung werden die Faktoren für P_{dyn} und P_{static} über die Methode der kleinsten Fehlerquadrate bestimmt. Eine Bestimmung der Faktoren über die Levenberg-Marquard-Methode, wie sie zum Beispiel beim Approximieren in Gnuplot verwendet wird, führt aber auch zu den gleichen Ergebnissen. Aus diesen drei Werten wird dann der Durchschnitt P_{dyn}^{mean} und P_{static}^{mean} berechnet. Somit ergibt sich ein einzelner, globaler Skalierungsfaktor s_{glob}. Die Tabellen 4.14 und 4.15 zeigen die resultierenden Werte für die globalen Skalierungsfaktoren mit entsprechender Frequenz für p=1 und p=8 für die PARSEC und SPLASH-2 Benchmarks für den Haswell und Skylake Prozessor für die Energiemodellierung. Dieser globale Skalierungsfaktor wird dann für das Evaluations–Set, welches aus den Benchmarks besteht, die nicht Teil des Test–Sets waren, verwendet und die Differenz im Energieverbrauch vom tatsächlichen Energieminimum bestimmt. Die Abweichungen des Energieverbrauchs der modellierten Frequenzen und der gemessenen minimalen Energie sind in den letzten beiden Spalten der Tabellen 4.6 bis 4.9 gelistet.

Tab. 4.16: Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob}^{EDP} und die entsprechenden Frequenzen für die PARSEC Benchmarks.

	Optimaler Skalierungsfaktor EDP PARSEC	
Prozessor	$p = 1$	$p = 8$
Haswell	s_{glob}^{EDP}=1.11 (3.0 GHz)	s_{glob}^{EDP}=1.28 (2.7 GHz)
Skylake	s_{glob}^{EDP}=1.23 (2.8 GHz)	s_{glob}^{EDP}=1.60 (2.1 GHz)

Tab. 4.17: Anwendungsunabhängige optimale Skalierungsfaktoren s_{glob}^{EDP} und die entsprechenden Frequenzen für die SPLASH-2 benchmarks.

	Optimaler Skalierungsfaktor EDP SPLASH-2	
Prozessor	$p = 1$	$p = 8$
Haswell	s_{glob}^{EDP}=0.79 (3.4 GHz)	s_{glob}^{EDP}=0.86 (3.4 GHz)
Skylake	s_{glob}^{EDP}=1.13 (3.0 GHz)	s_{glob}^{EDP}=1.29 (2.7 GHz)

Bei den PARSEC Benchmarks lag auf der Haswell Architektur die prozentuale Abweichung im Durchschnitt bei 2.8 % für p=1 und 5.7 % für p=8. Die höchste Abweichung hat der Ferret Benchmark mit 11.3 % für p=1. Beim Skylake Prozessor beträgt die Abweichung für p=1 sogar nur 1.7 % und 1.4 % für p=8. Die größte Abweichung liegt mit Abstand beim Swaptions Benchmark mit 6.3 % für p=1. Überraschenderweise ist beim Haswell Prozessor die anwendungsunabhängige Modellierung akkurater als die anwendungsspezifische Modellierung. Beim Skylake sind die Abweichung bei beiden Modellen sehr klein. Aber auch hier passt das Modell beim Skylake Prozessor wieder etwas besser als beim Haswell Prozessor. Bei den SPLASH-2 Benchmarks zeigen sich ähnlich Ergebnisse: Die Abweichungen für p=1 sind sogar noch kleiner als bei den PARSECs. Für p=8 führen die beiden Ocean Benchmarks wegen ihrem irregulärem Verhalten zu größeren Abweichungen.

4.6.5 Anwendungsunabhängiges Optimum der EDP-Frequenzen

Bei der anwendungsunabhängigen Modellierung des EDP wird der gleiche Ansatz wie in der anwendungsunabhängigen Modellierung der Energie genutzt, das heißt, man hat hier wieder ein Test-Set und ein Evaluierungs-Set. Die prozentualen Abweichung sind in den letzten beiden Spalten der Tabellen 4.10 bis 4.13 aufgelistet. Die entsprechenden Skalierungsfaktoren für die jeweiligen EDP Optima sind in den Tabellen 4.16 und 4.17 notiert.

Größere Abweichungen zwischen Modell und Messung können bei zwei Benchmarks beobachtet werden: facesim und ferret auf dem Skylake Prozessor für p=8. Aufgrund irregulärem Verhaltens und der sehr geringen Laufzeit ist es für die beiden Benchmarks

schwierig im Vergleich zu den restlichen Anwendungen diese zu modellieren, da das Modell eine Skalierbarkeit voraussetzt, die hier nicht gegeben ist. Auf dem Haswell Prozessor sind die Abweichungen bei beiden Benchmarks auch auffallend, jedoch nicht ganz so gravierend. Die durchschnittliche Abweichung bei den PARSEC Benchnmarks auf dem Haswell System sind 4.8% und 8.5% für p=1 und p=8. Auf dem Skylake System sind die Abweichungen 1.8% und 10.8% für p=1 und p=8. Für p=8 sind die Abweichungen somit etwas höher als bei der Modellierung der Energie. Dies liegt daran, dass beim EDP die Laufzeit, welche schwieriger zu erfassen ist als die Leistungsaufnahme, deutlich mehr Einfluss hat, da diese beim EDP quadratisch einberechnet wird.
Bei den SPLASH-2 Benchmarks sieht es ähnlich aus. Ausnahme bilden hier wieder die beiden Ocean Benchmarks aufgrund ihrer nicht vorhandenen Skalierbarkeit.

4.7 Verwandte Arbeiten

Das in Kapitel 4.2 eingeführte Leistungs- und Energiemodell basierte bislang nur auf Basis der CPU-Frequenz und der unterschiedlichen Threadzahlen. In [35] wird ein weiteres Leistungs- und Energiemodell eingeführt welches in [36] erweitert wurde, sodass auch die Uncore-Frequenzen der Prozessoren einbezogen wird. Auch bei diesem Modell handelt es sich um ein analytisches Modell. Grundlage ist die Leistungsaufnahme verschiedener Benchmarks, wie es in 3.3 für die Splash-2 und PARSEC Benchmarks vorstellt wurde. Da die Verläufe einer Parabel ähneln, werden die Verläufe der Energie und Leistung in diesem Modell durch ein Polynom zweiten Grades angenähert. Ähnlich wie das Energiemodell in 4 verwendet das alternative Model eine Grundleistungs (statische Leistungsaufnahme) und eine dynamischen Leistungsaufnahme. Auch gleich in beiden Modellen ist der Ansatz der linearen Abhängigkeit zwischen Performance und Frequenz. Man beschränkt das Modell jedoch nur auf stark skalierbare Anwendungen. Anders als in dem in Kapitel 4.2 eingeführten Modell wird in diesem Modell die Performance in die Energie mit einbezogen. Dies sieht man in Abbildung 4.5. Der Benchmark `dgemm` aus der MKL Bibliothek wird hier auf drei verschiedenen Prozessoren jeweils über alle Kerne gemessen und die Performance in $GFlops/s$ auf der x-Achse abgebildet. Auf der y-Achse wird die Energie durch die Performance geteilt und man erhält die Energiekosten in der Einheit $pj/Flop$. Je kleiner dieser Wert ist, desto effizienter verhält sich die Anwendung bezogen auf den Energieverbrauch. Dies wurde für jeweils drei verschiedene Frequenzen gezeigt.

In [36] wird das Modell um die Uncore-Frequenz erweitert. Die Grundleistung der CPU wird nicht mehr als konstant angenommen, sondern als abhängig von der Uncore-

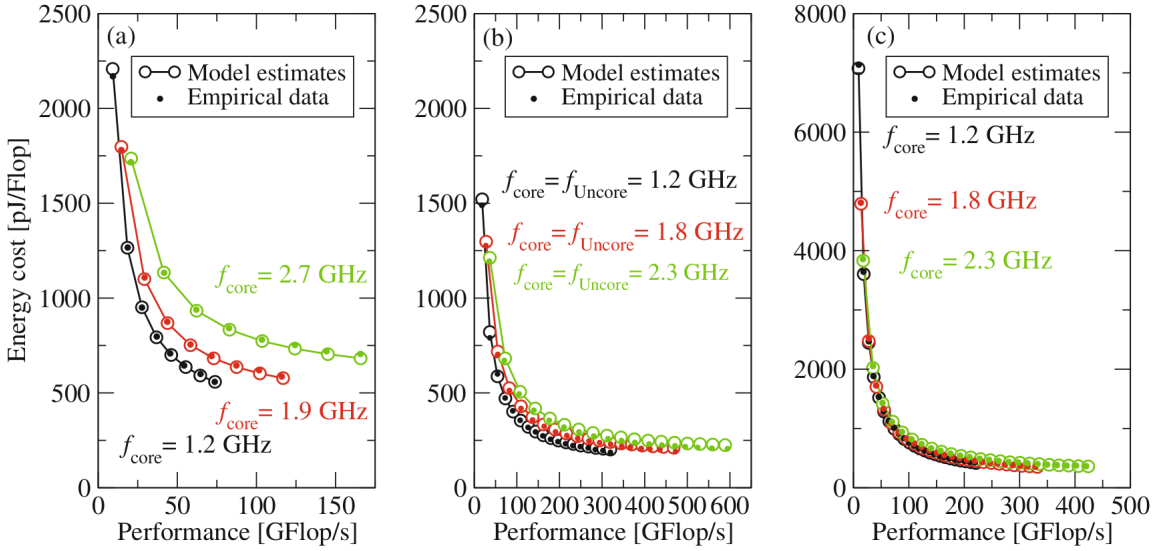

Abb. 4.5: Gemessene und modellierte Energie für DGEMM aus der Intel MKL Bibliothek, auf drei verschiedenen Prozessoren (Sandy Bridge-EP, Broadwell-EP und AMD EPYC) [36].

Frequenz. Dadurch ergibt sich eine neue Grundleistung. Solange die Bandbreite des Prozessors nicht erreicht wird, profitiert jeder Kern von einer höheren Uncore-Frequenz und die Performance steigt.Wenn eine Sättigung erreicht wird, stagniert die Leistung, aber die Energie steigt weiter. Über verschiedene Parameter werden diese Effekte aufgefangen. Dies verbessert zwar das Modell, macht es aber auch deutlich komplexer. Obwohl das Modell gute Ergebnisse liefert ist es nicht allgemeingültig. In einem großen Clustersystem werden viele verschiedene Prozessoren des gleichen Typs gleichzeitig betrieben. Durch die große Zahl kann es zu kleinen Abweichungen in der Produktion der Prozessoren kommen oder auch äußere Einflüsse im Betrieb können den einzelnen Chip beeinflussen, wie beispielsweise die Abwärme. Außerdem ist das Modell durch die große Anzahl an Eingabeparameter sehr komplex und die Parameter müssen erst bestimmt werden, bevor sie für das Modell verwendet werden

Der Ansatz, die optimalen Skalierungsfaktoren zur Minimierung des Energieverbrauchs der CPUs durch Anpassung der dynamischen Leistung und der Leckleistung wurde bereits in [41, 40, 64] für sequentielle Programme betrachtet und diskutiert. Leistungs-Modelle für Multicore-Prozessoren wurden in [27] entwickelt. Eine detaillierte Analyse von Energiemodellen in Bezug auf Scheduling-Algorithmen ist in [43] gegeben. Das Energiemodell aus diesem Kapitel wurde in dem Artikel [50] für sequentielle Tasks verwendet um damit Scheduling-Algorithmen für fork-join-Parallelität zu analysieren.

Grundlegende Aspekte von Speed-Scaling und Power-Down (Leistungssenkung) Scheduling werde in [24] beschrieben. Power-Down Mechanismen und dymanische Speed-Scaling-Techniken werden in [16] angesprochen.
Ein detailliertes Energie-Roofline-Modell, welches zugleich Details der Prozessor-Architektur in Betracht zieht, wie etwa Speicherhierarchien, wurde in [23] präsentiert. Dieses Modell wurde mit Mikro-Benchmarks für verschiedene Architekturen wie x86, ARM und GPUs, evaluiert. Im Kontrast dazu ist das eingeführte Modell sowohl anwendungsspezifisch als auch anwendungsunabhängig anwendbar. Das Modell aus diesem Kapitel abstrahiert die Details der zugrundeliegenden Hardware. Stattdessen werden anwendungsspezifische Effekte durch Parameter des Energiemodells eingefangen, die von gemessenen Laufzeiten und Energie-Werten abgeleitet werden.
Anwendungsspezifische thermische Energiemodelle die sowohl hardwarespezifische als auch anwendungsspezifische Parameter instrumentieren sind in [33] beschrieben. Dieser Artikel beschreibt ein detailliertes thermisches Hardware-Modell mit einer experimentellen Verifikation durch Messungen direkt auf der Hardware.
In [44] wird auch eine Temperatur-orientierte Optimierung der Energie verfolgt. Hohe Temperaturen wie sie beispielsweise bei sequentiellen und rechenintensiven Tasks vorkommen können, drosseln die Frequenz. Da die Kerne auf dem Chip nah beieinander liegen, hat man eine thermische Abhängigkeit der einzelnen Kerne. Wenn ein Kern zu heiß wird, drosseln gegebenenfalls somit auch die benachbarten Kerne. Durch alternierenden Einsatz der einzelnen CPU-Kerne und gezieltem DVFS auf den einzelnen Kernen versucht man über einen Scheduling-Ansatz von Tasks einer Überhitzung entgegenzuwirken. In [37] wird der Einfluss des Workloads auf verschiedene heterogenen Architekturen wie der big.LITTLE Architektur untersucht. Bei der big.LITTLE Architektur wirken sich die Workloads anders, je nachdem, auf welchen Kernen der Workload liegt. Hier zeigen die Resulatate, dass das Einbeziehen des Workload-Typs in den Scheduler zwischen 7.1 % und 31.3 % Energie einsparen kann.

4.8 Zusammenfassung der Modellierung

In Abschnitt 4.6 wurden das analysitsche Leistungs- und Energiemodell aus Kapitel 4.2 ausgewertet, um die anwendungsspezifische und anwendungsunabhängige optimalen Frequenzen vorherzusagen. Die anwendungsspezifischen Modelle wurden durch Berechnung der Werte für die dynamische und statische Leistungsaufnahme abgeleitet, siehe Tabelle 4.11, und durch Berechnung der optimalen Frequenzen in den Tabellen 4.6 – 4.13, entsprechend den Gleichungen in Kapitel 4.2, gezeigt. Folgendes Verhalten und

Qualität der Modelle wurde festgestellt:

- **Benchmark Suites**: Die berechneten opimalen Skalierungfaktoren für die PARSEC und SPLASH-2 Benchmarks sind sich auf dem Skylake Prozessor recht ähnlich. Dies kann in Abbildung 4.3 gesehen werden, wo die anwendungsunabhängigen Skalierungsfaktoren nahezu gleich sind. Auf der Haswell Architektur sieht es anders aus, siehe Abbildung 4.4. Dort sind die Skalierungsfaktoren für die SPLASH-2 Benchmarks signifikant größer als die Skalierungsfaktoren für die PARSEC Benchmarks. Ein Grund dafür könnte die höhere Intensität der Berechnungen der SPLASH-2 Benchmarks sein, die auf dem Haswell Prozessor eine höhere Auswirkung haben als auf dem Skylake.
- **Prozessor-Architektur**: Die optimalen Skalierungsfaktoren sind generell kleiner als die auf dem Haswell Prozessor. Dies kann sowohl bei der Modellierung der Energie als auch des EDP beobachtet werden. Bei den SPLASH-2 Benchmarks ist dieser Effekt besonders deutlich. Somit ist die minimal verbrauchte Energie auf dem Skylake Prozessor bei kleineren Frequenzen zu verbuchen, als beim Haswell Prozessor.

Insgesamt lässt sich sagen, dass die Modellierung, speziell für die Energie, sehr gut funktioniert. Ausnahmen bilden hauptsächlich Benchmarks mit irregulärem Verhalten. Anwendungen die nicht oder nur sehr schlecht mit der Ferquenz und der Threadzahl skalieren, können nicht oder nur sehr schlecht modelliert werden, da die Modelle eine gewissen Skalierbarkeit voraussetzen. Die Abweichungen der Energie bei der Modellierung über das EDP sind etwas größer als bei der Energie, insbesondere bei p=8.Beim EDP wirkt sich die Laufzeit deutlicher auf die Modellierung aus als bei der Energie, da diese beim EDP quadratisch einbezogen wird. Durch die Schwierigkeit, die Laufzeit nachzubilden entstehen somit beim EDP etwas größere Abweichungen.
Auch die anwendungsunabhängige Modellierung zeigte gute Ergebnisse. Zu beachten ist hier jedoch, dass man in das Test-Set keine Benchmarks mit irregulärem Verhalten aufnimmt. Dies würde zu größeren Abweichungen führen. Wählt man das Test-Set aus Anwendungen, die gut skalieren erhält man einen guten allgemeinen Skalierungsfaktor. Die optimalen Frequenzen, bezogen auf den Energieverbrauch, bewegen sich in einem gewissen Frequenz-Bereich. Ohne die Ausreißer würde sich somit bei dem globalen Skalierungsfaktor wenig ändern, wenn man das Test-Set ändert.

5 | DVFS auf GPUs am Beispiel eines Autotuning Frameworks für energieeffizientes Kryptomining

In den bisherigen Kapiteln wurde DFVS auf CPUs untersucht. Diese sind jedoch nicht die einzigen "Prozessoren" auf dem Markt. Auch bei Grafikkarten handelt es sich genau genommen um Prozessoren und auch diese verfügen heutzutage über die DVFS Technik. Genau um diese geht es in diesem Kapitel. Unter Verwendung eines eigens entwickelten Autotuners wird das Verhalten und Potential von DVFS auf Grafikkarten untersucht. Das Autotuning soll die Energiebilanz für Programme auf GPUs automatisch verbessern. GPU steht für Graphics Processing Unit. Bei Berechnungen auf GPUs, die weniger mit Visualität zu tun haben, spricht man auch von GPGPU Programmierung (engl. *General Purpose Computation on Graphics Processing Unit*). Darunter fallen beispielsweise das Lösen von linearen Gleichungssystem (LGS, engl. ODE) oder aktuell auch das Thema Krytomining. Da Kryptowährungen wie Ethereum und Bitcoin den Grafikkartenmarkt in den letzten Jahren ziemlich durcheinander gebracht haben und Grafikkarten stark für das Hashen von Kryptowährungen verwendet werden, werden verschiedene Mining-Algorithmen für Kapitel als Anwendungsfall verwendet. Da der Stromverbrauch enorm die Rentabilität des Hashens von Kryptowährungen beeinflusst, erwies sich das Thema als sehr passend. Alternativ wären auch andere Anwendungsgebiete, wie zum Beispiel verschiedene Simulationen oder das Rendern von Videodateien denkbar. Nach einer kurzen Einführung in die Welt der Kryptowährungen liegt der Fokus auf der Evaluierung des Autotunings und den Messergebnissen. Man bekommt dadurch einen guten Einblick in den grundsätzlichen Aufbau und die Umsetzung eines Autotuners, denn die grundsätzlichen Ansatzpunkte, wie der Einsatz von Suchstrategien, werden so gut wie von jedem Autotuner genutzt.Zudem werden spezielle Aspekte, wie die Formulierung einer Zielfunktion, gezeigt.
Das Framework für den Autotuner wurde gemeinsam während einer Masterarbeit [7] entwickelt. In [5] findet sich entsprechende Publikation zu diesem Kapitel. Das entwickel-

te Autotuning Framework steht frei auf Github unter [91] zum Download bereit. Wenn nicht anders angegeben, wird sich auf diese beiden Quellen bezogen. Weiterführende Quellen werden speziell genannt.

5.1 Motivation und Aufbau des Kapitels

Der Einsatz von GPUs in modernen Supercomputern erhöht die Rechenleistung dieser enorm. Der in Kapitel 1.2 eingeführte und in der TOP500 Liste vertretene Supercomputer Summit hat bei 4608 Rechenknoten über zwei IBM POWER9 Prozessoren und sechs NVIDIA Tesla V100 GPUs pro Knoten. Die maximale Leistungsaufnahme liegt für den Superrechner bei insgesamt 13 MW. Davon entfallen bei 300 W Peak-Leistung pro GPU rund 8.29 MW auf die Grafikkarten (vgl. [82]). Automatische Optimierungen der Energieeffizienz zur Laufzeit sind somit auch auf GPUs sehr sinnvoll und sollten bei rechenintensiven Programmen eingesetzt werden, um die Stromkosten zu senken. Im besten Fall sollten diese Optimierungen keinen negativen Einfluss auf die Performance eines Programms haben. Am Beispiel eines eigens entwickelten Autotuning-Frameworks wird in diesem Kapitel gezeigt, dass dies durchaus möglich ist. DVFS auf Grafikkarten bringt viel Potential für Energieeinsparungen mit sich. Zur Evaluation unseres Frameworks und des DVFS Verhaltens dienen verschiedene Blockchain Mining-Algorithmen, die auf unterschiedlichen NVIDIA Grafikkarten im Energieverbrauch optimiert werden. Dieses Kapitel ist nach der Einleitung wie folgt aufgebaut:

- Kapitel 5.2 gibt einen Überblick über die wichtigsten technischen Grundlagen. Unter anderem gibt es dort eine Einführung in die Blockchain-Technologie und in die Funktionsweise von GPUs in diesem Kontext. Außerdem wird gezeigt, wie die Energiemessung auf GPUs funktioniert und welche Bibliotheken dafür benötigt werden. Zudem wird auf die Funktionsweise von DVFS und die Einschränkungen eingegangen. Auch hierfür sind verschiedene Bibliotheken notwendig. Abgeschlossen wird das Unterkapitel mit einer Übersicht der Eigenschaften von den Grafikkarten, die für die Evaluierung genutzt wurden.
- Kapitel 5.3 geht genauer auf den Aufbau und die Funktionsweise des Frameworks ein. Hauptbestandteil sind zwei Tuning-Phasen (Frequenzoptimierungsphasen) und das Monitoring. Außerdem werden die drei eingesetzten Suchalgorithmen Hill Climbing, Simulated Annealing und Nelder-Mead beschrieben.
- Kapitel 5.4 evaluiert die Auswirkungen von DVFS auf GPUs mithilfe des Autotuning-Frameworks. Zuerst werden dort die für die experimentelle Auswertung verwendeten Kryptowährungen eingeführt. Dabei liegt der Schwerpunkt auf den drei

Hauptwährungen Ethereum, Monero und ZCash, welche hier vorab mit einem NVIDIA-Profiler auf Speicherbandbreite und Instruktionen pro Taktzyklus untersucht wurden. Danach wird über eine breite Suche (exhaustive search) das Energieoptimum auf verschiedenen Grafikkarten bestimmt. An diesen Optima werden im Anschluss die drei Suchalgorithmen gemessen und bewertet. Auch wird die Zeit für das Finden der Optima der einzelnen Suchalgorithmen gegenübergestellt. Es folgt noch eine Optimierung unter Nebenbedingungen. Hier wird eine Mindestperformance als Nebenbedingung eingebaut. Abschließend wird das Monitoring untersucht, welches die Kryptowährungen während der Laufzeit aufgrund von Leistung und Kosten austauschen konnte und so jeder Grafikkarte die profitabelste Währung zuodnen konnte.

