TECHNISCHE UNIVERSITÄT DRESDEN

Fakultat Informatik Institut für Technische Informatik, Professur für Rechnerarchitektur

Ein Framework zur Optimierung der Energieeffizienz von HPC-Anwendungen auf der Basis von Machine-Learning-Methoden

Andreas Gocht-Zech

Dissertation

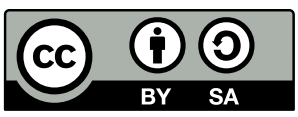
zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

Erstgutachter
Prof. Wolfgang E. Nagel

Zweitgutachter
Prof. Kristian Kersting

Eingereicht am: 14.04.2022
Verteidigt am: 30.09.2022



Zusammenfassung

Ein üblicher Ansatzpunkt zur Verbesserung der Energieeffizienz im High Performance Computing (HPC) ist, neben Verbesserungen an der Hardware oder einer effizienteren Nachnutzung der Wärme des Systems, die Optimierung der ausgeführten Programme. Dazu können zum Beispiel energieoptimale Einstellungen, wie die Frequenzen des Prozessors, für verschiedene Programmfunktionen bestimmt werden, um diese dann im späteren Verlauf des Programmes anwenden zu können. Mit jeder Änderung des Programmes kann sich dessen optimale Einstellung ändern, weshalb diese zeitaufwendig neu bestimmt werden muss. Das stellt eine wesentliche Hürde für die Anwendung solcher Verfahren dar. Dieser Prozess des Bestimmens der optimalen Frequenzen kann mithilfe von Machine-Learning-Methoden vereinfacht werden, wie in dieser Arbeit gezeigt wird. So lässt sich mithilfe von sogenannten Performance-Events ein neuronales Netz erstellen, mit dem während der Ausführung des Programmes die optimalen Frequenzen automatisch geschätzt werden können. Performance-Events sind prozessorintern und können Einblick in die Abläufe im Prozessor gewähren.

Bei dem Einsatz von Performance-Events gilt es einige Fallstricke zu vermeiden. So werden die Performance-Events von Performance-Countern gezählt. Die Anzahl der Counter ist allerdings begrenzt, womit auch die Anzahl der Events, die gleichzeitig gezählt werden können, limitiert ist. Eine für diese Arbeit wesentliche Fragestellung ist also: Welche dieser Events sind relevant und müssen gezählt werden?

Bei der Beantwortung dieser Frage sind Merkmalsauswahlverfahren hilfreich, besonders sogenannte Filtermethoden, bei denen die Merkmale vor dem Training ausgewählt werden. Viele bekannte Methoden gehen dabei entweder davon aus, dass die Zusammenhänge zwischen den Merkmalen linear sind, wie z. B. bei Verfahren, die den Pearson-Korrelationskoeffizienten verwenden, oder die Daten müssen in Klassen eingeteilt werden, wie etwa bei Verfahren, die auf der Transinformation beruhen. Beides ist für Performance-Events nicht ideal. Auf der einen Seite können keine linearen Zusammenhänge angenommen werden. Auf der anderen Seite bedeutet das Einteilen in Klassen einen Verlust an Information.

Um diese Probleme zu adressieren, werden in dieser Arbeit bestehende Merkmalsauswahlverfahren mit den dazugehörigen Algorithmen analysiert, neue Verfahren entworfen und miteinander verglichen. Es zeigt sich, dass mit neuen Verfahren, die auf sogenannten Copulas basieren, auch nichtlineare Zusammenhänge erkannt werden können, ohne dass die Daten in Klassen eingeteilt werden müssen. So lassen sich schließlich einige Events identifiziert, die zusammen mit neuronalen Netzen genutzt werden können, um die Energieeffizienz von HPC-Anwendung zu steigern.