- Kapitel 5.5 fasst dieses Kapitel und die wichtigsten Ergebnisse der Evaluation nochmal einmal kurz zusammen.
- Kapitel 5.6 gibt einen Überblick über verwandte Arbeiten.
- Kapitel 5.7 gibt einen Ausblick über das Autotuning und dem DVFS auf Grafikkarten.

5.2 Technischer Hintergrund

In diesem Kapitel steht der technischen Hintergrund im Vordergrund. Hierfür gibt es eine kurze Einführung in die Blockchains Technologie und in die Arbeitsweise von Grafikkarten in diesem Kontext. Außerdem wird gezeigt, wie man DVFS auf GPUs verwenden kann und wie der Energieverbrauch ausgelesen werden kann. Dazu sind verschiedene Bibliotheken zu unterscheiden.

5.2.1 Einführung in das Autotuning

Beim Autotuning wird die Suche nach Werten automatisiert, mit denen die Eigenschaften einer Zielanwendung beeinflusst werden. Die Eigenschaften sind beispielsweise die Laufzeit, die Performance einer Anwendung, die man minimieren will, oder die Energieeffizienz, die man maximieren will. Mithilfe von Autotunern können größere Parameterräume, die Einfluss auf die Performance von Anwendungen haben, effizient durchsucht werden. Die Zeit für die Optimierung gestaltet sich in diesem Fall sehr effizient [17]. Da solche Parameterräume sehr vielseitig sein können, braucht es effektive Suchstrategien dafür. Beim Energie-Tuning spielen wie auch schon bei den Energiemodellen die Frequenzen der Prozessoren eine wichtige Rolle. Aber nicht nur diese haben einen Einfluss: Blockgrößen und Implementierungsvarianten können einen starken

Einfluss auf die Performance haben und somit auch das Energieverhalten sehr stark verändern. Es gibt nach [58] drei Kategorien von Autotunern:

1. Selbstoptimierende Bibliotheksgeneratoren wie ATLAS [62], PhiPAC [20]und OSKI [60] für lineare Algebra und FTTW [29] und SPIRAL [45] für die Signalverarbeitung.
2. Compiler-basierende Autotuner, welche automatisch eine Menge von alternativen Implementierungen einer Berechnung generieren und durchsuchen. Als Beispiel sei hier das Framework CHiLL der University of Utah[1] erwähnt.
3. Autotuner auf Anwendungsebene, welche die empirische Suche über eine Menge von Parameterwerten, die vom Anwendungsprogrammierer vorgeschlagen werden, automatisieren.

Bekannte Vertreter des dritten Typs sind OpenTuner [17] und Active Harmony [56]. Es wäre problemlos möglich gewesen, einen fertigen Autotuner zu verwenden. Durch den eigenen Autotuner war es jedoch einfacher, spezielle Anpassungen für die Kryptowährungen, wie beispielsweise verschiedene Autotuning-Phasen, zu realisieren.

5.2.2 Einführung in die Blockchain

Als Anwendungsfeld für den Autotuner wurden verschiedene Kryptowährungen verwendet, welche auf der Blockchain-Technologie basieren. Eine Blockchain ist eine Liste an Daten, auch als Block bezeichnet, die durch einen krytographischen Hash miteinander verbunden sind. Jeder Block enthält einen Header und einen Body. Der Body beinhaltet beendete Transaktionen und deren Hashwerte, die in einem Hash-Tree (Merkle Tree) gespeichert werden. Der Header speichert die Wurzel des Baums, einen Zeitstempel und eine Zufallszahl, welche als *nonce* bezeichnet wird. Zusätzlich besitzt der Header den Hash-Wert des vorherigen Blocks. Dies garantiert eine nicht-modifizierbare Transaktion. Abhängig von der Blockchain kann der Header auch noch mehr Informationen speichern. Auch kann sich die Hashing-Funktion von Blockchain zu Blockchain unterscheiden.
Eine Blockchain wird dezentral verwaltet. Es gibt also keine zentralen Server, auf dem die Blockchain gespeichert wird. Stattdessen wird ein Peer-to-Peer (P2P) Netzwerk verwendet. Jeder Teilnehmer speichert die gesamte Blockchain. Sobald ein neuer Block generiert wurde, wird dieser Block erst lokal von jeden Knotenpunkt verifiziert, bevor dieser in die Blockchain eingepflegt wird. Um vor Fehlern oder Angriffen geschützt zu sein, verwenden unterschiedliche Blockchains unterschiedliche Arten von Sicherheit.

[1] **https://www.osti.gov/servlets/purl/1156961**.

Eine davon ist das Proof-of-Work (PoW). PoW Teilnehmer werden Miner genannt. Ein Miner kann einen neuen Transaktions-Block nur dann hinzufügen, wenn er eine Zahl X (*nonce*) berechnet, die folgende Ungleichung erfüllt (5.1):

$$hash(block\_data \,||\, X) < Y \tag{5.1}$$

Die gesuchte Zahl X ist hier die *nonce* und die Zahl Y wird als *target* bezeichnet. Der generierte Hash aus dem Block und der *nonce* muss also kleiner sein als die *target*. Damit das Minen eines Blocks immer in etwa die gleiche Dauer hat, wird die *target* dynamisch vom Netzwerk festgelegt. Diese wird an die gesamten Rechenleistung des Netzwerks angepasst. Die Schwierigkeit einen Block zu minen (engl. difficulty) hängt somit von *target* ab. Je kleiner dieses ist, desto weniger Zahlen existieren, die die Ungleichung erfüllen. Hat ein Miner Gleichung (5.1) gelöst, bedeutet dies, er hat den Block gemint [81].
Das Minen eines Blocks erfolgt über Brute-Force, das heißt, es werden alle Möglichkeiten für X eingesetzt und ausprobiert. Als Belohnung für das Minen eines Blocks gibt es einen Block-Reward [80].
Es erfordert also eine Menge Rechenleistung und Parallelisierung. GPUs eigenen sich also schon einmal grundsätzlich zum schnellen Minen. Jedoch werden für die Berechnungen jede Menge Energie benötigt. Da es fast unmöglich ist, einen Block selbst zu finden, schließen sich Miner in sogenannten Mining-Pools zusammen, um die Rechenleistung zu bündeln. Jeder Miner bekommt vom Mining-Pool einen bestimmten Zahlenbereich zugeordnet, den er per Brute-Force ausprobiert. Der Miner mit der höchsten Rechenkraft bekommt dementsprechend den meisten Reward.
Die drei bekanntesten Währungen, die für die Auswertung des Frameworks verwendet wurden, sind Ethereum (ETH), Monero (XMR) und ZCash (ZEC).
Alternativ zu PoW exisiert das Proof-of-Stake (PoS) bei dem eine gewisse Anzahl an Token hinterlegt wird. Alle untersuchten Kryptowährungen verwenden jedoch PoW (aktuell auch noch Ethereum), weswegen nicht genauer auf PoS eingegangen wird.

5.2.3 Einblick in die Arbeitsweise von GPUs in diesem Kontext

Grafikkarten sind massiv parallele Manycore-Prozessoren, die verglichen mit CPUs eine hohe Leistungsfähigkeit und eine große Speicherbandbreite vorweisen. Auf Grafikkarten kommt eine deutlich höhere Anzahl an Transistoren für das Data-Processing zum Einsatz als auf CPUs, bei denen eine hohe Anzahl an Transistoren stattdessen für Caches und Kontrolleinheiten verwendet wird. Hintergrund sind die unterschiedlichen Einsatzzwecke:

CPUs sind dafür konzipiert, die Latenz für einen Thread zu minimieren, wohingegen GPUs dafür konzipiert wurden, einen hohen Durchsatz für alle Threads zu maximieren. Auf einer GPU entspricht ein Thread einer Sequenz an SIMD Operationen (Single Instruction Multiple Data). Somit eignen sich GPUs gut, um gleiche Berechnungen parallel auf unterschiedlichen Daten auszuführen, ganz im Stil von SIMD. Das Ausführen gleicher Instruktionen auf unterschiedlichen Daten erlaubt es, Speicherzugriffszeiten (Latenzen) zu verstecken.
Die verschiedenen Hashing-Algorithmen, die beim Mining von Kryptowährungen verwendet werden, nutzen diesen Umstand während des Proof-of-Work (PoW) aus. Um den PoW zu lösen, können verschiedene Zahlen (*nonces*) parallel mit dem gleichen Hashing-Algorithmus geprüft werden. Je mehr solcher Nonces parallel geprüft werden können, desto schneller kann eine Lösung für das PoW gefunden werden. Die Anzahl an geprüften Nonces pro Sekunde wird als Hashrate bezeichnet. Je höher die Hashrate ist, desto leistungsfähiger ist die GPU beim Mining.

5.2.4 Energiemessung auf GPUs

Die Energiemessung auf den Grafikkarten erfolgt mithilfe der *NVML* [86] Bibliothek. Hierfür wurde ein eigenes Modul geschrieben, welches für jede GPUs im Hintergrund den Energieverbrauch periodisch misst. Die Ausgabe erfolgt in Watt und wird über eine Zeitperiode gemessen und anschließend gemittelt, um die Energie in Joule zu bekommen. Zum Plotten der Daten wird die Energie und die Systemzeit in einer Datei gespeichert.

5.2.5 DVFS auf GPUs

Dynamische Frequenzanpassung erlaubt es auch auf Grafikkarten, bestimmte Frequenzen zur Laufzeit zu verändern. Die beiden einstellbaren Frequenzen auf den GPUs sind die *VRAM*-Frequenz, also die Frequenz des Hauptspeichers, und die *Core*-Frequenz, also die Frequenz der Recheneinheiten. Oft spricht man bei der VRAM- bzw. Core-Frequenz auch von der VRAM-Clock bzw. Core-Clock. Hierfür kommen neben der *NVML* je nach Betriebssystem auch noch andere Bibliotheken zum Einsatz. Auf Windows verwendet man zusätzlich die *NVAPI* [85] Bibliothek und auf Linux die *NV-Control X* [84] Bibliothek. Die Frequenzanpassung ist als eigenständige Funktion implementiert worden, welche die Parameter `device_clock_info`, die VRAM-Frequenz, welche als Differenz zur Standardfrequenz angegeben wird, sowie die Core-Frequenz als Index eines Vektors mit allen verfügbarern Frequenzen übergeben bekommt. Die Core-Frequenz kann nicht beliebig gewählt werden, sondern muss wie auf CPUs aus einem verfügbaren Bereich

ausgewählt werden. Das Festsetzen der Frequenzen hängt davon ab, welche APIs (Application Programming Interface) auf der jeweiligen GPU unterstützt werden. Es gibt hier drei mögliche Szenarien:

- **NVML und NVAPI/NV-Control X:** Core- und VRAM-Frequenz sind in vollem Bereich einstellbar.
- **Nur NVML:** Die Core-Frequenz kann in einem leicht eingeschränkten Wertebereich gesetzt werden und die VRAM-Frequenz kann nicht festgesetzt werden.
- **Nur NVAPI/NV-Control X:** Die Core-Frequenz kann in stark eingeschränktem Wertebereich und die VRAM-Frequenz kann in vollem Wertebereich festgesetzt werden.

Listing 5.1 zeigt die dazugehörende Struktur, die zum Speichern der Werte für jede Grafikkarte verwendet wurde. Es ist natürlich auch möglich, die GPUs zu übertakten. Dies ist jedoch nicht zu empfehlen und kann die GPUs beschädigen. Auch die Energieeffizienz verschlechtert sich deutlich nach den Versuchen im übertakteten Bereich.

```
1 struct device_clock_info{
        int device_id_;
3       bool nvml_supported_, nvapi_supported_;//nvapi == nv-control x under linux
        int default_mem_clock_, default_graph_clock_;
5       std::vector<int> available_graph_clocks_;
        int min_mem_oc_, max_mem_oc_;
7       int min_graph_oc_, max_graph_oc_;
};
```

Listing 5.1: Struktur für die unterstützten Frequenzen auf den Grafikkarten (device_clock_info) (aus [91])

In Kapitel 5.2.6 gibt es eine Übersicht über die eingesetzten Grafikkarten und die unterstützen APIs.

5.2.6 Hardware Setup

Tabelle 5.1 gibt eine Übersicht über die für die experimentellen Auswertungen genutzten Grafikkarten mit Informationen zur Hardware. Zudem ist zu sehen, welche der APIs aus Abschnitt 5.2.5 zum Takten der Frequenzen zur Verfügung stehen. Zudem ist der einstellbare Frequenzbereich abgebildet.

	TITAN X	**TITAN V**	**GTX 1080**	**P4000**
Architektur	Pascal	Volta	Pascal	Pascal
CUDA-Kerne	3584	5120	2560	1792
Speichertyp	12GB GDDR5X	12GB HBM2	8GB GDDR5X	8GB GDDR5
Speicherbandbreite	480GB/s	652GB/s	320GB/s	243GB/s
Power-Cap	250W	250W	180W	105W
NVML	Ja	Ja	Nein	Ja
NVAPI	Ja	Ja	Ja	Nein
NV-Control X	Ja	Nein	Ja	Nein
VRAM-Frequenz [MHz]	4005-6005	850-1050	4005-6005	3802
Core-Clock [MHz]	139-2011	139-2012	1711-2011	139-1708

Tab. 5.1: Auflistung der verwendeten Grafikkarten, die zur Auswertung genutzt wurden

5.3 Funktionsweise des Autotuning Frameworks

Um den Einfluss von DVFS auf den Energieverbrauch von Grafikkarten zu untersuchen, wurde im Rahmen einer Abschlussarbeit [7] ein Autotuning-Framework in C++ für NVIDIA GPUs entwickelt. Als Anwendungsfall wurde das energieoptimale Mining auf NVIDIA GPUs verwendet. Für eine detailiertere Beschreibung der Implementierung sei auf [7] verwiesen. Der Quellcode dazu findet sich frei unter [91] zum Download.

Das Framework ermöglicht das Mining von Kryptowährungen unter energieoptimalen Bedingungen auf allen im System verbauten GPUs. Hierfür werden über das DVFS die optimalen Frequenzen der GPUs dynamisch bestimmt. Es wird zudem sichergestellt, dass immer die profitabelste der verfügbaren Währungen auf jeder GPU gemint wird. Das entwickelte Framework teilt sich in zwei Autotuning Hauptphasen ein: Die Frequenzoptimierungsphase (offline und online, Kapitel 5.3.2) und die Monitoringphase (Kapitel 5.3.3). Als Suchstrategien wurden Hill Climbing, Simulated Annealing und Nelder-Mead implementiert, die in 5.3.1 genauer beschrieben werden.

5.3.1 Optimierungsverfahren

Das Finden von optimalen Parametern beim Autotuning erfolgt überlicherweise über verschiedene Suchstrategien. Im Framework wurden drei der bekanntesten Optimierungsverfahren umgesetzt:

- Hill Climbing
- Simulated Annealing
- Nelder-Mead [66]

Die Zielfunktion

$$f : [core_{min}, core_{max}] \times [vram_{min}, vram_{max}] \longmapsto \mathbb{R}_0^+$$

mit $core_{min}, core_{max}, vram_{min}, vram_{max} \in \mathbb{N}$ bildet die einstellbaren Frequenzbereiche einer GPU auf die Anzahl der Hashes pro Joule $[\frac{Hashes}{Joule}]$ ab. Die Parameter $core_{min}$ und $core_{max}$ entsprechen dem Bereich der einstellbaren Core- und $vram_{min}$ und $vram_{max}$ den VRAM-Frequenzen. Berechnet wird die Zielfunktion durch die Gleichung 5.2:

$$f(vram, core) = \frac{hashrate(vram, core)\ [\frac{Hashes}{Sekunde}]}{energy\_consum(vram, core)\ [\frac{Joule}{Sekunde}]}. \tag{5.2}$$

Das Ziel der Verfahren ist es, im einstellbaren Frequenzbereich diesen Wert zu maximieren. Der einstellbare Wertebereich wird zu Programmstart für jede Grafikkarte einzeln ermittelt und abgespeichert. Je nach unterstützten API unterscheidet sich dieser Bereich auf den einzelnen GPUs. So ist je nach unterstützten APIs die Zielfunktion aus Gleichung 5.2 null-dimensional (keine Optimierung möglich), eindimensional (Core-Frequenz optimierbar) oder zweidimensional (beide Frequenzen optimierbar). In den nachfolgenden Unterkapiteln werden die Optimierungsverfahren genauer beschrieben.

Hill Climbing

Beim Hill Climbing handelt es sich um ein einfaches, heuristisches Optimierungsverfahren, das die Zielfunktion schrittweise verbessert. Es wird dabei jeweils eine lokale Veränderung durchgeführt. Sollte der neue Lösungskandidat als besser bewertet werden, wird dieser übernommen. Wenn vom aktuellen Punkt aus keine Verbesserung mehr möglich ist oder eine Schranke erreicht ist, beispielsweise eine vom Nutzer festgelegte maximale Anzahl an Iterationen oder die Lösung eine gewisse Güte erreicht, terminiert das Verfahren. Bei der Evaluierung wurde die maximale Anzahl der Iterationen auf fünf gesetzt. Bei lokalen Optima kann es jedoch zu Problemen beim Hill Climbing kommen und die Suchstrategie steckt fest.

Bei der im Framework implementierten Variante wird für die lokale Veränderung in jeder Dimension eine Schrittweite spezifiziert, um mehrere Frequenzstufen ein einmal zu überspringen, da es oft kaum Unterschiede gibt, wenn man die Frequenzen nur um ein paar wenige MHz ändert. Vom aktuellen Punkt ausgehend werden dann, in jeder Iteration abwechselnd, die vier horizontalen oder diagonalen Nachbarspunkte betrachtet. Solange die zweite Ableitung positiv ist, also solange die Steigungstangente

der Steigungstangente auch positiv steigt, wird in Richtung des besten Nachbarpunktes weitergesucht. Erst dann beginnt die nächste Iteration. Dadurch wird ein Maximum effizienter gefunden. Algorithmus 1 zeigt den Ablauf der Hill Climbing. Visualisiert man die Core-Frequenz und die VRAM-Frequenz als Gitter, so ist ein Knoten (eng. node) auf dem Gitter ein Tupel aus Core- und VRAM-Frequenz.

Algorithm 1: Ablauf des Hill Climbing (aus [7])

Input: Zielfunktion f, *Startknoten*, *Schrittweite*, *Iterationen*

Output: Frequenzen zur Maximierung von f

```
1  AktuellerKnoten = Startknoten
2  BesteAuswertung = f(AktuellerKnoten)
3  BesterKnoten = AktuellerKnoten
4  for i = 1 to Iterationen do
5      L = Nachbarknoten(AktuellerKnoten, Schrittweite)
6      tempBesteAuswertung = −INF
7      LetzterKnoten = AktuellerKnoten
8      foreach Knoten n in L do
9          if f(n) > tempBesteAuswertung then
10             AktuellerKnoten = n
11             tempBesteAuswertung = f(n)
12     BesteRichtung = AktuellerKnoten − LetzterKnoten
13     while f''(AktuellerKnoten) > 0 do
14         AktuellerKnoten = AktuellerKnoten + BesteRichtung
15         tempBesteAuswertung = f(AktuellerKnoten)
16     if empBesteAuswertung > BesteAuswertung then
17         BesterKnoten = AktuellerKnoten
18         BesteAuswertung = tempBesteAuswertung
19 return BesterKnoten
```

Simulated Annealing

Das Optimierungsverfahren Simulated Annealing ist dem Hill Climbing recht ähnlich. Konnte das Hill Climbing noch in lokalen Optima hängenbleiben, so kann das Simulated Annealing ein solches lokales Optimum wieder verlassen, um ein besseres zu finden, indem ungünstigere Zwischenlösungen mit einer gewissen Wahrscheinlichkeit akzeptiert werden. Somit ist es unwahrscheinlicher, in einem lokalem Optimum hängenzubleiben.

Algorithm 2: Ablauf des Simulated Annealing (aus [7])

Input: Zielfunktion f, *Startknoten*, *Schrittweite*, *Iterationen*
Output: Frequenzen zur Maximierung von f

$AktuellerKnoten = Startknoten$;
$BesteAuswertung = f(AktuellerKnoten)$;
$BesterKnoten = AktuellerKnoten$;
for $k = 1$ **to** *Iterationen* **do**
    /* Finde Nachbarn über eine Iteration des Hill Climbing, Algorithmus 1 */
    $NeuerKnoten = HillClimbing(f, AktuellerKnoten, Schrittweite, 1)$;
    if *f(NeuerKnoten)* $\geq$ *f(AktuellerKnoten)* **or** $exp\left(-\frac{f(AktuellerKnoten)-f(NeuerKnoten)}{T(k)}\right) > Zufall(0,1))$ **then**
        $AktuellerKnoten = NeuerKnoten$;
    if *f(AktuellerKnoten)* $>$ *BesteAuswertung* **then**
        $BesterKnoten = AktuellerKnoten$;
        $BesteAuswertung = f(AktuellerKnoten)$;
return *BesterKnoten*;

Die Wahrscheinlichkeit, ungünstigere Zwischenlösungen (a) zu akzeptieren, ist durch die Gleichung (5.3)

$$P(Akkzept.\ a\ als\ Lösungskandidaten) = exp\left(-\frac{f(b)-f(a)}{T(k)}\right),\ f(b) > f(a) \quad (5.3)$$

beschrieben. Dabei ist $T(k)$ die Temperatur der Iteration k, die mit jeder Iteration, die das Simulated Annealing durchläuft, sinkt und a oder b als neuer oder alter Lösungskandidat angenommen wird. Je größer die Temperatur ist, desto größer ist die Wahrscheinlichkeit, schlechtere Zwischenlösungen zu akzeptieren. Die Starttemperatur sollte größer als die Differenz von Maximum und Minimum von f sein, siehe Gleichung (5.4):

$$T(1) > max_x(f(x)) - min_x(f(x)), \quad \forall k \in \mathbb{N} : T(k+1) < T(k). \quad (5.4)$$

Der Rest des Algorithmus erfolgt dann mit dem Hill Climbing. Mit abnehmendem T verhält sich dieses Verfahren immer mehr wie das Hill Climbing. Das Finden neuer Lösungskandidaten erfolgt anschließend identisch zum Hill Climbing. Algorithmus (2) veranschaulicht das Verfahren. Auch hier wird als Knoten wieder ein Tupel aus Core- und VRAM-Frequenz bezeichnet.

Nelder-Mead

Die Idee hinter dem Nelder-Mead Verfahren ist es, ein Optimierungsverfahren zu implementieren, das eine Folge von Simplexen erzeugt, die einen immer kleiner werdenden Durchmesser aufweisen, um das gesuchte Optimum einzukreisen. Ein Simplex ist das einfachste Volumen im N-dimensionalen Raum, welches aus $N+1$ Punkten aufgespannt wird. Deswegen wird das Nelder–Mead Verfahren oft auch als Simplex–Verfahren bezeichnet. Im Eindimensionalen ist das eine Linie, im Zweidimensionalen ein Dreieck und im Dreidimensionalen ein Tetraeder.

Mit einem initialen Simplex $()x_0, x_1, \dots x_N)$ sowie den Parametern $\alpha > 0$, $\gamma > 1$, $0 < \beta < 1$ und $0 < \sigma < 1$ startet die Suche. Die $N+1$ Punkte des Simplex werden in jeder Iteration nach Funktionswert $f(x_0) \geq f(x_1) \geq \cdots \geq f(x_N)$ geordnet. Dann wird der Mittelpunkt $m = \frac{1}{N}\sum_{i=0}^{N-1} x_i$ der N besten Punkte gemäß der Zielfunktion f berechnet und der Punkt $r = (1+\alpha)m - \alpha x_N$ bestimmt. Anschließend wird der schlechteste Punkt x_N entweder durch:

1. den besseren der beiden Punkte $e = (1+\gamma)m - \gamma x_N$ und r ersetzt, falls $f(r) > f(x_0)$.
2. r ersetzt, falls $f(r) > f(x_{N-1})$.
3. $c = \beta m + (1-\beta)h$ ersetzt, falls $f(c) > f(x_N)$. Dabei ist h der bessere der Punkte x_N und r.

Dabei heißen die Punkte e, r und c expandierter, reflektierter und kontrahierter Punkt, da diese jeweils durch Expansion/Reflexion/Kontraktion von x_N am Mittelpunkt m entstehen. Falls keiner der Fälle zutrifft, wird der Simplex in Richtung des besten Punktes x_0 verkleinert: $\forall\, i \in [1, N] : x_i = \sigma x_0 + (1-\sigma)x_i$ (siehe Abbildung 5.1 links). So wird in jeder Iteration der Simplex in Richtung des Optimums verlagert, wie es in Abbildung 5.1 rechts dargestellt ist. Wenn die maximale Anzahl an Iterationen erreicht ist, der Simplex eine bestimmte Größe unterschreitet ($max_{0 \leq i < j \leq N}(||x_i - x_j||_2) < \epsilon$) oder die Funktionswerte des Simplex nah genug aneinander liegen ($f(x_0) - f(x_N) < \epsilon$), terminiert der Algorithmus.

Eine Implementierung exisitert bereits in der Eigen-Bibliothek in [66]. Diese Bibliothek wurde auch in der Implementierung verwendet.

5.3.2 Frequenzoptimierungsphasen

Dieses Kapitel beschreibt die beiden Frequenzoptimierungsphasen. Bei den Optimierungsphasen läuft für jede GPU separat einen Thread im Hintergrund, der die energieoptimalen Frequenzen für jede auf der GPU zugewiesenen Währungen mithilfe eines der

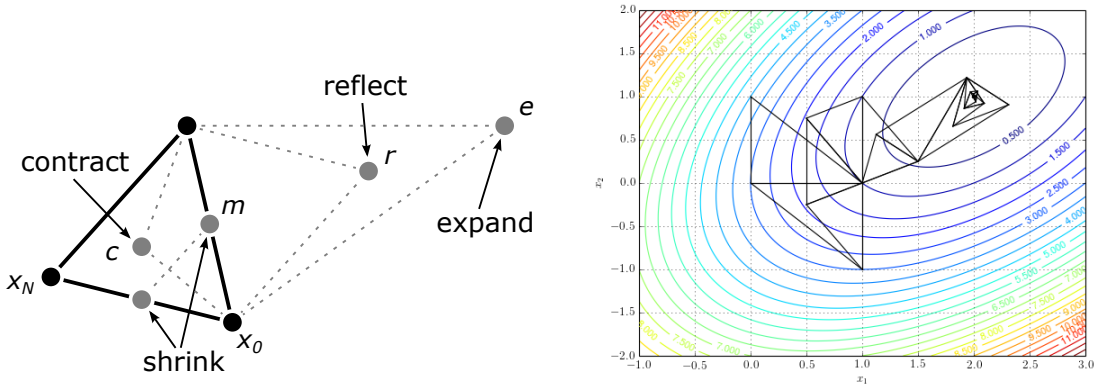

Abb. 5.1: Nelder-Mead: Änderung des Simplex während einer Iteration (links) und Darstellung des Iterationsverlaufs bei der Optimierung einer zweidimensionalen Zielfunktion (rechts) (aus [68] und [79])

drei in Abschnitt 5.3.1 ausgewählten Optimierungsverfahren (Hill Climbing, Simulated Annealing, Nelder-Mead) sucht. Welches der drei Optimierungsverfahren verwendet wird, kann der Benutzer in einer Konfigurationsdatei selbst aussuchen. Ein Vergleich der Verfahren findet sich im Kapitel 5.4.

Die *Frequenzoptimierungsphase* teilt sich auf in eine Offline- und Onlinephase. Beide Phasen unterscheiden sich durch die Auswertung der aus Gleichung (5.2) bekannten Zielfunktion, d.h. durch die Hashrate und den Energieverbrauch bei gegebener/n Frequenz/en. Unabhängig vom Kryptomining kann bei den meisten anderen Anwendungsfällen oder Benchmarks die Offlinephase reichen. Die Onlinephase wurde speziell für die Kryptowährungen hinzugenommen. Es ist jedoch möglich, sowohl die Offline- als auch die Onlinephase zu überspringen, sollte man das Framework für andere Anwendungsfälle verwenden wollen. In der Praxis hat sich bei den Miningalgorithmen gezeigt, dass es sinnvoll ist, beide Phasen zu verwenden, um gute Werte in der Nähe des Optimums schnell zu finden.
In den beiden nachfolgenden Abschnitten werden beide Phasen genauer beschrieben. Danach wird noch eine dritte Optimierungsphase, die Monitoringphase, erläutert. Diese Phase überwacht die getroffenen Einstellungen in periodischen Zeitabschnitten und untersucht, ob diese noch optimal sind. Es könnte ja sein, dass aufgrund von Kursschwankungen eine andere Währung lukrativer ist zu minen, oder sich die Mining-Difficulty negativ auswirkt, sodass es sich aufgrund sinkender Hashrate nicht mehr lohnt, eine Währung zu minen. Die Monitoringphase kann dann die Kryptowährung ändern und eine der in den ersten beiden Phasen getesteten Währungen auswählen. Ein kompletter Ablauf wird in Abbildung 5.2 als Ablaufdiagram mit den einzelnen Phasen schematisch

dargestellt.

Offlinephase

In der Offlinephase wird die Frequenz anhand eines kurzen Offline-Benchmarks optimiert. Offline bedeutet, dass hier keine Verbindung zum Mining-Pool hergestellt wird. Man hat also keine Netzwerkkommunikation, die eventuell die Rohleistung beeinflusst. Die Miningleistung, also die Anzahl der Hashes pro Sekunde, wird so unter idealen Bedingungen bestimmt. Netzwerkunterbrechungen oder -verzögerungen bilden somit keinen Engpass.

Für jede Auswertung der Zielfunktion (Gleichung (5.2))startet der Miner für die zu optimierende Währung im Benchmark-Modus. Die dabei erzielte Hashrate und der maximalen Energieverbrauch, die während des Benchmarks anfallen, werden nach einem Durchlauf in einer Struktur vom Typ `measurement` gespeichert (siehe Listing 5.2). Zusätzlich werden in dieser Struktur die Informationen zu den für die Messungen verwendeten Frequenzen und die Dauer des dazugehörigen Benchmarks abgespeichert. Ein `measurement` entspricht dem Ergebnis einer Auswertung der Zielfunktion.

Die Dauer der Offlinephase ist von der Dauer einer Messung im Benchmark-Modus der entsprechenden Kryptowährung und dem verwendeten Optimierungsverfahren abhängig. Bei den getesteten Währungen beträgt die Messdauer zwischen zwei und 30 Sekunden. Abhängig vom Startpunkt benötigen die Optimierungsverfahren zwischen 10 und 15 Funktionsauswertungen.

Diese Phase kann man überspringen und direkt in der Onlinephase anfangen zu optimieren. Dies hätte jedoch negative Auswirkung auf die Zeit zum Finden des Optimums. Aufgrund der Netzwerklatenzen müsste die Messdauer zwischen zwei Punkten erhöht werden, um verwertbare Hashrates zu bekommen, da die Werte vom Mining-Pool verzögert ankommen. Außerdem hätte man sonst kein allgemein gültiges Modul mit dem man auch andere Benchmarks, deren Performance gerade nicht von einer Onlineanbindung abhängt, energieeffizient Tunen können.

```
struct measurement{
        int mem_clock_, graph_clock_;
        double power_, hashrate_, energy_hash_;
        int nvml_graph_clock_idx_;
        int mem_oc_, graph_oc_;
        int power_measure_dur_ms_, hashrate_measure_dur_ms_;
};
```

Listing 5.2: Struktur zur Abspeicherung einer Messung (measurement) (aus [91])

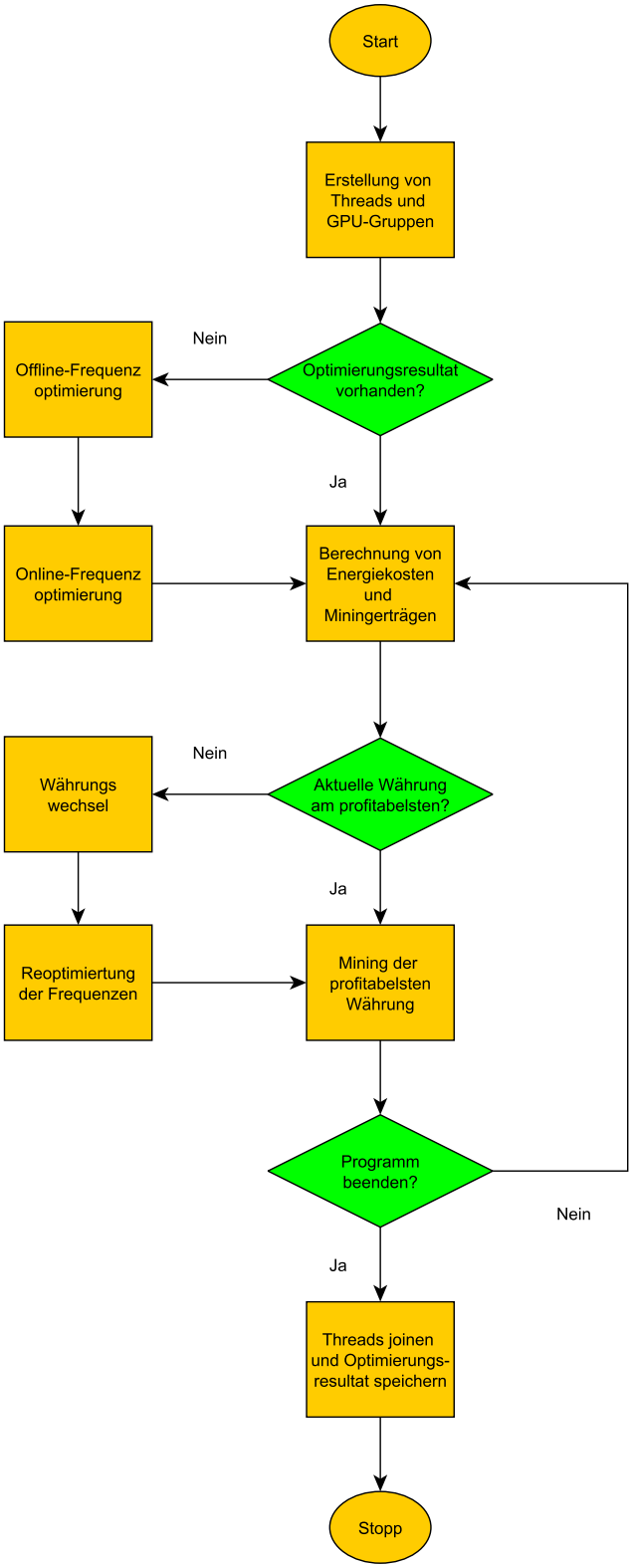

Abb. 5.2: Schematischer Ablauf des Autotunings im implementieren Framework (aus [7])

Onlinephase

In der Onlinephase herrschen nun Realbedingungen. Die Frequenzen werden inklusive Netzwerklatenzen optimiert. Zu Beginn der Onlinephase wird das Minen im Mining-Pool als Hintergrundprozess gestartet. Über den Hintergrundprozess wird die momentan erzielte Hashrate zusammen mit der Systemzeit kontinuierlich in eine Logdatei geschrieben.

Für eine Auswertung der Zielfunktion werden in dieser Phase die gewünschten Frequenzen gesetzt und die aktuelle Systemzeit gespeichert. Dann wird eine Zeit lang, typischerweise zwischen zwei und drei Minuten, gemint. Diese Phase dauert also deutlich länger als die Offlinephase. Danach wird die mittlere erzielte Hashrate während dieser Zeitperiode durch Auslesen der Logdatei bestimmt. Die Leistungsaufnahme während dieser Zeit wird gemittelt. In einer `measurement`-Datei werden dann Hashrate, Energieverbrauch, Frequenzinformation und Messdauer wie in der Offlinephase als Struktur gespeichert.

Als Startpunkt wird in der Onlinephase das Ergebnis der Offlinephase verwendet. Man startet in diese Phase somit schon mit einem energieeffizienten Anfangswert. Somit ist man während der online Optimierungsphase schon sehr energieeffizient. Die Erstellung eines Messpunktes (= Funktionsauswertung) dauert während der Onlinephase deutlich länger als in der Offlinephase. Insgesamt werden sehr viel weniger Funktionsauswertungen benötigt, da man schon sehr nah am Energieoptimum ist und durch die Netzwerklatenz eine Funktionsaufwertung deutlich länger dauert, da die Hashrate nun, falls verfügbar, von der REST-API[2] (Representational State Transfer), des Mining-Pools kommt und diese Kommunikation nicht in Echtzeit stattfindet. Unterstützt der Pool keine REST-API wird die lokale Hashrate des Mining-Programms verwendet. An dieser Stelle sei jedoch zu raten, einen Mining-Pool mit REST-API zu verwenden, da nach dieser Hashrate der Reward ausgezahlt wird.

Zudem werden während der Onlinephase bereits Miningerträge erzielt, da das Minen im Hintergrund läuft. Dies bietet die Offlinephase nicht. Man verwendet in der Offlinephase viel Energie, nur um dem optimalen Set aus Frequenzen sehr nah zu kommen, startet aber dadurch gut in die Onlinephase. Die Realisierung über zwei Phasen ist sinnvoll und bildet einen guten Kompromiss zum Optimieren des Energieverbrauchs.

[2] `https://www.redhat.com/de/topics/api/what-is-a-rest-api`

5.3.3 Monitoring

Nachdem in der Offline- und Onlinephase die Energieeffizienz hinsichtlich Hashingrate optimiert wurde, bezieht das Monitoring die Berechnung der Energiekosten in die Miningerträge (*mining_reward*) mit ein. Somit wird ein zweites Mal ein Optimum gesucht. Auch diese Phase kann ein- oder ausgeschaltet werden. Die Energiekosten pro Sekunde berechnen werden folgendermaßen berechnet (Gleichung 5.5):

$$mining\_costs[\frac{Euro}{Sekunde}] = energy\_consumption\ [W] \cdot \frac{energy\_cost\ [\frac{Euro}{kWh}]}{1000 \cdot 3600}. \quad (5.5)$$

Dies sind die Kosten pro Sekunde, die das Mining Rig verursacht. Es setzt sich aus den Energiekosten, typischerweise in Deutschland in Kilowattstunden und der Leistungsaufnahme des Systems, zusammen.
Für die Berechnung der Miningerträge pro Sekunde wird folgende Formel (5.6) verwendet (aus [73]):

$$mining\_reward = \frac{user\_hashrate}{net\_hashrate} \cdot \frac{1}{block\_time} \cdot block\_reward \cdot stock\_price \quad (5.6)$$

Je höher die eigene Hashrate (*user_hashrate*) im Vergleich zur gesamten Hashrate des Pools (*net_hashrate*) ist, desto höher ist die Belohnung (in Euro), die man bekommt. Diese ergibt sich aus dem Blockreward (*block_reward*) in der entsprechenden Kryptowährung, welche anschließend über den aktuellen Börsenpreis in Euro umgerechnet wird. Die durchschnittliche Zeit, einen Block zu Minen wird in *block_time* angegeben.

Die durchschnittliche Blockzeit ist von der Kryptowährung vorgegeben. Bei Ethereum beträgt sie beispielsweise 15 Sekunden. Die Blockzeit soll unabhängig von den Minern bzw. der Hashrate des Netzwerkes konstant bleiben. Die Blockschwierigkeit wird daher je nach aktueller Hashrate des Netzwerkes dynamisch angepasst. Steigt bzw. sinkt diese, wird auch die Blockschwierigkeit erhöht bzw. gesenkt. Daher gilt in Gleichung 5.7:

$$net\_hashrate = \frac{block\_diff}{block\_time} \quad (5.7)$$

Aus der Differenz zwischen Energiekosten und Miningerträgen ergibt sich dann der Profit (*mining_profit*) in Gleichung (5.8):

$$mining\_profit = mining\_reward - mining\_costs. \quad (5.8)$$

Im Framework wird *stock_price* von *CryptoCompare* (siehe [69]), und *net_hashrate*,

block_time sowie *block_reward* von *WhatToMine* (siehe [96]) über deren REST-APIs abgefragt.

Falls die aktuell geminte Währung nach Neuberechnung von Energiekosten und Miningerträgen nicht mehr die profitabelste Kryptowährung ist, so wird das im Hintergrund laufende Mining der aktuellen Währung beendet und das Mining der neuen profitabelsten Währung gestartet. Es wurden in der Offlinephase alle vom Benutzer ausgewählten Währungen bereits gut optimiert und die Währungen können direkt während dem Monitoring ausgetauscht werden.

Sollte eine Währung ausgetauscht werden, wird für diese neue Währung nochmals nach energieeffizienteren Frequenzen gesucht. Das läuft wie im Online-Teil der Frequenzoptimierungsphase (siehe 5.3.2) ab. Die zuletzt verwendeten Frequenzen bilden den Startpunkt der Suche. Das bisherige Optimierungsresultat wird überschrieben.

5.4 Evaluierung

In diesem Abschnitt werden die Optimierungen von DVFS auf GPUs durch das Autotuning evaluiert. Der Autotuner wird dazu mit verschiedenen GPUs, die in Kapitel 5.2.6 aufgelistet wurden, und verschiedenen Währungen, jedoch hauptsächlich mit den drei Hauptwährungen, die in Kapitel 5.4.1 gezeigt werden, optimiert. Die Ergebnisse von Frequenzoptimierungs- und Monitoringphase werden in den jeweiligen Unterkapiteln dargestellt.

5.4.1 Verwendete Währungen

Als Währungen für die Evaluation kommen hauptsächlich Ethereum (ETH), Monero (XMR) und ZCash (ZEC) zum Einsatz. Die Ergebnisse einige neuer, weniger verbreiteten Währungen (LUX, RVN, BTX und VTC), sogenannte Altcoins, werden am Ende des Kapitels für einen größere Testreihe hergenommen. Für das Minen von ETH wird der *ethminer* in Version 0.15.0.dev11 [88], für XMR der *xmr-stak* in Version 2.4.5 [72], für ZEC der *excavator* in Version 1.1.0a [83] und für die Altcoins der *ccminer* in Version 2.3 [89] genutzt. ETH, XMR und ZEC werden in den nachfolgenden Abschnitten genauer untersucht. Die Messungen wurden unter Windows 10 mit dem NVIDIA-Treiber in Version 391.24 und dem CUDA-Toolkit in Version 9.1 gestartet.

Anfangs wird das Mining der drei Hauptwährungen mit dem *NVIDIA Visual Profiler* (siehe [87]) untersucht. Dies wird gemacht, um vorab einen Einblick zu haben, welche Bottlenecks die einzelnen Währungen haben könnten. Die Performance einer Kryptowährung ist abhängig von der Hashfunktion, die in entsprechender Blockchain

aufgerufen wird. ETH verwendet als Hashfunktion Ethash, XMR Cryptonight und ZEC Equihash. Zum Zeitpunkt der Evaluierung liegt Cryptonight in Version 7 vor. Monero ändert die Hashfunktion alle sechs Monate, damit der Algorithmus nicht auf ASICs (engl. application-specific integrated circuit, dt. anwendungsspezifische integrierte Schaltung) lauffähig ist. Anschließend wird die Hashrate und der Energieverbrauch bei allen auf der jeweiligen Grafikkarte verfügbaren Frequenzen gemessen und ein Vergleich der Energieeffizienz in Hashes pro Joule bei optimalen und Standardfrequenzen gezogen. Für eine detailierte Beschreibung der drei Mining-Algorithmen sei auf die Publikation [5] verwiesen.

5.4.2 Profiling

Vorab wurden die drei Algorithmen auf Speicherbandbreite und Performance untersucht. Abbildung 5.3 zeigt die Ergebnisse des NVIDIA-Profilers der Währungen ETH, ZEC und XMR auf der Titan X (Pascal). Neben der genutzte Speicherbandbreite für Lese- und Schreibzugriffe wurde auch die Anzahl der Instruktionen pro Taktzyklus (engl. instructions per cycle, IPC) gemessen. Die Speicherbandbreite misst, wie stark ein Programm memory-bound ist. Die IPC hingegen messen, wie stark eine Anwendung compute-bound ist. Es wird das nach der GPU-Zeit gewichtete Mittel der einzelnen Kernel darstellt, da die Miner mehrere CUDA-Kernel verwenden.

Grundsätzlich kann man festhalten: Durch die genutzten Speicherbandbreiten ist zu erkennen, dass ETH von den drei Währungen am stärksten speicherbandbreitenlimitiert ist, gefolgt von XMR und ZEC. An den ausgeführten IPC sieht man, dass ZEC am stärksten an die Rechengeschwindigkeit der GPU limitiert ist, gefolgt von ETH und XMR. Je stärker eine Währung memory-bound ist, desto schneller und länger steigt die Hashrate beim Hochstellen der VRAM-Frequenz. Zusätzlich kann die Core-Frequenz gesenkt werden, um Energie zu sparen. Ebenso steigt die Hashrate beim Hochstellen der Core-Frequenz schneller und länger, je stärker eine Währung compute-Bound ist.

5.4.3 Energieoptimum ETH-Ethash

Die Energieoptima (maximale Anzahl an Hashes pro Joule) werden zunächst durch eine erschöpfende Suche (engl. Brute-Force Search, exhaustive search) bestimmt. Es wird also für alle verfügbaren Frequenzen auf der jeweiligen GPU Hashrate und Energieverbrauch gemessen und danach die Messung mit dem Optimum hergenommen.

Die Abbildungen 5.4 und 5.5 zeigen das Ergebnis der Messungen auf den verfügbaren GPUs (siehe Tabelle 5.1) für ETH. Zu sehen ist auf der x-Achse die Core-Frequenz

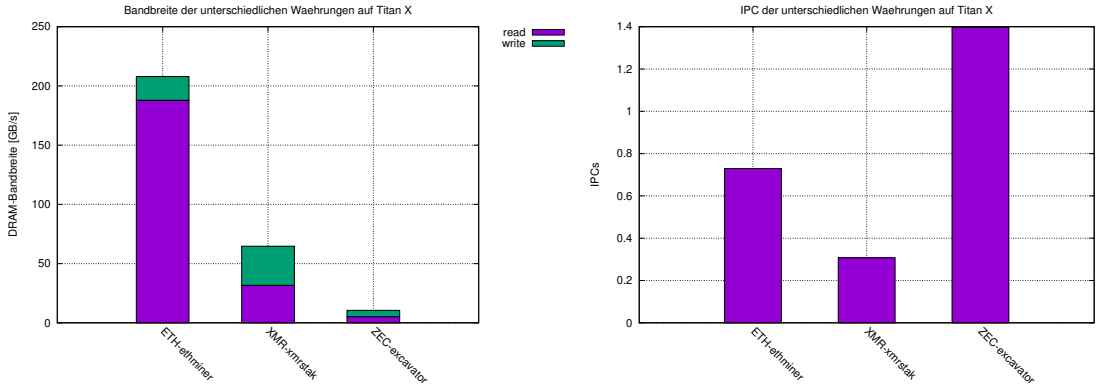

Abb. 5.3: Auslastung der VRAM-Speicherbandbreite (links) und ausgeführte IPC (rechts) beim Mining von Ethereum (ETH, ethminer), Monero, (XMR, xmrstak) und ZCash (ZEC, excavator) auf einer Titan X (Pascal) mit Standardfrequenzen (aus [5]).

und auf der y-Achse die VRAM-Frequenz. Die z-Achsen unterscheiden sich. Die beiden oberen Diagramme zeigen die Hashrate pro Sekunde (in [H/s]), die mittlere Zeile zeigt die Leistungsaufnahme (in [J/s]) und die letzte Zeile zeigt die Energieeffizient in Performance pro Zeit (in [Hashes/Joule]). In der linken Spalte in Abbildung 5.4 sind die Werte zur NVIDIA Titan X (Pascal) abgebildet und in der rechte Spalte die Titan V mit ihrem schnellen HBM2 Speicher. In Abbildung 5.5 sind links die NVIDIA GTX 1080 und rechts die Quadro P400 abgebildet. Hohe Werte gehen farblich in Rot über, wohingegen niedrige Werte ins Blau übergehen. Außerdem sind die Standardfrequenzen (als Kreis) und die optimalen Frequenzen (als x) eingezeichnet.

Zur Ermittelung der Effizienzsteigerung werden die Hashes pro Energie bei optimalen Frequenzen mit denen bei den Standardfrequenzen in Tabelle 5.2 verglichen. Die Frequenzen, die standardmäßig von der GPU verwendet werden, werden ermittelt, indem der Miner auf der jeweiligen Grafikkarte gestartet wird und die vom NVIDIA-Treiber eingestellten Frequenzen ausgelesen werden.

In den Diagrammen wird deutlich, dass ETH im Vergleich zu den anderen Währungen am meisten von höheren VRAM-Frequenzen profitiert, was das Ergebnis des Profiling aus Abschnitt 5.4.2 bestätigt. Das Energieoptimum ist zumeist bei mittleren Core- und mittleren bis hohen VRAM-Frequenzen zu finden. Durch die niedrigen IPC lohnt es sich nicht, die Core-Frequenz hochzutakten. Diese Energie kann man sich hier sparen. Auch der sehr schnelle HBM2-Speicher der Titan V wirkt sich positiv auf die Performance aus. Dadurch ist jedoch nur eine Effizienzsteigerung von etwas über 16% zu erreichen, weil die Grafikkarte schon standardmäßig gute Werte liefert. Bei der Titan X ist eine Steigerung der Effizienz von über 40% zu sehen. Auch die Quadro P4000 steigerte die Effizienz um über 35%. Einzig die GTX 1080 kann nur eine Steigerung der Effizienz von etwas über

3% erreichen. Dies liegt an dem sehr beschränkten Frequenzbereich, den man einstellen kann. Bei kleineren Frequenzänderungen ist es zudem schwer, die Leistungsaufnahme der GTX 1080 zu messen. Deswegen kommt es bei der 1080 sehr häufig zu Zackenbildungen. Generell kann man von dieser Grafikkarte für solche Versuche abraten und stattdessen sollte man auf eine andere der drei Grafikkarten ausweichen.

Tab. 5.2: ETH: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7])

Währung: ETH		**TITAN X**	**TITAN V**	**GTX 1080**	**Quadro P4000**
Optimale Frequenz	VRAM-Frequenz	4955	1010	5805	3802
	Core-Frequenz	1151	1035	1711	1088
	Hashrate	31119928	82603192	25339050	26496646
	Leistungsaufnahme	113.095	155.207	156.29	77.036
	Hashes/Joule	275166	532213	162128	343951
Standard Frequenz	VRAM-Frequenz	5005	850	5005	3802
	Core-Frequenz	1822	1335	1885	1695
	Hashrate	31532634	67375607	21387776	26630501
	Leistungsaufnahme	161.008	147.519	135.96	104.775
	Hashes/Joule	195845	456725	157309	254168
Effizienzsteigerung		40.5%	16.53%	3.06%	35.32%

5.4.4 Energieoptimum XMR-Cryptonight

XMR hat ein ähnliches Verhalten wie ETH, wie die Abbildungen 5.6 und 5.7 zeigen. Die Achsen entsprechen denen bei der ETH Darstellung. Da XMR aber sowohl weniger compute-, als auch weniger memory-bound wie ETH ist (siehe Abbildung 5.3), fällt der Anstieg der Hashrate bei Erhöhung der Frequenzen etwas flacher aus. Auffallend ist auch, dass XMR relativ wenig Energie benötigt. In Tabelle 5.3 sieht man die Steigerung der Effizienz durch das Optimieren der Frequenzen. Bei der Titan X kann die Effizienz über mehr als 84% gesteigert werden. Hier führt die Reduzierung der Core-Frequenz von 1847 MHz auf nur 1202 MHz zu einer Halbierung der Leistungsaufnahme. Dabei bleibt die Hashrate nahezu gleich. Die Titan V steigert die Effizienz um über 31%, die GTX 1080 sogar um über 63% und die P4000 um knapp 50%. Die GPU NVIDIA GTX 1080 ist hier sehr schwer zu messen. Der geringe Bereich von etwa 600 MHz auf der Achse der VRAM-Frequenz, der eingestellt werden kann, führt zu starken Schwankungen bei den Messungen der Leistungsaufnahme was zu unsauberen Zacken wie im Abbildung 5.9 (Mitte, links) führt. Dies hat auch einen Effekt auf die Effizienz in dem Plot direkt darunter.

Mining von XMR lohnt sich im Gegensatz zu den meisten anderen Währungen auch

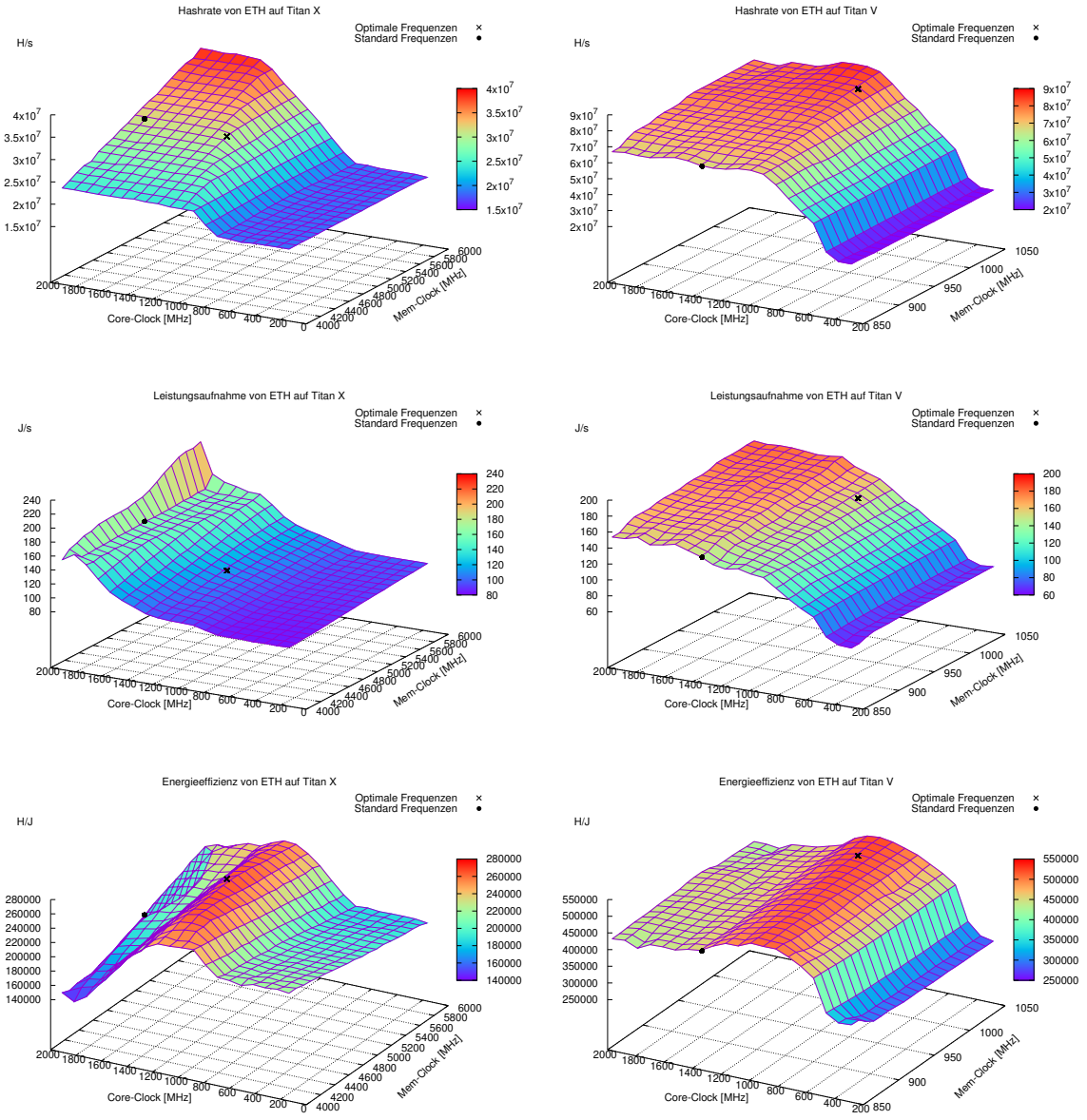

Abb. 5.4: Ethereum (ETH): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte)

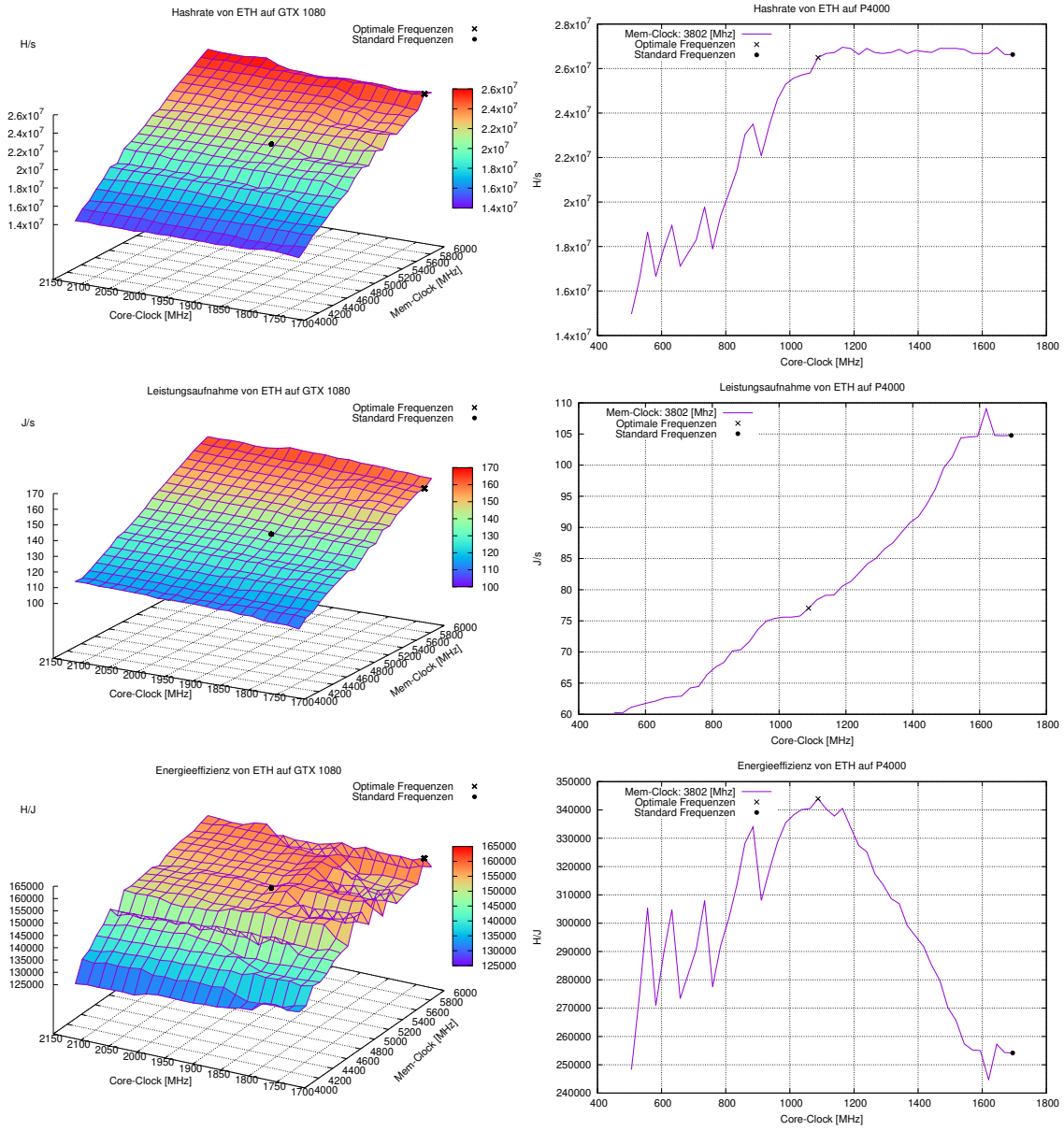

Abb. 5.5: Ethereum (ETH): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte)

auf CPUs. Ein plausibler Grund dafür ist, dass es die höhere Anzahl an Recheneinheiten einer GPU kaum nutzen kann, was der niedrige IPC-Wert verrät. Dies ist auch der Grund für den niedrigen Energiebedarf.

Tab. 5.3: XMR: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7])

Währung: XMR		**TITAN X**	**TITAN V**	**GTX 1080**	**Quadro P4000**
Standard Frequenz	VRAM-Frequenz	5155	1010	5502	3802
	Core-Frequenz	1202	1035	1811	999
	Hashrate	744.5	1277.6	581.0	554.1
	Leistungsaufnahme	86.42	88.115	85.51	45.589
	Hashes/Joule	8.6149	14.4992	6.79453	12.1542
Normale Frequenz	VRAM-Frequenz	5005	850	5005	3802
	Core-Frequenz	1847	1335	1911	1708
	Hashrate	753.5	1178.2	523.8	593.9
	Leistungsaufnahme	161.02	106.495	126.34	73.179
	Hashes/Joule	4.67954	11.0634	4.14596	8.11572
Effizienzsteigerung		84.1%	31.06%	63.88%	49.76%

5.4.5 Energieoptimum ZEC-Equihash

Anders als ETH und XMR ist ZEC im Gegensatz eher compute-bound, wie in Abbildung 5.3 zu sehen ist. Daher findet sich das Energieoptimum meistens bei niedrigen VRAM- und etwas erhöhten Core-Frequenzen (siehe Abbildungen 5.8 und 5.9). Auch hier sind die Achsen entsprechend denen der bei der ETH Darstellung. Deshalb ist das Minen von ZEC auf der Titan V auch ineffizienter, da der schnellen HBM2-Speicher keinen Vorteil mit sich bringt. ZEC hat aufgrund des hohen IPC-Wertes einen relativ hohen Energiebedarf.

Die Energieeffizienz konnte hier auf der Quadro P4000 gesteigert werden. Verglichen mit ETH und XMR profiert die P4000 GPU auch am meisten von Frequenzanpassungen bei ZEC, da hier hauptsächlich die Core-Frequenz Einfluss auf den Energieverbrauch hat und man bei der P4000 GPU auch nur diese Frequenz ändern kann. Die fest eingestelle VRAM-Frequenz spielt hier kaum eine Rolle. Die Titan X steigert sich bei ZEC um über 22% und die GTX 1080 um über 32%. Die Titan V GPU profitiert hier aus genannten Gründen kaum und kann die Effizienz nur um etwas über 6% steigern.

Tabelle 5.4 zeigt zusammenfassend die Effizienzsteigerung für ZEC bei Verwendung der optimalen Frequenzen.

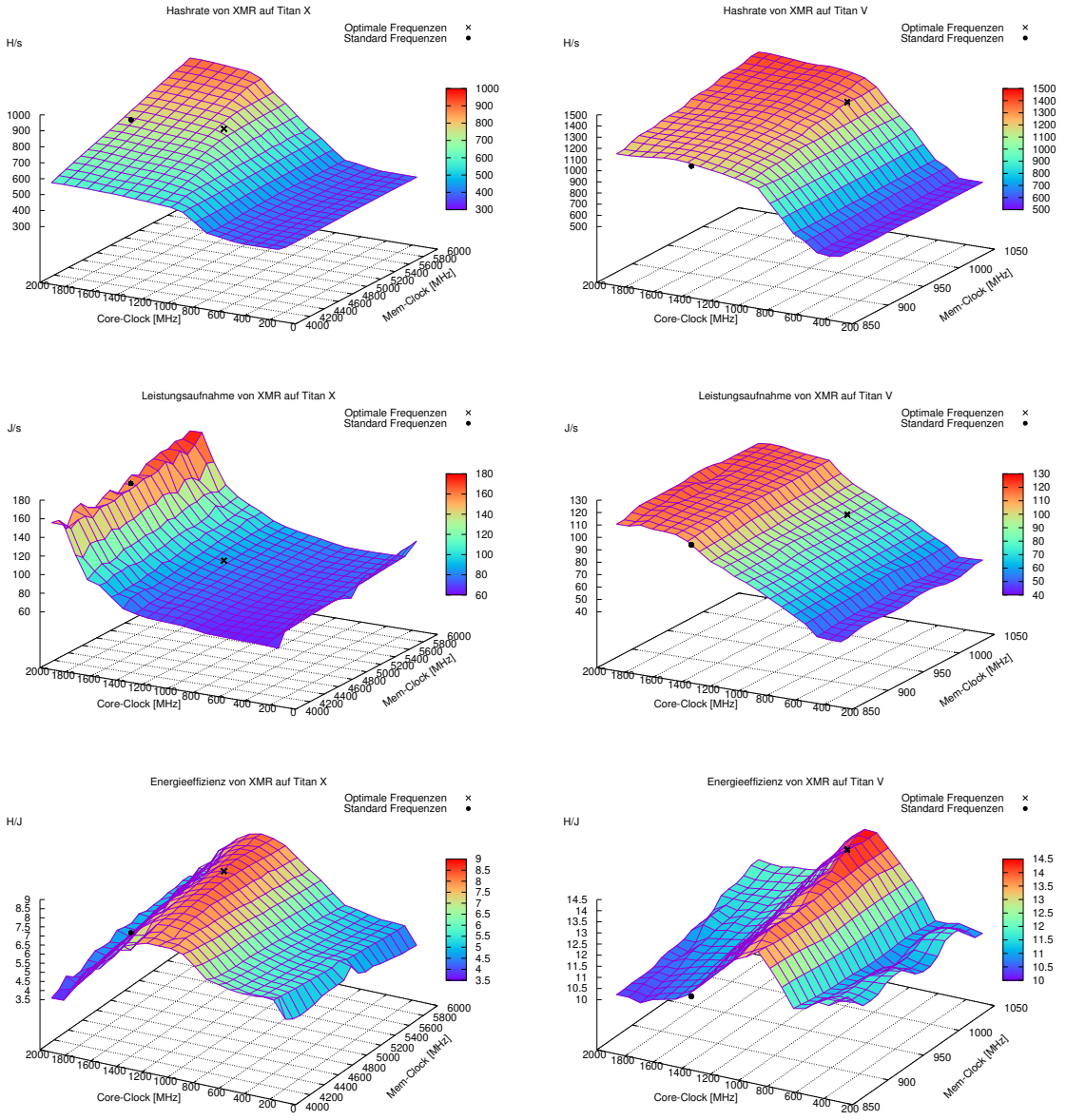

Abb. 5.6: Monero (XMR): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte) (aus [7])

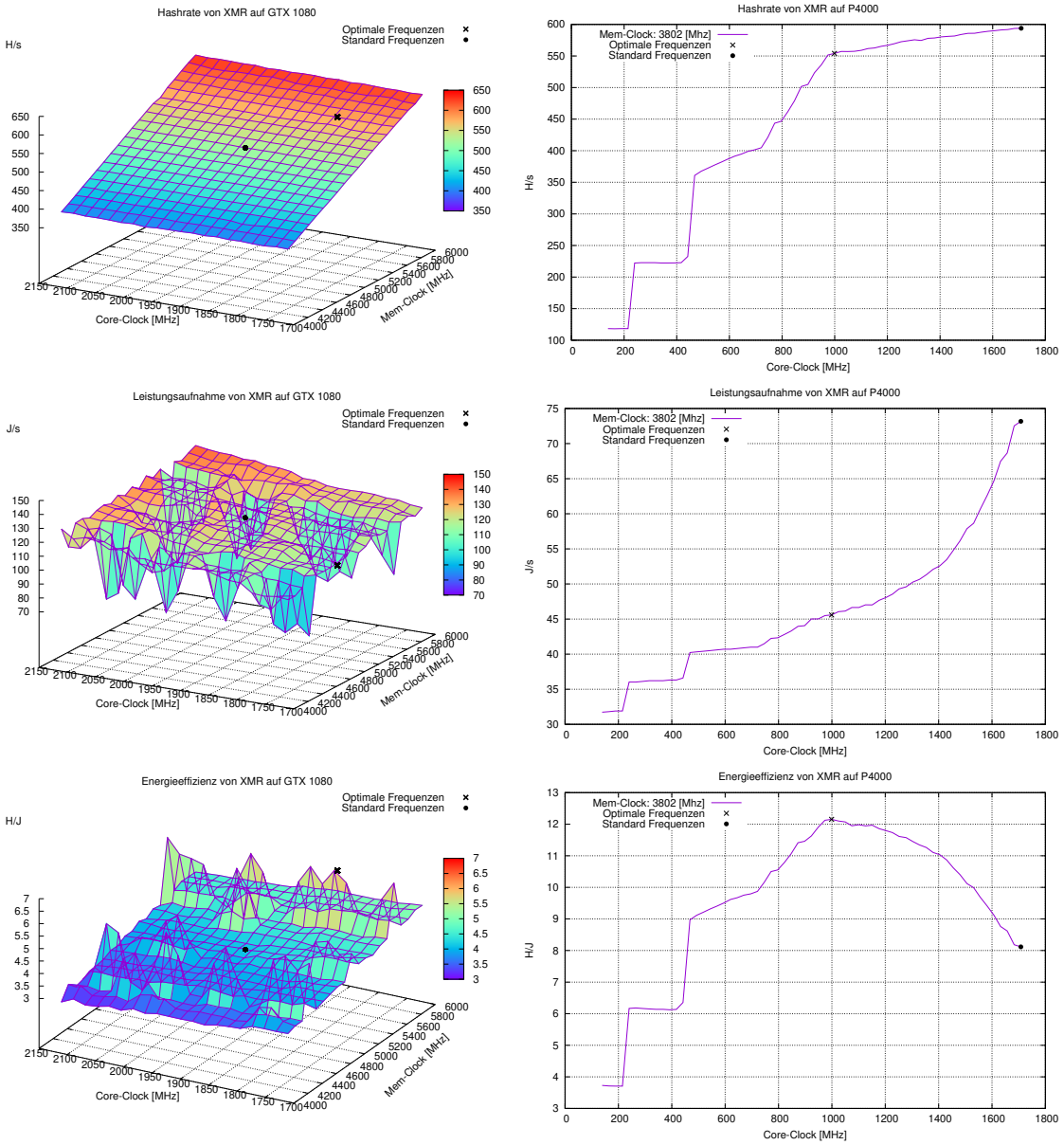

Abb. 5.7: Monero (XMR): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7])

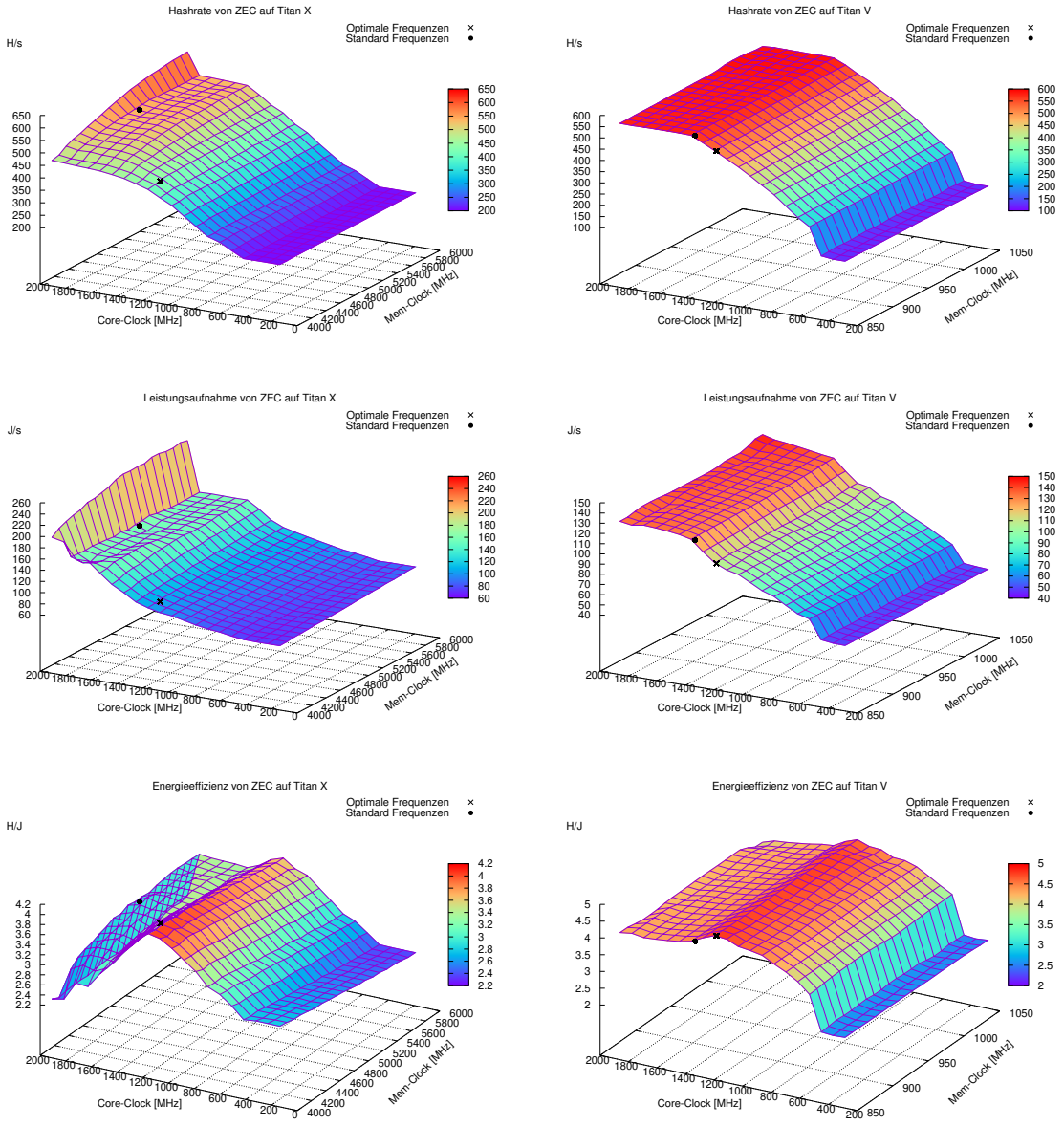

Abb. 5.8: ZCash (ZEC): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Titan X (Pascal) (linke Spalte) und der Titan V (rechte Spalte) (aus [7])

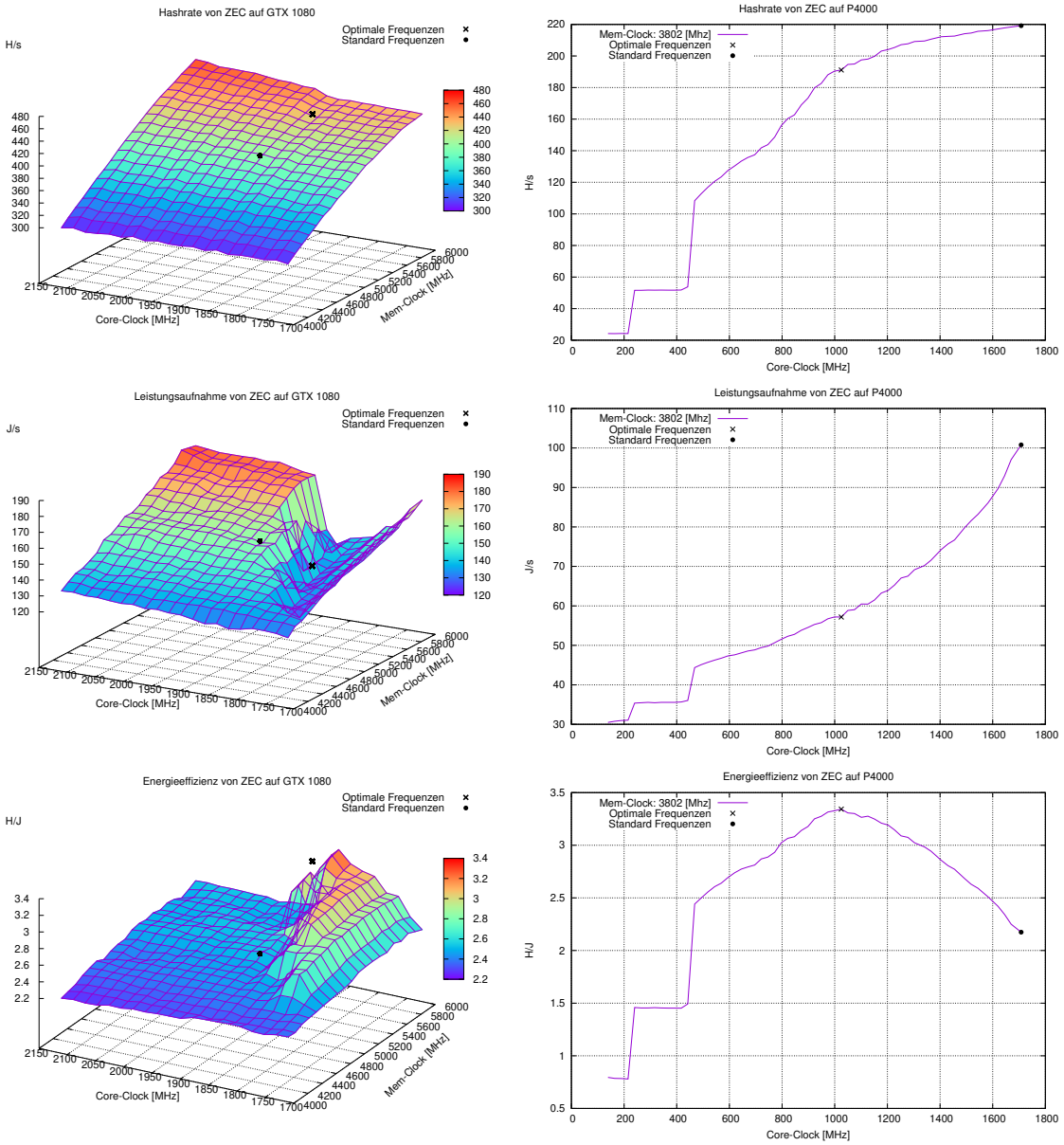

Abb. 5.9: ZCash (ZEC): Hashrate (oben), Energieverbrauch (mitte) und Energieeffizienz (unten) bei allen verfügbaren Frequenzen auf der Geforce GTX 1080 (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7])

Tab. 5.4: ZEC: Optimale vs. Standardfrequenzen auf den verschiedenen Grafikkarten (aus [7])

Währung: ZEC		TITAN X	TITAN V	GTX 1080	Quadro P4000
Optimale Frequenz	VRAM-Frequenz	4155	850	5555	3802
	Core-Frequenz	1151	1185	1861	1025
	Hashrate	433.525144	525.599705	432.495997	191.152922
	Leistungsaufnahme	104.826	109.25	129.22	57.176
	Hashes/Joule	4.13566	4.81098	3.34697	3.34324
Standard Frequenz	VRAM-Frequenz	5005	850	5005	3802
	Core-Frequenz	1784	1335	1885	1708
	Hashrate	538.831065	579.105294	395.216173	219.175340
	Leistungsaufnahme	159.64	128.46	156.30	100.79
	Hashes/Joule	3.37529	4.50806	2.52857	2.17457
Effizienzsteigerung		22.53%	6.72%	32.37%	53.74%

5.4.6 Verwendung eines Power-Limits

Mit der *NVML*-Bibliothek ist es auch möglich, die maximale Leistungsaufnahme (engl. Power) von Grafikkarten zu limitieren. Damit können die Grafikkarten ab diesem Limit nicht mehr Strom ziehen, als vorgegeben. Die Funktion `nvmlDeviceSetPowerManagementLimit` stellt ein Limit ein, das die Grafikkarte nicht überschreiten darf. Jetzt ist es möglich die Energie zu senken, indem man ein Power-Limit einstellt. Das klappt jedoch nur teilweise. Mit der Beschränkung sorgt der NVIDIA-Treiber dafür, dass nur die Core-Frequenz reduziert wird. Die VRAM-Frequenz wird nicht verändert. Im Umkehrschluss folgt, dass eine manuelle Erhöhung der VRAM-Frequenz die Core-Frequenz bei eigeschaltetem Power-Limit immer weiter senkt. Bei Algorithmen, die durch die Speicherbandbreite limitiert sind, wie es bei Ethereum der Fall ist, erreicht man über diesen Weg auch eine recht gute Effizienz, wie Tabelle 5.5 zeigt. Andere Algorithmen, die stark von der Rechenintensität abhängig sind, wie es bei ZEC der Fall ist, brechen stark in der Performance ein. Hier führt eine Erhöhung der VRAM-Frequenz nur zu einer Erhöhung des Energieverbrauch. Es tritt also der gegenteiliger Effekt ein.

Der Vorteil liegt demnach nur bei memory-bound Algorithmen, da man keine Optimierungsalgorithmen benötigt. Auf der anderen Seite muss man trotzdem einen guten Wert für das Power-Limit finden. Eine zu starke Limitierung der Core-Frequenz kann sich auch hier auf die Performance auswirken.

Tabelle 5.5 zeigt die erzielte Hashrate unter der Verwendung eines Power-Limits von 125W auf Titan X und 60W auf der Quadro P4000 für die Währungen ETH, XMR und ZEC. Neben der erzielten Hashrate ist auch die vom Treiber eingestell-

te Core-Frequenz und der tatsächliche Leistungsaufnahme im Format `|Core-Clock| Leistungsaufnahme|Hashrate|` zu sehen. Auf der Titan X wurden zusätzlich verschiedene VRAM-Frequenzen untersucht. Mit Ausnahme von ZEC wird die höchste Hashrate bei der maximalen VRAM-Frequenz erzielt. Selbst bei ZEC ist, obwohl dieses relativ compute-bound ist, die Abweichung nicht groß.

	VRAM-Frequenz	**ETH**	**XMR**	**ZEC**
Titan X (125W)	4005	1683|125|24230325	1847|125|591.2	1506|125|450.6
	5005	1455|125|31828634	1809|125|767.4	1303|125|454.8
	6005	1265|125|34486963	1721|125|934.2	1151|125|425.7
P4000 (60W)	3802	620|60|17187522	1708|60|586.1	1177|60|202.9

Tab. 5.5: Gesetzte Core-Frequenz, tatsächlich gemessene Leistungsaufnahme und erzielte Hashrate auf der Titan X bei einem Power-Limit von 125W und auf der Quadro P4000 bei einem Power-Limit von 60W für verschiedene Währungen (aus [7])

5.4.7 Frequenzoptimierung durch Suchstrategien

Nachdem in den Kapiteln 5.4.3 bis 5.4.5 die erschöpfende Suche dargestellt wurde, bei der die Optima der drei Hauptwährungen liegen, konzentriert sich dieser Abschnitt auf die verschiedenen Optimierungsalgorithmen (siehe Abschnitt 5.3.1), welche während der *Frequenzoptimierungsphase* (siehe Abschnitt 5.3.2) verwendet werden. Hierfür wird eine 2D-Optimierung auf der Titan X und eine 1D-Optimierung auf der Quadro P4000 betrachtet. ZEC wird als Kryptowährung verwendet.

Jeder der drei Such-Algorithmen (Hill Climbing, Simulated Annealing und Nelder-Mead) wird dreimal mit verschiedenen Startfrequenzen ausgeführt. Die verschiedenen Startpunkte liegen bei maximalen, minimalen und mittleren Core- und VRAM-Frequenzen. Außerdem werden in Kapitel 5.4.11 die Algorithmen mit einer Nebenbedingung ausgeführt. Die Nebenbedingung setze eine Hashrate voraus, welche nicht unterschritten werden darf. Als maximale Anzahl an Iterationen wird für alle Ausführungen 6 verwendet.

In den drei folgenden Abschnitten werden der Optimierungsverlauf und die Performance der einzelnen Suchalgorithmen vorgestellt. Die Performance der Optimierung wird mit folgenden Kriterien gemessen:

- Abweichung zwischen realem Optimum und gefundener Lösung.
- Benötigte Anzahl an Funktionsauswertungen zum Finden des Optimums.

5.4.8 Performance Hill Climbing

In Tabelle 5.6 ist die Optimierung mit Hill Climbing für die Währung ZEC für die beiden Grafikkarten Titan X und Quadro P4000 zu finden. Die erste Tabellenzeile zeigt das gesuchte Optimum (Hashes pro Joule) aus der erschöpfenden Suche aus Tabelle 5.4. In Klammern sind jeweils die Core- und VRAM-Frequenz angeben. Bei der P4000 ist die VRAM-Frequenz immer gleich, da man sie nicht manipulieren kann. Im zweiten Tabellenzeilenblock wird die gefundene Energieeffizienz in Hashes pro Joule angegeben und mit den dazugehörigen Frequenzen in Klammern für die verschiedenen Startpunkte gezeigt. Beim Startpunkt $Start - Min$ beispielsweise sind sowohl VRAM- als auch Core-Frequenz auf das Minimum der entsprechenden Grafikkarte gesetzt. Die benötigte Anzahl an Funktionsauswertungen zum Finden der besten Energieeffizienz ist in dem dritten Tabellenzeilenblock gegeben. Ebenson findet sich dort die Gesamtzahl der Funktionsauswertungen bis zur Terminierung, welches in Klammern angegeben ist. In der linken Spalte sind die Werte für die Titan X (2D-Optimierung) und in der rechten Spalte sind die Werte für die Quadro P4000 (1D-Optimierung) angegeben.

Abbildung 5.10 zeigt die dazugehörigen Optimierungsverläufe. Es ist jeweils der Weg, den der Algorithmus zurücklegt, zusammen mit der zu optimierenden Funktion bei den verschiedenen Startpunkten für beide GPUs dargestellt. Die erste Zeile zeigt den Start bei minimalen, die mittlere Zeile bei mittleren Frequenzen und die letzte Zeile bei maximalen Frequenzen. Die schwarze Linie zeigt den Weg des Hill Climbing vom Start bis zum gefunden Optimum mit den Zwischenstationen (Frequenzen).

Unabhängig vom Startpunkt findet das Hill Climbing einen guten Wert nahe des energieeffizientesten Punktes auf beiden Grafikkarten. Im Schnitt wird ein Wert von 3.98 H/J auf der Titan X gefunden. Das Optimum aus Tabelle 5.4 liegt bei 4.13 H/J. Auf der Quadro P4000 liegt der durchschnittlich gefundene Wert von 3.31 H/J fast beim Optimum aus Tabelle 5.4 (3.34 H/J). Kleine Unterschiede wie diese sind aber eher im Bereich der Messungenauigkeiten zu verbuchen, da es sich bei der Energie um gemittelte Werte über eine kurze Zeitperiode handelt, die schwanken. Bei der 1D-Optimierung auf der Quadro P4000 wird das Optimum leichter gefunden. Die zum Finden des besten Wertes durchschnittlich benötigte Anzahl an Funktionsauswertungen beträgt auf der Titan X 15 und auf der Quadro P4000 fünf.

5.4.9 Performance Simulated Annealing

Auf beiden GPUs ist die Performance von Simulated Annealing ähnlich der von Hill Climbing. Dies liegt daran, dass die Funktion keine lokalen Optima besitzt, die das

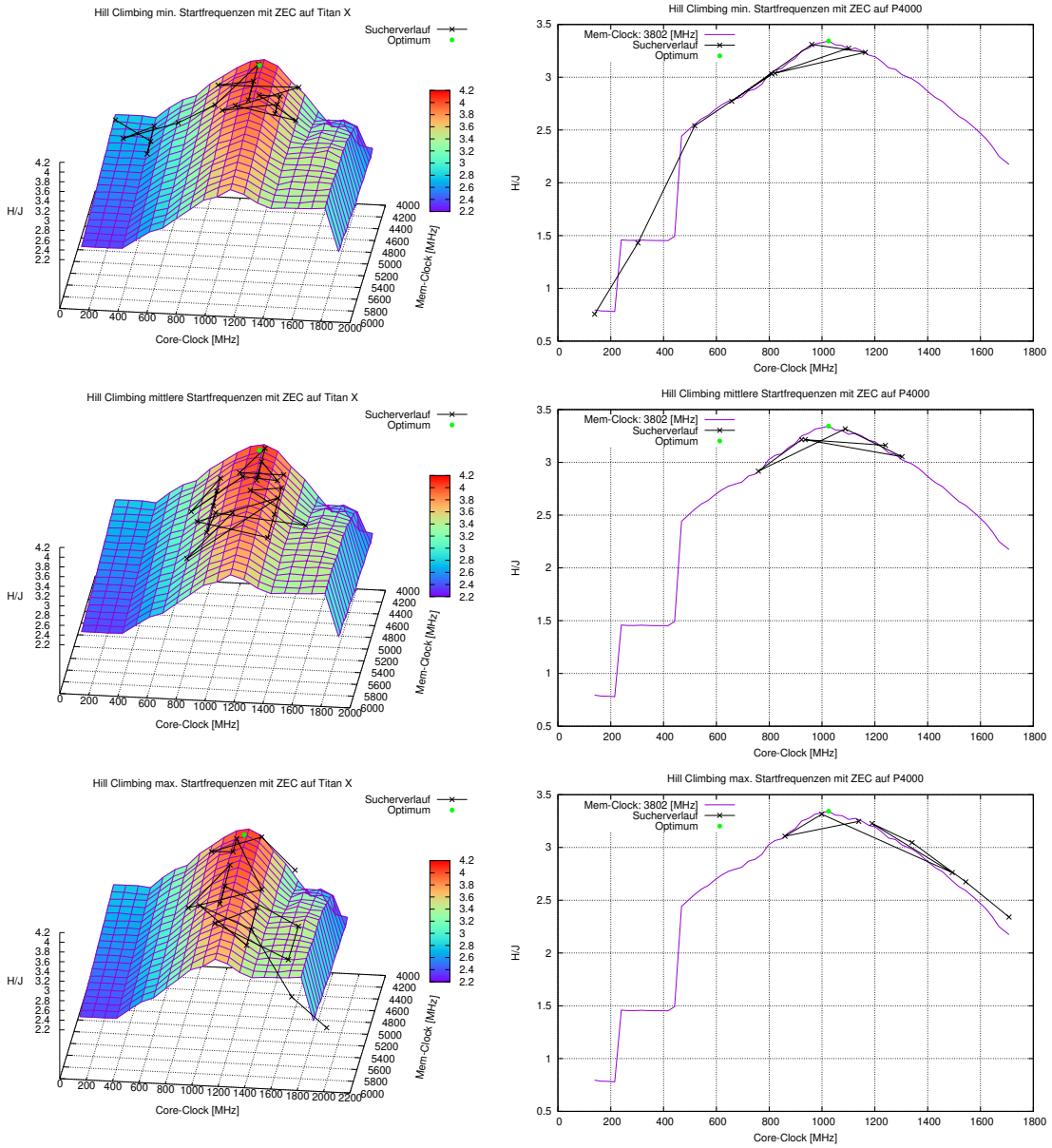

Abb. 5.10: Hill Climbing: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7])

Tab. 5.6: Hill Climbing: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7])

Währung: ZEC		**TITAN X**	**Quadro P4000**
Optimum		4.13566 (4155,1151)	3.34324 (3802,1025)
Ergebnis	Start-Min	3.97659 (4006,1126)	3.31102 (3802,961)
	Start-Mid	4.02499 (4248,1177)	3.31719 (3802,1088)
	Start-Max	3.94662 (4024,1265)	3.31502 (3802,999)
	Durchschnitt	3.98273 (4093,1189)	3.31438 (3802,1016)
Funktion auswer-tungen	Start-Min	8 (19)	6 (9)
	Start-Mid	18 (23)	3 (6)
	Start-Max	18 (19)	6 (8)
	Durchschnitt	14.66 (20.33)	5 (7.66)

Simulated Annealing verlassen muss oder in denen das Hill Climbing feststecken kann. In Tabelle 5.7 finden sich die Ergebnisse und die benötigte Anzahl an Funktionsauswertungen der Optimierung. Der Tabellenaufbau ist wie in Abschnitt 5.4.8 beschrieben. Die dazugehörigen Verläufe der Suchstrategie bei den verschiedenen Startpunkten sind in Abbildung 5.11 dargestellt. Auch hier sind die Startpunkte wieder von minimalen Startfrequenzen zu maximalen Startfrequenzen von oben nach unten sortiert.

Die Titan X erreicht durchschnittlich einen Wert von 4 H/J. Das entspricht gerade einmal einer Abweichung von 0.13 H/J vom Optimum. Auf der Quadro P4000 wird, abgesehen von Messungenauigkeiten, wie beim Hill Climbing auch der Bestwert gefunden. Die durchschnittlich benötigte Anzahl an Funktionsauswertungen ist beim Hill Climbing mit 15 auf der Titan X und fünf auf der Quadro P4000 gleich. Das Simulated Annealing verwendet für das Finden neuer Lösungskandidaten zum Teil das Hill Climbing. (siehe Algorithmus 1).

5.4.10 Performance Nelder-Mead

Etwas schlechter als Hill Climbing und Simulated Annealing schneidet die Optimierung mit dem Nelder-Mead Verfahren ab. In Tabelle 5.8 sind die gefundenen Werte sowie die benötigte Anzahl an Funktionsauswertungen gezeigt. Der Aufbau der Tabelle ist identisch wie in den beiden letzten Abschnitten (Kapitel 5.4.8 und 5.4.9). Abbildung 5.12 zeigt die dazugehörigen Verläufe der Suchstrategie bei verschiedenen Startfrequenzen.

Auf der Titan X beträgt die durchschnittlich gefundene Energieeffizienz 3.88 H/J, was einer Abweichung von 0.25 H/J vom Optimum aus der erschöpfenden Suche entspricht. Beim Start mit maximalen Core- und VRAM-Frequenzen wird der VRAM-Frequenzbereich nicht ausreichend abgesucht und deshalb wird hier nur ein Wert von

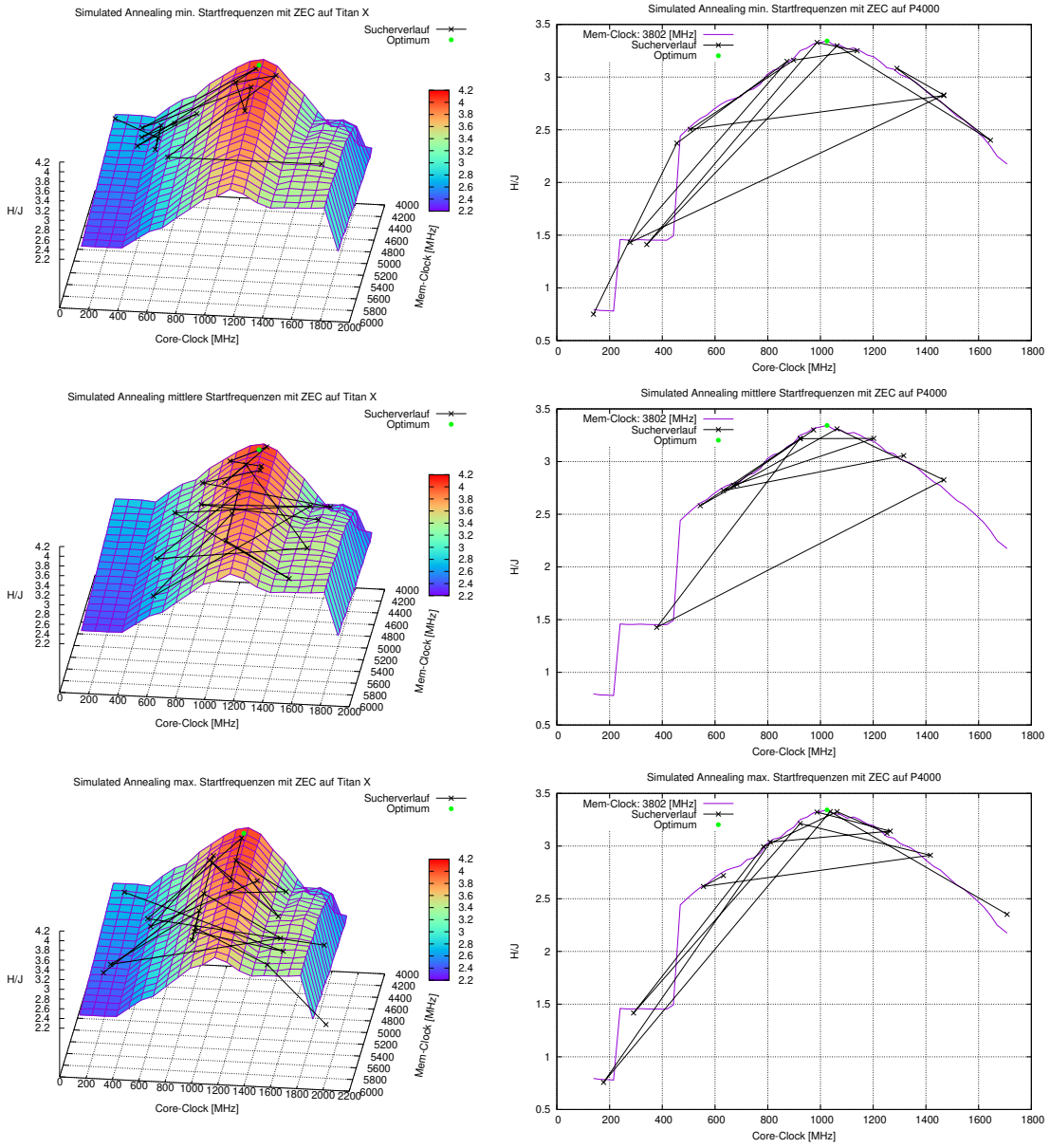

Abb. 5.11: Simulated Annealing: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7])

Tab. 5.7: Simulated Annealing: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7])

Währung: ZEC		**TITAN X**	**Quadro P4000**
Optimum		4.13566 (4155,1151)	3.34324 (3802,1025)
Ergebnis	Start-Min	4.02485 (4127,1126)	3.32891 (3802,987)
	Start-Mid	4.05375 (4036,1189)	3.31112 (3802,1063)
	Start-Max	3.9611 (4091,1126)	3.32744 (3802,1037)
	Durchschnitt	4.013 (4085,1147)	3.32249 (3802,1029)
Funktion auswer-tungen	Start-Min	9 (16)	10 (14)
	Start-Mid	12 (19)	4 (13)
	Start-Max	24 (26)	2 (15)
	Durchschnitt	15 (20.33)	5.33 (14)

3.72 H/J gefunden. Auf der Quadro P4000 wird wie beim Hill Climbing und Simulated Annealing abgesehen von Messungenauigkeiten der Optimalwert gefunden. Zum Finden des Optimums benötigt das Nelder-Mead Verfahren auf beiden Grafikkarten (Titan X und Quadro P4000) durchschnittlich zehn Funktionsauswertungen. Das Nelder-Mead Verfahren ist also auf der Titan X schneller (10 gegen 15 Auswertungen), dafür aber auf der Quadro P4000 langsamer (10 gegen 5 Auswertungen) als Simulated Annealing und Hill Climbing.

Das Nelder-Mead Verfahren ist allgemein beim Finden neuer Lösungskandidaten relativ unabhängig von der Dimension. Zwar ändert sich beim Simplex je nach Dimension die Anzahl an Punkten, aber in jeder Iteration muss, unabhängig von der Dimension, der schlechteste Punkt des Simplex ersetzt oder der Simplex komprimiert werden. Nur bei der Berechnung des initialen Simplex und bei einer Komprimierung wird pro Dimension höher eine Funktionsauswertung mehr benötigt.

5.4.11 Optimierung unter Nebenbedingungen

Bisher gab es für das Optimum, also den Punkt, bei denen die Anzahl des Hashes pro Joule maximal ist, keine Einschränkungen. Theoretisch wäre es möglich gewesen, dass die Frequenzen sehr klein gewählt werden konnten. Dies würde die Leistungsaufnahme senken, jedoch würde auch die Performance stark darunter leiden. Abhilfe hierfür schafft die Optimierung mit Nebenbedienungen. Als Nebenbedingung führt man eine minimale Hashrate ein, die nicht unterschritten werden darf. Die Hashrate wird in Prozent, abhängig von der maximalen Hashrate angeben. Die Evaluation der Optimierung unter Nebenbedingungen der drei verschiedenen Such-Algorithmen erfolgt mit einer minimal einzuhaltenden Hashrate von 95% der maximalen Hashrate auf der Titan X und Quadro

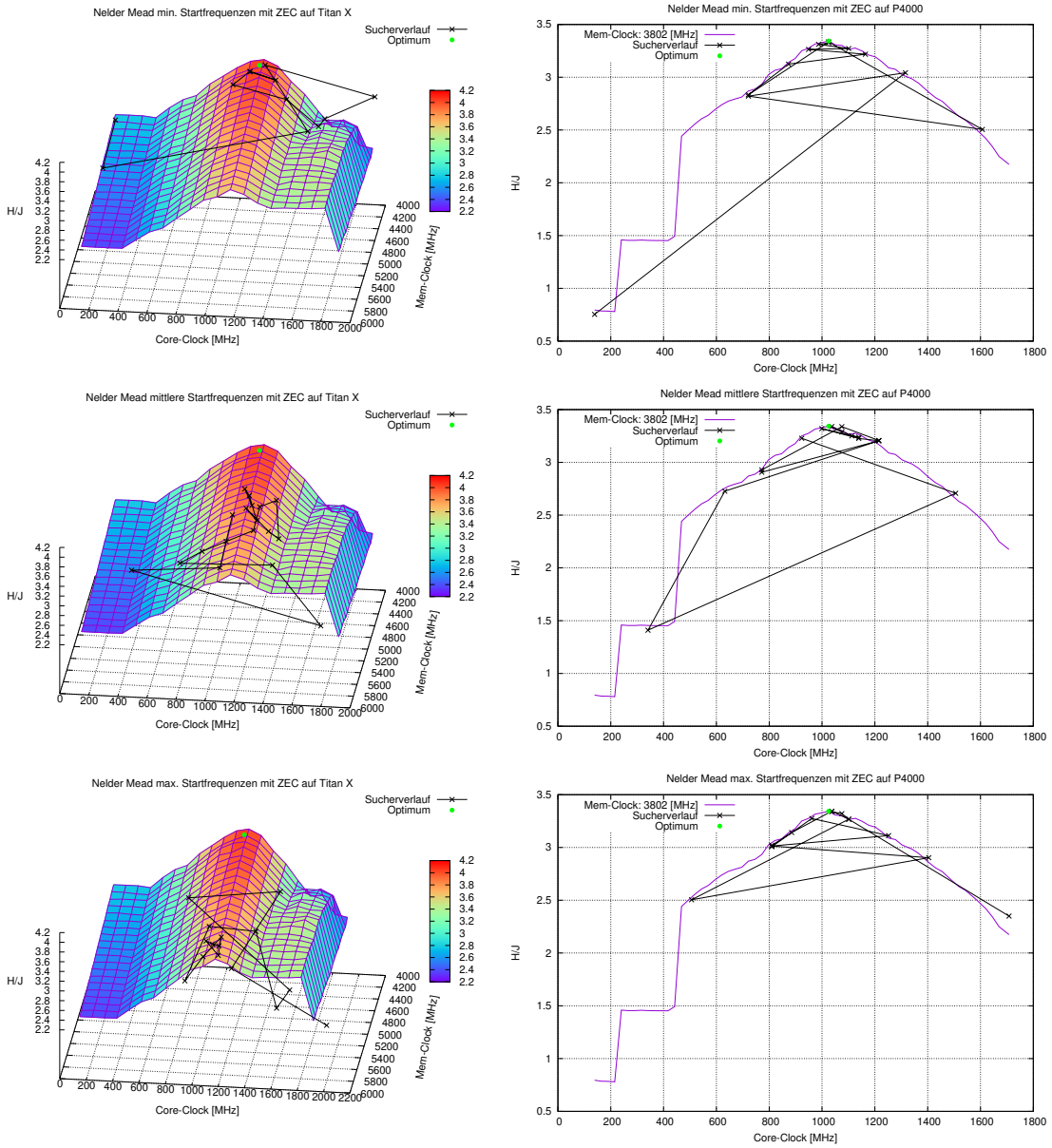

Abb. 5.12: Nelder-Mead: Verlauf des Suchalgorithmus von ZEC mit Startpunkt bei minimalen (oben), mittleren (mitte) und maximalen (unten) Frequenzen auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte) (aus [7])

Tab. 5.8: Nelder-Mead: Ergebnis der Suche in Hashes pro Joule inkl. dazugehörigen Frequenzen und Anzahl an Funktionsauswertungen der Optimierung von ZEC mit Start bei minimalen, mittleren und maximalen Frequenzen (aus [7])

Währung: ZEC		**TITAN X**	**Quadro P4000**
Optimum		4.13566 (4155,1151)	3.34324 (3802,1025)
Ergebnis	Start-Min	4.01787 (4050,1177)	3.33445 (3802,1025)
	Start-Mid	3.90598 (4732,1151)	3.33988 (3802,1037)
	Start-Max	3.72097 (5549,1164)	3.34415 (3802,1037)
	Durchschnitt	3.8816 (4777,1164)	3.3395 (3802,1029)
Funktion auswer-tungen	Start-Min	4 (12)	5 (12)
	Start-Mid	16 (18)	15 (16)
	Start-Max	11 (15)	11 (14)
	Durchschnitt	10.33 (15)	10.33 (10.66)

P4000. Die Startfrequenzen werden bei der Optimierung mit minimaler Hashrate beim Maximum festgesetzt, denn auf Basis dieses Messwertes bei maximalen Frequenzen wird die maximale Hashrate bestimmt. Alternativ könnte man die maximale Hashrate aus einer Online-Liste über eine API abfragen. Da es beim Herstellungsprozess bei Grafikkarten jedoch kleinere Unterschiede gibt, könnte es sein, dass die abgefragte maximale Hashrate bei der eigenen GPU nie erreicht wird.

In Tabelle 5.9 sind die Ergebnisse, sowie die benötigte Anzahl an Funktionsauswertungen der Optimierung mit den verschiedenen Algorithmen zu sehen. Das Tabellenformat ist wie in Abschnitt 5.4.8 beschrieben. Die unterschiedlichen Startpunkte weichen hier jedoch den verschiedenen Suchalgorithmen. Die Optimierungsverläufe sind in Abbildung 5.13 zu finden. Das violette Raster (Titan X) und die grüne Linie (Quadro P4000) markieren den Bereich der Funktion, auf dem auch die Nebenbedingung der einzuhaltenden Hashrate erfüllt ist. Der beste gefundene Wert für die Energieeffizienz, der auf diesem Raster bzw. dieser Linie liegt, ist das Ergebnis der Optimierung. Das Optimum unterliegt Messschwankungen und beträgt 3.05 H/J auf der Quadro P4000 und 3.6 H/J auf der Titan X.

Die verschiedenen Algorithmen liefern ähnliche Resultate. Das Nelder-Mead Verfahren ist aber etwas schlechter in Hinblick auf die gefundenen Werte für das Optimum. Der Optimierungsverlauf in Hinsicht auf die Funktionsauswertungen verläuft ähnlich dem ohne Nebenbedingungen. Das Nelder-Mead Verfahren braucht bei der 2D-Optimierung auf der Titan X weniger bzw. bei der 1D-Optimierung auf der Quadro P4000 mehr Funktionsauswertungen als das Hill Climbing und das Simulated Annealing.

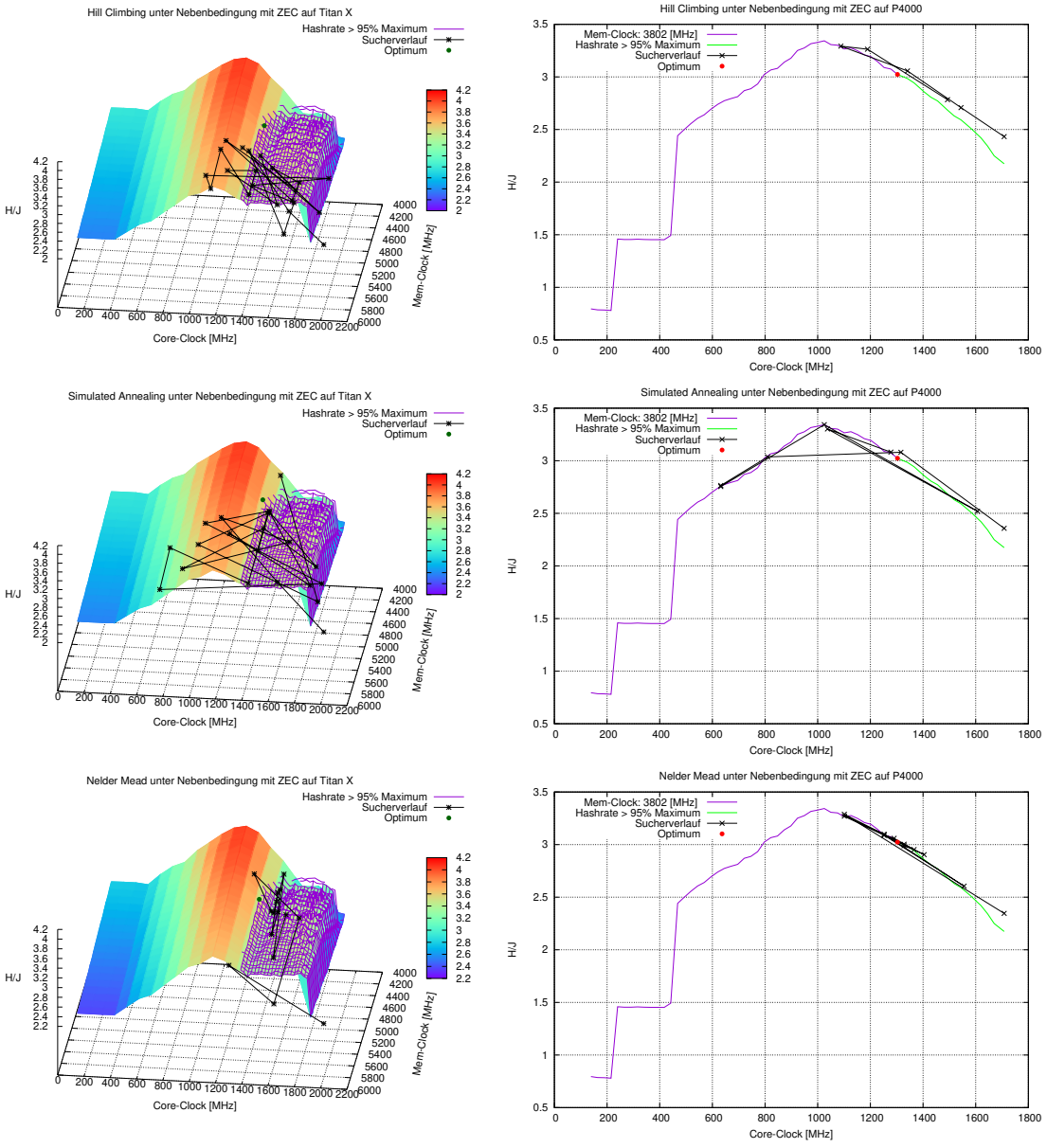

Abb. 5.13: Optimierung unter Nebenbedingung von ZEC: Verlauf der Frequenzoptimierungen mit Hill Climbing (oben), Simulated Annealing (mitte) und Nelder-Mead (unten) mit einer minimal einzuhaltenden Hashrate (95 % max. Hashrate) auf der Titan X (linke Spalte) und der Quadro P4000 (rechte Spalte)

Tab. 5.9: Ergebnis der Suche in Hashes pro Joule inkl. der dazugehörigen Frequenzen und der Anzahl an Funktionsauswertungen der Optimierung von ZEC mit den verschiedenen Suchalgorithmen unter der Nebenbedingung einer minimal einzuhaltenden Hashrate von 95 % (aus [7])

Währung: ZEC		**TITAN X**	**Quadro P4000**
Optimum		ca. 3.6 (4755,1379)	ca. 3.05 (3802,1303)
Ergebnis	Hill Climbing	3.44178 (5421,1430)	3.05715 (3802,1341)
	Simulated Annealing	3.48682(4749,1430)	3.0806 (3802,1316)
	Nelder-Mead	3.3656 (4836,1468)	3.00388 (3802,1328)
Funktion auswertungen	Hill Climbing	12 (22)	3 (6)
	Simulated Annealing	18 (21)	2 (9)
	Nelder-Mead	6 (16)	8 (12)

5.4.12 Monitoring

Für die Evaluierung der *Monitoringphase* wird der Autotuner auf einem System mit den vier GPUs aus Tabelle 5.1 im Zeitraum vom 04.10.18 20:00h bis 07.10.18 20:00h ausgeführt. Die Kryptowährungen ETH, XMR und ZEC, sowie die Altcoins LUX, RVN, BTX und VTC werden hierfür verwendet. Die Energiekosten werden auf 0.1 Euro/kWh festgesetzt, um realitätsnah zu sein. Bei einem Preis von 0.3 Euro pro Kilowattstunden, wie er aktuell in Deutschland beträgt, ist nicht profitabel für das Mining und würde zu Verlusten führen.

Abbildung 5.14 zeigt für diesen Zeitraum den berechneten Mining-Profit bei energieoptimalen Frequenzen für jede Währung auf den einzelnen GPUs. Hier wird jeweils nur die profitabelste Kryptowährung gemint. Abbildung 5.15 zeigt die dabei erzielten Erträge und die Energiekosten. Zusätzlich sieht man den sich daraus berechneten Profit für alle GPUs einzeln und insgesamt.

Zu erkennen ist in den Abbildungen, dass die Titan V die höchsten Mining-Erträge erzielt, gefolgt von der Titan X. Die beiden GPUs GTX 1080 Quadro P4000 liegen im Durchschnitt etwa gleichauf. Die Stromkosten sind bei den beiden Titan GPUs ungefähr gleich hoch. Danach folgt die GTX 1080 und die Quadro P4000, welche am sparsamsten sind. Auf der Titan V, der Titan X und der Quadro P4000 wird hauptsächlich ETH gemint. Auf der GTX 1080 ist es überwiegend LUX. LUX ist zur Zeit der Experimente sehr populär und hatte zur Zeit der Messungen zwischenzeitlich einen hohen Kurs. Die Kursschwankungen von LUX zeigen sich jedoch ganz deutlich auf allen GPUs. Bei den

bekannten Währungen ist der Kurs deutlich stabiler. ZEC hatte vor den Experimente einen starken Einbruch erlebt und ist den Umständen entsprechend die unprofitabelste Währung. Dementsprechend gibt es eine starke Zackenbildung bei den Diagrammen in 5.14 und desöfteren Experimenten Währungen durch andere ersetzt werden. Hier zeigt sich, wie wichtig die Monitoringphase ist. Man würde sonst zwar eine Währung energieeffizient minen, jedoch nicht die profitabelste. Zumindest beim Kryptomining ist diese Phase somit unverzichtbar.

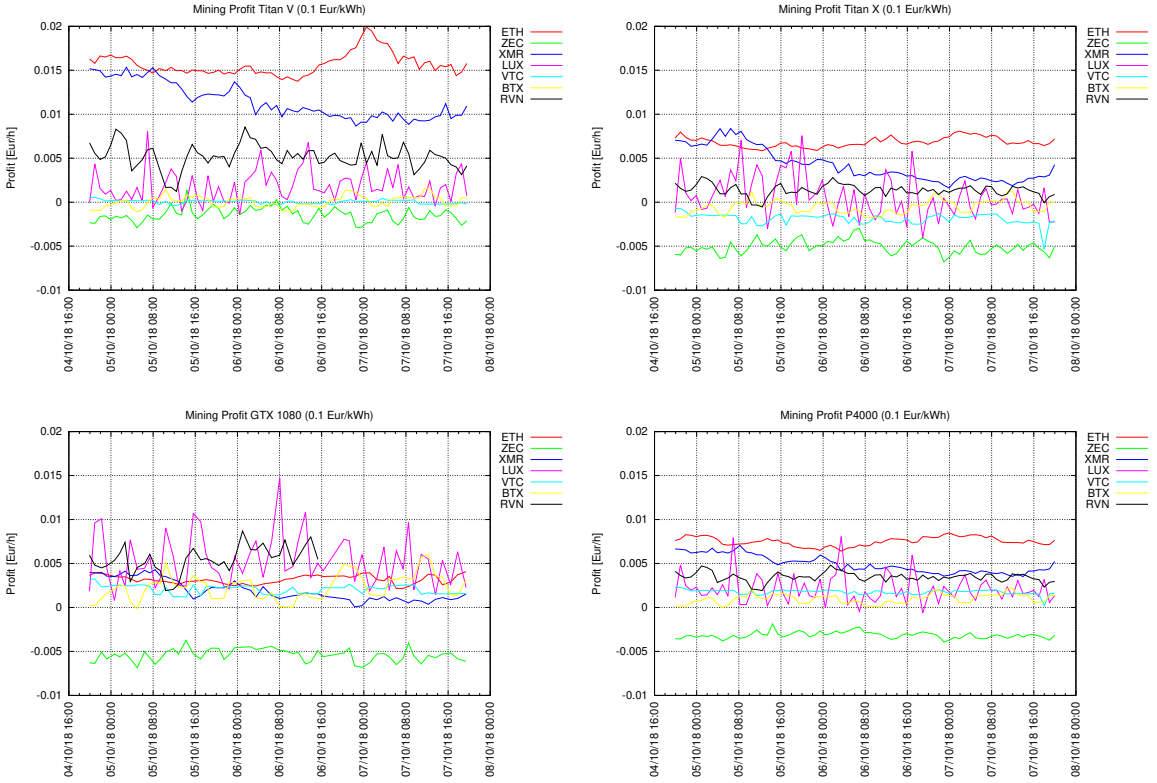

Abb. 5.14: Kalkulierter Mining-Profit der verschiedener Kryptowährungen auf den einzelnen Grafikkarten eines Systems während des Monitorings (aus [7])

5.5 Zusammenfassung

In diesem Kapitel wurden die Möglichkeiten gezeigt, den Energieverbrauch mittels DVFS auf GPU zu senken. Dabei wurde ein Autotuning-Framework zur Erhöhung der Energieeffizienz auf NVIDIA-GPUs verwendet und evaluiert. Als besonderer Anwendungsfall wurde das aktuell sehr beliebte Kryptomining hergenommen. Der Autotuner kann dabei mehrere GPUs eines Rechners verwenden und gleichzeitig über verschiedene Suchstrategien das Energieoptimum finden. Zudem ist der Autotuner in zwei Teile,

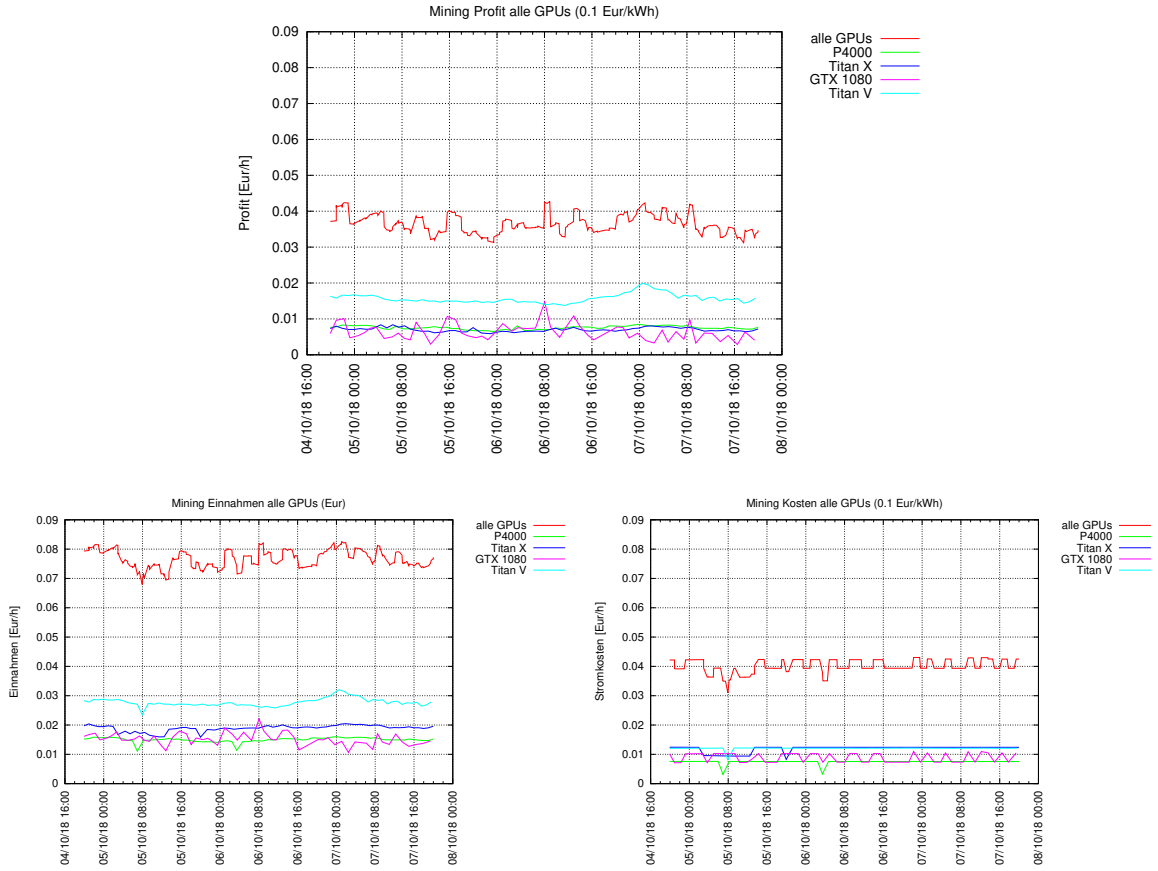

Abb. 5.15: Kalkulierter Mining-Profit (oben), Miningerträge (unten links) und Energiekosten (unten rechts) der jeweils profitabelsten Kryptowährung auf allen Grafikkarten eines Systems während des Monitorings (aus [7])

aufgeteilt: einer Frequenzoptimierungs- und einer Monitoringphase.

In der Frequenzoptimierungsphase erfolgt die Optimierung der Frequenzen auf allen spezifizierten GPUs gleichzeitig, jedoch wird pro GPU jede gewählte Währung einzeln optimiert. Das Konzept der GPU-Gruppen für gleiche GPUs schafft hier Abhilfe und ermöglicht bei der Optimierung eine Aufteilung der Währungen auf die einzelnen GPUs einer Gruppe. Dies ist jedoch mit viel Vorsicht zu genießen: In den Experimenten stellte sich heraus, dass baugleiche GPUs teilweise einen Unterschied in der Leistungsaufnahme von über 10 Watt haben. Wer bei den Frequenzoptimierungen das Maximum an Energieeffizienz herausholen möchte, der sollte trotzdem jede GPUs einzeln betrachten. Auch zeigte sich, dass es einen Unterschied macht, in welchem Slot eine GPU verbaut ist. Die inneren Slots hatten einen minimal höheren Energieverbrauch, da sich hier leichter Wärme staut, was Auswirkungen auf die GPUs hat. Ab gewissen Temperaturen können die GPUs nicht mehr die volle Leistung halten. Wenn die Optima im hohen Frequenzbereich liegen, kann die gewüschnte Leistung (Taktrate) nicht mehr garantiert werden und die GPU wird tiefer getaktet.

Es waren die drei NVIDIA-spezifischen Bibliotheken *NVML*, *NVAPI* und *NV-Control X* nötig, um die Anpassung der Frequenzen in einem möglichst großen Bereich, sowie die Messung des Energiebedarfs auf Windows und Linux zu ermöglichen. Für die Optimierung selbst wurden die drei verschiedene Optimierungsalgorithmen Hill Climbing, Simulated Annealing und Nelder-Mead verwendet. Die Frequenzoptimierungsphase dauerte im Schnitt etwa 45 bis 60 Minuten, wobei die Online-Frequenzoptimierung hier den Großteil ausmacht, da das Messen eines Auswertungspunktes wegen Netzwerklatenzen deutlich länger dauert um verlässliche Werte zu bekommen. Da beim Mining von Kryptowährungen die Miner rund um die Uhr laufen, ist dies wohl vernachlässigbar und macht sich schnell bezahlt. Am Ende der Frequenzoptimierungsphase sind auf allen GPUs für alle verfügbaren Währungen energieoptimale Frequenzen bekannt. In der darauffolgenden Monitoringphase werden Energiekosten und Miningerträge bei den gefundenen optimalen Frequenzen berechnet und das Mining wird mit der profitabelsten Währung gestartet. Energieverbrauch und Miningerträge werden periodisch aktualisiert. Hier werden der aktuelle Börsenpreis sowie die beim Mining-Pool auf Basis der eingereichten Shares erzielte Hashrate berücksichtigt. Ist die aktuell geminte Währung nicht mehr die Profitabelste, erfolgt ein Wechsel der Währung mit anschließender Reoptimierung der Frequenzen.

Der Autotuner wurde mit verschiedenen Grafikkarten und Kryptowähreungen evaluiert. Zunächst wurden die Frequenzen für das energieoptimale Mining mittels einer erschöpfenden Suche ermittelt und ein Vergleich der Energieeffizienz bei optimalen und

den Standardfrequenzen, die vom NVIDIA-Treiber ausgewählt hat, getätigt. Je nach verwendeter Grafikkarte und Kryptowährung wurde eine Steigerung der Energieeffizienz von bis zu 84% ermittelt. Es wurden die verschiedenen Optimierungsalgorithmen anhand des gefundenen Optimums und der benötigten Anzahl an Funktionsauswertungen evaluiert. Abschließend wurde der während der Monitoringphase berechnete Mining-Profit für die verfügbaren Währungen auf einem Rechner mit vier Grafikkarten über einen längeren Zeitraum untersucht.

5.6 Verwandte Arbeiten

Es wurden bereits ähnliche Arbeiten im Bereich DVFS auf GPUs publiziert. So werden in [48] verschiedene Techniken für DVFS auf Grafikkarten vorgestellt und in einer Studie gegenübergestellt. In [47] wird der Effekt von DVFS auf einer NVIDIA Geforce GTX 560 Ti anhand unterschiedlichen Programmen untersucht. Es werden sowohl Core- und VRAM-Frequenzen als auch Core- und VRAM-Voltages mittels den Tools *NVIDIA Inspector* und *MSI Afterburner* manuell angepasst. In [31] wird die Matrixmultiplikation auf die Auswirkungen von DVFS auf GPUs und CPUs am Beipiel untersucht. In [22] wird der Energieverbrauch von GPU-Programmen zur Laufzeit mittels DVFS im Zusammenspiel mit der Überwachung der aktuell genutzten Speicherbandbreite gezeigt. Dieser Artikel war auch die Inspiration für die Entwicklung eines energieeffizienten Autotuners auf GPUs.

Modelle zur Vorhersage des Energieverbrauchs einer GPU bei verschiedenen Core- und VRAM-Frequenzen werden in [26] eingeführt. Mit Machine-Learning-Techniken durch Messdaten von verschiedenen Anwendungen werden die Modelle trainiert. In [49] werden solche Modelle verwendet, um die Energieeffizienz von mobilen Videospielen zu steigern. Die trainierten Modelle werden in einem Power-Management-System verwendet, welches die GPU- und CPU-Frequenzen zur Laufzeit einstellt. Einen Überlick über unterschiedliche Autotuning-Techniken ist in [25] zu finden. Es gibt auch hier eine Kategorisierung statt.

Die kommerzielle Mining-Software *Awesome Miner* [65] kann für jede Grafikkarte und Kryptowährung manuell ein Profil mit Frequenzen, Hashrate und Energieverbrauch erstellen. Auf Basis dieser Profile und entsprechenden Coin-Statistiken wird der Mining-Profit berechnet und immer die jeweils profitabelste Währung gemint.

5.7 Ausblick

In diesem Kapitel werden einige Ideen für eine Erweiterung und weiteren Möglichkeiten des Autotuning auf Grafikkarten vorgestellt.

Anstelle des Autotunings mit Nebenbedingungen wäre es auch denkbar, aus der Single-Objective-Optimierung der Energieeffizienz eine Multi-Objective-Optimierung (Pareto-Optimierung) nach Hashrate und Energieverbrauch zu realisieren. Die Pareto-Optimierung bestünde dann aus einer Menge an Pareto-Optima, auch als Pareto-Front bezeichnet. Ein Pareto-Optimum ist dabei ein Zustand, in dem es nicht möglich ist, ein Kriterium (engl. Objective) zu verbessern, ohne gleichzeitig ein anderes verschlechtern zu müssen, siehe [32].

Eine weitere Möglichkeit zum Auffinden von Frequenzen für energieoptimales Mining ist die Erstellung eines Energiemodells, wie es beispielsweise in den verwandten Arbeiten gemacht wurde. Dabei muss eine Annahme über die Form der zu optimierenden Funktion gemacht werden. Dies hat den Vorteil, dass weniger Funktionsauswertungen und damit zeitintensive Messungen nötig sind.

Aktuell beschränken sich die zu optimierenden Parameter auf die einstellbaren Core- und VRAM-Frequenzen. Möglich wäre auch die Launchparameter der CUDA-Kernel, wie die Anzahl der Blöcke oder die Anzahl der Threads pro Block, zu verändern. Hierfür müsste aber der Quellcode der Miner angepasst werden. Unter Windows ist es mit der *NVAPI* zudem auch möglich, die Core- und VRAM-Voltages einer GPU zu verändern. Hier gilt jedoch, besonders vorsichtig zu sein, um die Hardware nicht zu beschädigen.

Es wäre auch interessant DVFS auf GPUs mit anderen Anwendungen als dem Kryptomining zu validieren. Als Beispiel seien Implementierungsvarianten eines ODE-Löser (ordinary differential equations, dt. gewöhnliche Differentialgleichungen) oder Simulationen genannt. Anstelle der Hashrate wird hier die Laufzeit als Metrik für die Performanz eingesetzt. Hierfür würde die Offline-Frequenzoptimierung des Autotuning-Frameworks ausreichen.

6 | Schluss

In der vorliegenden Arbeit wurden der Einfluss von DVFS auf den Energieverbrauch von Prozessoren an diversen Beispielen untersucht. Das letzte Kapitel resümiert alle Kapitel und zeigt im Abschluss noch einen Ausblick über mögliche künftige Bereiche.

6.1 Zusammenfassung und Fazit

Diese Arbeit beschäftige sich mit dem Energieverbrauch von DVFS Prozessoren im Bereich parallelen Rechnens und Kryptowährungen. DVFS ermöglicht das manuelle Setzen der Frequenzen der Prozessoren zur Laufzeit. Speziell wurden die Auswirkungen der CPU-Frequenz auf CPUs und der Core-Frequenz und VRAM-Frequenz auf GPUs in Zusammenhang mit dem Energieverbrauch untersucht. Dabei haben wurden drei Schwerpunkte gesetzt:

- Metriken zur Bewertung der Energieeffizienz (Kapitel 3),
- modell-basierte Optimierung des Energieverbrauchs (Kapitel 4),
- energie-orientiertes Autotuning auf GPUs am Beispiel von Miningalgorithmen (Kapitel 5).

In Kapitel 3 wurde auf bereits etablierte Metriken wie dem EDP eingegangen und diese dargestellt. In diesem Bereich liegt der Beitrag dieser Arbeit beim Einführen neuer Metriken. Beispielhaft sei der Power Increase Faktor *PI* zu nennen, der den Zuwachs der Leistungsaufnahme bei steigender Kernzahl im Relation zur sequentiellen Leistungsaufnahme festhält. Der *PI* ist sehr hardware-abhängig und tendiert zu unterschiedlichen Steigungen, wie man in den Messungen gesehen hat. Auch ist der Anstieg von 4 auf 8 Threads bei der Haswell CPU ausgeprägter, wohingegen das Hyperthreading auf der Skylake CPU nur einen geringfügigen Einfluss auf die Leistungsaufnahme hat. Eine weitere neu eingeführte Metrik ist der Relative Power Increase Power Faktor *RPI*, der die Korrelation zwischen Speedup und Leistungszuwachs darstellt. Diese neuen Metriken helfen dabei, Programme in Anbetracht des Energieverbrauchs besser bewerten zu

können.

Energiemodelle können verwendet werden, um den Energieverbrauch einer Anwendung analytisch nachzubilden. Mit solchen Modellen kann im bestmöglichen Szenario auch der Energieverbrauch von Programmen a priori vorhergesagt werden. Die eingeführten Energiemodelle für Energie und das EDP eignen sich nicht nur für x86 Prozessoren, sondern auch für ARM-Prozessor. Dies wurde auf einem Odroid XU-4 Einplatinencomputer gezeigt. Um jedoch die Energie und das EDP gut modellieren zu können, müssen die Programme eine gewissen Skalierbarkeit mit sich bringen. Benchmarks die wenig oder gar nicht mit der Anzahl an Threads und der Frequenzsteigerung skalierten, konnten kaum nachgebildet werden. Da die Modelle jedoch auf einer generellen Skalierbarkeit aufbauen, war dies zu erwarten. Des Weiteren wurde gezeigt, dass es möglich ist, nur eine globale, anwendungsunabhängige CPU-Frequenz für ein System abzuleiten und somit den Energieverbrauch zu optimieren. Zwar ist diese eine Frequenz nicht für alle Benchmarks perfekt geeignet, jedoch ist der Aufwand, diese zu bestimmen, deutlich geringer, als für jeden einzelnen Benchmark eine eigene, anwendungsabhängige Frequenz.

Autotuning ermöglicht das Tunen von Paramtern, wie beispielsweise die CPU-Frequenz oder die VRAM-Frequenz auf Grafikkarten zur Laufzeit eines Programms. Über Suchstrategien werden verschiedene Kombinationen dieser Paramameter ausgewertet, um eine definierte Zielfunktion zu maximieren oder zu minimieren. Da auch Grafikkarten DVFS-Prozessoren sind und sich hier auch die Energie messen und verschiedene Frequenzen einstellen lassen, wurde am Beispiel von Kryptowährungen das Potential von DVFS auf GPUs mithilfe von Autotuning gemessen. Die großflächige Evaluation ergab eine Effizienzsteigerung von bis zu 84 % im Vergleich zu den Standardfrequenzen beim Mining der Kryptowährung Monero. Alle anderen getesteten Währungen haben vom Autotuning der Frequenzen profitiert. Der limitierende Faktor war die Hardware, da nur wenige Grafikkarten eine große Breite an einstellbaren Frequenzen geboten haben. So war es schwierig, kleinere Sprünge bei den Frequenzen zu messen, da sich sowohl Energieverbrauch, als auch die Leistungsaufnahme nur wenig veränderten.

A | Systeme

Nachfolgender Anhang gibt eine Übersicht über die in dieser Arbeit eingesetzten Computer-Systeme. Die meisten Informationen können einfach mit `lscpu` auf der Linux-Kommandozeile abgerufen werden. Fehlende Informationen wurden von der Intel-Website, von `https://en.wikichip.org/wiki/WikiChip` und `www.cpu-world.com` übernommen, da nicht alle Informationen auf jeder Quelle zu finden waren.
Manchmal gab es bei Prozessoren kein einheitliches Abstufen der einstellbaren Frequenzen. So fangen alle der Intel Core i7 Reihe bei 800 MHz an und steigert sich vorerst in 200 MHz Schritten bis 1,4 GHz und macht ab hier bei 1,5 GHz weiter. Deswegen findet man in den Tabellen manchmal 2 Werte beim Stepping.

A.1 Skylake

Intel Core i7-6700 Prozessor	
CPUs	8
Threads pro Kern	2
Kerne pro Socket	8
Einführungsdatum	Q3'15
Lithographie	14nm
Sockets	1
NUMA Knoten	1
CPU MHz min	800 MHz
CPU MHz max	3400 MHz
CPU MHz Turbo	4000 MHz
CPU MHz Step	100/200 MHz
TDP	65 W
L1d Cache	128 KiB (4x32 KiB, 8-way set associative)
L1i Cache	128 KiB (4x32 KiB, 8-way set associative)
L2 Cache	1 MiB (4x256 KiB, 4-way set associative)
L3 Cache	8 MiB (4x2 MiB, 16-way set associative shared)
RAM	16 GB DDR3
Betriebssystem	OpenSuse 13.2

A.2 Haswell

Intel Core i7-4770 Prozessor	
CPUs	8
Threads pro Kern	2
Kerne pro Socket	8
Einführungsdatum	Q2'13
Lithographie	22nm
Sockets	1
CPU MHz min	800 MHz
CPU MHz max	3400 MHz
CPU MHz Turbo	3900 MHz
CPU MHz Step	100/200 MHz
TDP	84 W
L1d Cache	128 KiB (4x32 KiB, 8-way set associative)
L1i Cache	128 KiB (4x32 KiB, 8-way set associative)
L2 Cache	1 MiB (4x256 KiB, 4-way set associative)
L3 Cache	8 MiB (4x2 MiB, 16-way set associative shared)
RAM	16 GB DDR3
Betriebssystem	OpenSuse 13.2

A.3 Kaby Lake

Intel Core i7-7700 Prozessor	
CPUs	8
Threads pro Kern	2
Kerne pro Socket	8
Einführungsdatum	Q1'17
Lithographie	14nm
Sockets	1
CPU MHz min	800 MHz
CPU MHz max	3600 MHz
CPU MHz Turbo	4200 MHz
CPU MHz Step	200 MHz
TDP	65 W
L1d Cache	128 KiB (4x32 KiB, 8-way set associative)
L1i Cache	128 KiB (4x32 KiB, 8-way set associative)
L2 Cache	1 MiB (4x256 KiB, 4-way set associative)
L3 Cache	8 MiB (4x2 MiB, 16-way set associative shared)
RAM	16 GB DDR4
Betriebssystem	OpenSuse 15.1

A.4 Skylake-W

Intel Xeon W-2123 Prozessor	
CPUs	8
Threads pro Kern	2
Kerne pro Socket	8
Einführungsdatum	Q3'17
Lithographie	14nm
Sockets	1
CPU MHz min	1200 MHz
CPU MHz max	3600 MHz
CPU MHz Turbo	3900 MHz
CPU MHz Step	100/200 MHz
TDP	120 W
L1d Cache	128 KiB (4x32 KB, 8-way set associative)
L1i Cache	128 KiB (4x32 KB, 8-way set associative)
L2 Cache	4 x 1 MB (16-way set associative)
L3 Cache	8,25 MiB (8.25 MB 11-way set associative shared)
RAM	16 GB DDR4
Betriebssystem	OpenSuse 15.1

A.5 Odroid XU-4

Odroid XU-4	
CPU	Exynos 5 Octa (5422)
CPU 1	Cortex-A15 (2.1GHz Quad-Core)
CPU 1	Cortex-A7 (1.4GHz Quad-Core)
Threads pro Kern	1
Kerne pro Socket	4
Einführungsdatum	Q1'14
Lithographie	32nm
Sockets	2
NUMA Knoten	1
CPU MHz min	100 MHz
CPU MHz max	2000 MHz
CPU MHz Step	100 MHz
TDP	15 W
Befehlssatz	ARMv7
RAM	2GB LPDDR3e
Betriebssystem	Ubuntu 16.4

Literatur

Eigene Publikationen

[1] T. Rauber, G. Rünger und M. Stachowski. „Model-based optimization of the energy efficiency of multi-threaded applications“. In: *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*. 2017, S. 1–6. DOI: 10.1109/IGCC.2017.8323578.

[2] T. Rauber, G. Rünger und M. Stachowski. „Model-based optimization of the energy efficiency of multi-threaded applications“. In: *Sustainable Computing: Informatics and Systems* 22 (2019), S. 44–61. ISSN: 2210-5379. DOI: https://doi.org/10.1016/j.suscom.2019.01.022. URL: http://www.sciencedirect.com/science/article/pii/S221053791830091X.

[3] T. Rauber, G. Rünger und M. Stachowski. „Performance and Energy Metrics for Multi-threaded Applications on DVFS Processors“. In: *Sustainable Computing: Informatics and Systems* 17 (2017), S. 55–68. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2017.10.015.

[4] T. Rauber, G. Rünger und M. Stachowski. „Towards New Metrics for Appraising Performance and Energy Efficiency of Parallel Scientific Programs“. In: *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Juni 2017, S. 466–474. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.75.

[5] M. Stachowski, A. Fiebig und T. Rauber. „Autotuning based on frequency scaling toward energy efficiency of blockchain algorithms on graphics processing units“. In: *The Journal of Supercomputing (2020)* (Apr. 2020). DOI: https://doi.org/10.1007/s11227-020-03263-5.

[6] M. Stachowski u. a. „Influence of locality on the scalability of method- and system-parallel explicit peer methods“. In: *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*. Sep. 2016, S. 685–694.

Betreute Abschlussarbeiten

[7] A. Fiebig. „Entwicklung eines Autotuning Frameworks mittels DVFS zur Steigerung der Energieeffizienz von NVIDIA Grafikkarten am Beispiel von Mining Algorithmen“. Master's Thesis. University of Bayreuth, Dez. 2018.

[8] A. Fritzmann. „Evaluierung und Erweiterung von Active Harmony zum automatischen Tunen des Energieverbrauchs von DFVS-Prozessoren“. Bachelor's Thesis. University of Bayreuth, Okt. 2018.

[9] N. Hilberg. „Erweiterung eines C++ Analyse-Tools zum Erfassen und Bewerten des Energieverbrauchs verschiedener Benchmarks für verteilte Systeme“. Bachelor's Thesis. University of Bayreuth, Dez. 2017.

[10] D. Kober. „Erweiterung von OpenTuner zum automatischen Optimieren des Energieverbrauchs vvon DVFS-Prozessoren im Vergleich zu anderen Frameworks“. Master's Thesis. University of Bayreuth, Sep. 2018.

[11] M. Malik. „Implementation and Evaluation of different data fitting methods for inverse Modeling of Energy Consumption“. Master's Thesis. University of Bayreuth, Sep. 2018.

[12] D. Nickl. „Analyse und Veranschaulichung des Energieverhaltens verschiedener DVFS-Prozessoren am Beispiel der PASRSEC und SPLASH-2 Benchmarksuites unter Berücksichtigung der Uncorefrequenz“. Bachelor's Thesis. University of Bayreuth, Juni 2021.

[13] H. Pillny. „Entwicklung eines Plugins für GKrellM für das Monitoring des Energieverbrauchs für Intel Prozessoren“. Bachelor's Thesis. University of Bayreuth, Juni 2021.

[14] C. Prell. „Interrelations between different implementation variants of the matrix multiplication and the power consumption of DVFS processors“. Master's Thesis. University of Bayreuth, Sep. 2018.

Fremdquellen

[15] S. Abdulsalam u. a. „Using the Greenup, Powerup, and Speedup metrics to evaluate software energy efficiency“. In: *IEEE Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*. Dez. 2015, S. 1–8. DOI: 10.1109/IGCC.2015.7393699.

[16] S. Albers. „Energy-efficient algorithms“. In: *Commun. ACM* 53.5 (Mai 2010), S. 86–96. ISSN: 0001-0782. DOI: 10.1145/1735223.1735245. URL: http://doi.acm.org/10.1145/1735223.1735245.

[17] J. Ansel u. a. „OpenTuner: An Extensible Framework for Program Autotuning“. In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. PACT '14. Edmonton, AB, Canada: ACM, 2014, S. 303–316. ISBN: 978-1-4503-2809-8. DOI: 10.1145/2628071.2628092. URL: http://doi.acm.org/10.1145/2628071.2628092.

[18] C. Bienia, S. Kumar und K. Li. „PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors“. In: *2008 IEEE International Symposium on Workload Characterization*. 2008, S. 47–56. DOI: 10.1109/IISWC.2008.4636090.

[19] C. Bienia u. a. „The PARSEC Benchmark Suite: Characterization and Architectural Implications“. In: *ACM: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. PACT '08. Toronto, Ontario, Canada: Association for Computing Machinery, 2008, S. 72–81. ISBN: 9781605582825. DOI: 10.1145/1454115.1454128.

[20] J. Bilmes u. a. „Optimizing matrix multiply using PHiPAC: A portable high-performance ANSI C coding methodology“. In: *Proceedings of the International Conference on Supercomputing* (Juni 2014). DOI: 10.1145/2591635.2591656.

[21] J.A. Butts und G.S. Sohi. „A static power model for architects“. In: *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*. 2000, S. 191–201. DOI: 10.1109/MICRO.2000.898070.

[22] D. M. Cameirinha. „Exploiting DVFS for GPU Energy Management“. dissertation. Tecnico Lisboa, 2015. URL: https://fenix.tecnico.ulisboa.pt/downloadFile/563345090414604/Dissertacao.pdf.

[23] J. Choi u. a. „Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks“. In: *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. Mai 2014, S. 447–457. DOI: 10.1109/IPDPS.2014.54.

[24] M. Chrobak. „Algorithmic Aspects of Energy-Efficient Computing“. In: *Handbook of Energy-Aware and Green Computing*. Hrsg. von I. Ahmad und S. Ranka. CRC Press, 2012, S. 311–329.

[25] J.J. Durillo und T. Fahringer. „From Single- to Multi-Objective Auto-Tuning of Programs: Advantages and Implications“. In: *Scientific Programming* 22 (Jan. 2014), S. 285–297. DOI: 10.1155/2014/818579.

[26] B. Dutta, V. Adhinarayanan und W. Feng. „GPU power prediction via ensemble machine learning for DVFS space exploration“. In: Mai 2018, S. 240–243. DOI: 10.1145/3203217.3203273.

[27] H. Esmaeilzadeh u. a. „Power challenges may end the multicore era“. In: *Commun. ACM* 56.2 (Feb. 2013), S. 93–102. ISSN: 0001-0782. DOI: 10.1145/2408776.2408797. URL: http://doi.acm.org/10.1145/2408776.2408797.

[28] A. Fedorova u. a. „Maximizing Power Efficiency with Asymmetric Multicore Systems“. In: *Commun. ACM* 52.12 (Dez. 2009), S. 48–57. ISSN: 0001-0782. DOI: 10.1145/1610252.1610270. URL: http://doi.acm.org/10.1145/1610252.1610270.

[29] M. Frigo und S. Johnson. „The Design and Implementation of FFTW3“. In: *Proceedings of the IEEE* 93.2 (2005). Special issue on “Program Generation, Optimization, and Platform Adaptation”, S. 216–231.

[30] R. Ge und K. W. Cameron. „Power-Aware Speedup“. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. März 2007, S. 1–10. DOI: 10.1109/IPDPS.2007.370246.

[31] R. Ge u. a. „Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU“. In: *2013 42nd International Conference on Parallel Processing*. 2013, S. 826–833. DOI: 10.1109/ICPP.2013.98.

[32] M. Geilen und T. Basten. „A Calculator for Pareto Points“. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '07. Nice, France: EDA Consortium, 2007, S. 285–290. ISBN: 9783981080124.

[33] V. Getov u. a. „Towards an Application-specific Thermal Energy Model of Current Processors“. In: *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. E2SC '15. Austin, Texas: ACM, 2015, 5:1–5:10. ISBN: 978-1-4503-3994-0. DOI: 10.1145/2834800.2834805. URL: http://doi.acm.org/10.1145/2834800.2834805.

[34] R. Gonzalez und M. Horowitz. „Energy dissipation in general purpose microprocessors“. In: *IEEE Journal of Solid-State Circuits* 31.9 (Sep. 1996), S. 1277–1284. ISSN: 0018-9200. DOI: 10.1109/4.535411.

[35] G. Hager u. a. „Exploring performance and power properties of modern multicore chips via simple machine models“. In: *Concurrency and Computation-Practice & Experience* 28(2) (2016), S. 189–210. ISSN: 1532-0626. DOI: 10.1002/cpe.3180. eprint: 1208.2908. URL: http://arxiv.org/abs/1208.2908.

[36] J. Hofmann, G. Hager und D. Fey. „On the accuracy and usefulness of analytic energy models for contemporary multicore processors“. In: *33rd International Conference, ISC High Performance 2018* (Mai 2018), S. 22–43. DOI: 10.1007/978-3-319-92040-5_2.

[37] S. Holmbacka und J. Keller. „Workload Type-Aware Scheduling on big.LITTLE Platforms“. In: *Algorithms and Architectures for Parallel Processing -17thInternational Conference, ICA3PP 2017, Helsinki, Finland, August21-23, 2017, Proceedings*. Hrsg. von Shadi Ibrahim u. a. Bd. 10393. Lecture Notes in Computer Science. Springer, 2017, S. 3–17. DOI: 10.1007/978-3-319-65482-9_1. URL: https://doi.org/10.1007/978-3-319-65482-9%5C_1.

[38] M. Horowitz, T. Indermaur und R. Gonzalez. „Low-power digital design“. In: *Proceedings of 1994 IEEE Symposium on Low Power Electronics* (Okt. 1994), S. 8–11.

[39] C. H. Hsu, W. C. Feng und J. S. Archuleta. „Towards efficient supercomputing: a quest for the right metric“. In: *19th IEEE Int. Parallel and Distributed Proc. Symp.* Apr. 2005, S. 8. DOI: 10.1109/IPDPS.2005.440.

[40] S. Irani, S. Shukla und R. Gupta. „Algorithms for Power Savings“. In: *ACM Trans. Algorithms* 3.4 (Nov. 2007), 41–es. ISSN: 1549-6325. DOI: 10.1145/1290672.1290678. URL: https://doi.org/10.1145/1290672.1290678.

[41] R. Jejurikar, C. Pereira und R. Gupta. „Leakage aware dynamic voltage scaling for real-time embedded systems“. In: *Proc. of the 41st Annual Design Automation Conference*. ACM, 2004, S. 275–280. ISBN: 1-58113-828-8.

[42] A. Kanduri u. a. „A perspective on dark silicon“. In: *Springer: The Dark Side of Silicon*. Springer, 2017, S. 3–20.

[43] L. Keqin. „Performance Analysis of Power-Aware Task Scheduling Algorithms on Multiprocessor Computers with Dynamic Voltage and Speed“. In: *IEEE Trans. Parallel Distrib. Syst.* 19.11 (2008), S. 1484–1497. DOI: http://dx.doi.org/10.1109/TPDS.2008.122.

[44] C. Kessler, J. Keller und S. Litzinger. „Temperature-Aware Energy-Optimal Scheduling of Moldable Streaming Tasks onto 2D-Mesh-Based Many-Core CPUs with DVFS“. In: *Proc. 24th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2021)*. Mai 2021.

[45] N. Kitai u. a. „An Auto-tuning with Adaptation of A64 Scalable Vector Extension for SPIRAL“. In: *International Workshop on Automatic Performance Tuning (iWAPT)*. 2021.

[46] J. Mair, K. Leung und Z. Huang. „Metrics and task scheduling policies for energy saving in multicore computers“. In: *2010 11th IEEE/ACM International Conference on Grid Computing*. Okt. 2010, S. 266–273. DOI: 10.1109/GRID.2010.5697984.

[47] X. Mei u. a. „A Measurement Study of GPU DVFS on Energy Conservation“. In: *Proceedings of the Workshop on Power-Aware Computing and Systems*. HotPower '13. Farmington, Pennsylvania: Association for Computing Machinery, 2013. ISBN: 9781450324588. DOI: 10.1145/2525526.2525852. URL: https://doi.org/10.1145/2525526.2525852.

[48] A. Mishra und N. Khare. „Analysis of DVFS Techniques for Improving the GPU Energy Efficiency“. In: *Open Journal of Energy Efficiency* 04 (Jan. 2015), S. 77–86. DOI: 10.4236/ojee.2015.44009.

[49] J-G. Park, N. Dutt und S-S. Lim. „ML-Gov: a machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming“. In: Okt. 2017, S. 12–21. DOI: 10.1145/3139315.3139317.

[50] T. Rauber und G. Rünger. „Modeling and Analyzing the Energy Consumption of Fork-Join-based Task Parallel Programs“. In: *Concurrency and Computation: Practice and Experience* 27.1 (2015), S. 211–236. ISSN: 1532-0634. DOI: 10.1002/cpe.3219.

[51] T. Rauber u. a. „Energy Measurement, Modeling, and Prediction for Processors with Frequency Scaling“. In: *Springer: The Journal of Supercomputing* (2014). ISSN: 0920-8542. DOI: 10.1007/s11227-014-1236-4.

[52] G. Rong u. a. „PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications“. In: *IEEE Transactions on Parallel and Distributed Systems* 21.5 (Mai 2010), S. 658–671. ISSN: 1045-9219.

[53] B. Rountree u. a. „Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound“. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum* (Mai 2012), S. 947–953. DOI: 10.1109/IPDPSW.2012.116.

[54] E. Saxe. „Power-efficient software“. In: *Commun. ACM* 53.2 (2010), S. 44–48. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/1646353.1646370.

[55] B. Shirazi. „Sustainable Computing: Informatics and Systems“. In: *Elsevier: Sustainable Computing: Informatics and Systems* 22 (Juni 2019). ISSN: 2210-5379.

[56] C. Ţăpuş, I. Chung und J.K. Hollingsworth. „Active Harmony: Towards Automated Performance Tuning“. In: *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. SC '02. Baltimore, Maryland: IEEE Computer Society Press, 2002, S. 1–11. ISBN: 0-7695-1524-X. URL: http://dl.acm.org/citation.cfm?id=762761.762771.

[57] D. Terpstra u. a. *Collecting Performance Data with PAPI-C*. Hrsg. von Matthias S. Müller u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 157–173. ISBN: 978-3-642-11261-4.

[58] A. Tiwari u. a. „A scalable auto-tuning framework for compiler optimization“. In: *2009 IEEE International Symposium on Parallel Distributed Processing*. Mai 2009, S. 1–12. DOI: 10.1109/IPDPS.2009.5161054.

[59] M. Tolentino und K. W. Cameron. „The Optimist, the Pessimist, and the Global Race to Exascale in 20 Megawatts“. In: *IEEE: Computer* 45.1 (Jan. 2012), S. 95–97. ISSN: 0018-9162. DOI: 10.1109/MC.2012.34.

[60] R. Vuduc, J. Demmel und K. Yelick. „OSKI: A library of automatically tuned sparse matrix kernels“. In: *Journal of Physics Conference Series* 16 (Jan. 2005), S. 521–530. DOI: 10.1088/1742-6596/16/1/071.

[61] V. M. Weaver u. a. „Measuring Energy and Power with PAPI“. In: *2012 41st International Conference on Parallel Processing Workshops*. 2012, S. 262–268. DOI: 10.1109/ICPPW.2012.39.

[62] R. Clint Whaley. „ATLAS (Automatically Tuned Linear Algebra Software)“. In: *Encyclopedia of Parallel Computing*. Hrsg. von David Padua. Boston, MA: Springer US, 2011, S. 95–101. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_85. URL: https://doi.org/10.1007/978-0-387-09766-4_85.

[63] L. Xiu. „Time Moore: Exploiting Moore's Law From The Perspective of Time“. In: *IEEE Solid-State Circuits Magazine* 11.1 (2019), S. 39–55. ISSN: 1943-0582. DOI: 10.1109/MSSC.2018.2882285.

[64] J. Zhuo und C. Chakrabarti. „Energy-efficient dynamic task scheduling algorithms for DVS systems“. In: *ACM Trans. Embed. Comput. Syst.* 7.2 (2008), S. 1–25. ISSN: 1539-9087.

Onlinequellen

[65] IntelliBreeze Software AB. *Awesome Miner*. 2018. URL: http://www.awesomeminer.com/home (besucht am 24. 10. 2021).

[66] ETHZ ASL. *Numerical methods*. 2018. URL: https://github.com/ethz-asl/numerical_methods (besucht am 24. 10. 2021).

[67] Dominik Brodowski. *Index of /doc/Documentation/cpu-freq/*. URL: https://www.kernel.org/doc/Documentation/cpu-freq/ (besucht am 25. 10. 2019).

[68] Jade Cheng. *Numerical Optimization*. 2018. URL: http://www.jade-cheng.com/au/coalhmm/optimization/ (besucht am 24. 10. 2021).

[69] CryptoCompare.com. *CryptoCompare API*. 2018. URL: https://www.cryptocompare.com/api/#-api-data-price- (besucht am 24. 10. 2021).

[70] Martin Dimitrov. *Intel Power Governor*. URL: https://software.intel.com/en-us/articles/intel-power-governor (besucht am 25. 10. 2019).

[71] Eklektix. *Power Capping Framework*. URL: https://lwn.net/Articles/574224/ (besucht am 25. 02. 2020).

[72] fireice-uk. *xmr-stak*. 2018. URL: https://github.com/fireice-uk/xmr-stak (besucht am 24. 10. 2021).

[73] Bitcoin Forum. *Ethereum (ETH) mining profit formula*. 2017. URL: https://bitcointalk.org/index.php?topic=2262328.0 (besucht am 24. 10. 2021).

[74] *Green 500*. URL: https://www.top500.org/ (besucht am 30. 03. 2019).

[75] *GreenCom-2019*. URL: http://cse.stfx.ca/~cybermatics/2019/greencom/ (besucht am 30. 03. 2019).

[76] Thomas Gruber. *RRZE-HPC/likwid*. URL: https://github.com/RRZE-HPC/likwid (besucht am 25. 02. 2020).

[77] *IGSC 2019*. URL: https://www.igscc.org/ (besucht am 30. 03. 2019).

[78] Thorge Jans. *Greenwashing – Die dunkle Seite der CSR*. Nov. 2018. URL: https://reset.org/knowledge/greenwashing-%E2%80%93-die-dunkle-seite-der-csr (besucht am 24. 10. 2021).

[79] Sachin Joglekar. *Nelder-Mead Optimization*. 2016. URL: https://codesachin.wordpress.com/2016/01/16/nelder-mead-optimization/ (besucht am 24. 10. 2021).

[80] Georgios Konstantopoulos. *Understanding Blockchain Fundamentals*. 2017. URL: https://medium.com/loom-network/search?q=Understanding%20Blockchain%20Fundamentals (besucht am 24. 10. 2021).

[81] Georgios Konstantopupulus. *Understanding Blockchain Funtamentals*. 2017. URL: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-stake-b6ae907c7edb.

[82] Oak Ridge National Laboratory. *SUMMIT*. 2018. URL: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/ (besucht am 24. 10. 2021).

[83] Nicehash. *excavator*. 2018. URL: https://github.com/nicehash/excavator (besucht am 24. 10. 2021).

[84] NVIDIA. *NV-CONTROL X Extension - API specificiation v 1.6*. 2018. URL: https://github.com/NVIDIA/nvidia-settings/blob/master/doc/NV-CONTROL-API.txt (besucht am 24. 10. 2021).

[85] NVIDIA. *NVAPI*. 2018. URL: https://developer.nvidia.com/nvapi (besucht am 24. 10. 2021).

[86] NVIDIA. *NVIDIA Management Library (NVML)*. 2018. URL: https://developer.nvidia.com/nvidia-management-library-nvml (besucht am 24. 10. 2021).

[87] NVIDIA. *NVIDIA Visual Profiler*. 2018. URL: https://developer.nvidia.com/nvidia-visual-profiler (besucht am 24. 10. 2021).

[88] Stefan Oberhumer. *ethminer*. 2018. URL: https://github.com/ethereum-mining/ethminer (besucht am 24. 10. 2021).

[89] Tanguy Pruvot. *ccminer*. 2018. URL: https://github.com/tpruvot/ccminer (besucht am 24. 10. 2021).

[90] AreaMobile Redaktion. *So viel Strom verbrauchen alle Smartphones in der EU*. 2017. URL: https://www.areamobile.de/Akku-Thema-275122/News/So-viel-Strom-verbrauchen-alle-Smartphones-in-der-EU-1322415/ (besucht am 24. 10. 2021).

[91] M. Stachowski und A. Fiebig. *DVFS GPU Autotuning Framework*. URL: https://github.com/UBT-AI2/dvfs_gpu (besucht am 25. 06. 2020).

[92] TOP500 Supercomputers. *PERFORMANCE DEVELOPMENT*. URL: https://www.top500.org/statistics/perfdevel/ (besucht am 25. 09. 2019).

[93] *TOP500 Supercomputers*. URL: https://www.top500.org/ (besucht am 30. 03. 2019).

[94] U.S. Department of Energy. *Exascale Challenges | U.S. DOE Office of Science (SC)*. März 2016. URL: https://science.energy.gov/ascr/research/scidac/exascale-challenges/ (besucht am 30. 03. 2019).

[95] unbekannt. *CPU frequency scaling*. URL: https://wiki.archlinux.org/index.php/CPU_frequency_scaling (besucht am 25. 10. 2019).

[96] whattomine.com. *Coin Calculators*. 2018. URL: https://whattomine.com/calculators (besucht am 24. 10. 2021).