Das in dieser Arbeit erstellte Framework erfüllt dabei neben der Auswahl der Performance-Events weitere Aufgaben: Es stellt sicher, dass diverse Programmteile mit verschiedenen optimalen Einstellungen voneinander unterschieden werden können. Darüber hinaus sorgt das Framework dafür, dass genügend Daten erzeugt werden, um ein neuronales Netz zu trainieren, und dass dieses Netz später einfach genutzt werden kann. Dabei ist das Framework so flexibel, dass auch andere Machine-Learning-Methoden getestet werden können. Die Leistungsfähigkeit des Frameworks wird abschließend in einer Ende-zu-Ende-Evaluierung an einem beispielhaften Programm demonstriert. Die Evaluierung illustriert, dass bei nur 7 % längerer Laufzeit eine Energieeinsparung von 24 % erzielt werden kann und zeigt damit, dass mit Machine-Learning-Methoden wesentliche Energieeinsparungen erreicht werden können.

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultat Informatik Institut für Technische Informatik, Professur für Rechnerarchitektur

Abstract

There are a variety of different approaches to improve energy efficiency in High Performance Computing (HPC). Besides advances to the hardware or cooling systems, optimising the executed programmes' energy efficiency is another a promising approach. Determining energy-optimal settings of program functions, such as the processor frequency, can be applied during the program's execution to reduce energy consumption. However, when the program is modified, the optimal setting might change. Therefore, the energy-optimal settings need to be determined again, which is a time-consuming process and a significant impediment for applying such methods. Fortunately, finding the optimal frequencies can be simplified using machine learning methods, as shown in this thesis. With the help of so-called performance events, a neural network can be trained, which can automatically estimate the optimal processor frequencies during program execution. Performance events are processor-specific and can provide insight into the procedures of a processor.

However, there are some pitfalls to be avoided when using performance events. Performance events are counted by performance counters, but as the number of counters is limited, the number of events that can be counted simultaneously is also limited. This poses the question of which of these events are relevant and need to be counted.

In answering this question, feature selection methods are helpful, especially so-called filter methods, where features are selected before the training. Unfortunately, many feature selection methods either assume a linear correlation between the features, such as methods using the Pearson correlation coefficient or require data split into classes, particularly methods based on mutual information. Neither can be applied to performance events as linear correlation cannot be assumed, and splitting the data into classes would result in a loss of information.

In order to address that problem, this thesis analyses existing feature selection methods together with their corresponding algorithms, designs new methods, and compares different feature selection methods. By utilising new methods based on the mathematical concept of copulas, it was possible to detect non-linear correlations without splitting the data into classes. Thus, several performance events could be identified, which can be utilised together with neural networks to increase the energy efficiency of HPC applications.

In addition to selecting performance events, the created framework ensures that different programme parts, which might have different optimal settings, can be identified. Moreover, it assures that sufficient data for the training of the neural networks is generated and that the network can easily be applied. Furthermore, the framework is flexible enough to evaluate other machine learning methods. Finally, an end-to-end evaluation with a sample application demonstrated the framework's performance. The evaluation illustrates that, while extending the runtime by only 7 %, energy savings of 24 % can be achieved, showing that substantial energy savings can be attained using machine learning approaches.

Inhaltsverzeichnis

1 Einleitung und Motiovation **7**

2 Energieeffizienz und Machine-Learning – eine thematische Einführung **12**
2.1 Energieeffizienz von Programmen im Hochleistungsrechnen 12
2.1.1 Techniken zur Energiemessung oder -abschätzung 13
2.1.2 Techniken zur Beeinflussung der Energieeffizienz in der Hardware . . . 15
2.1.3 Grundlagen zur Performanceanalyse 17
2.1.4 Regionsbasierte Ansätze zur Erhöhung der Energieeffizienz 19
2.1.5 Andere Ansätze zur Erhöhung der Energieeffizienz 23
2.2 Methoden zur Merkmalsauswahl 23
2.2.1 Merkmalsauswahlmethoden basierend auf der Informationstheorie . . 24
2.2.2 Merkmalsauswahl für stetige Merkmale 29
2.2.3 Andere Verfahren zur Merkmalsauswahl 31
2.3 Machine-Learning mit neuronalen Netzen 33
2.3.1 Neuronale Netze 33
2.3.2 Backpropagation 33
2.3.3 Aktivierungsfunktionen 35

3 Merkmalsauswahl für mehrdimensionale nichtlineare Abhängigkeiten **37**
3.1 Analyse der Problemstellung, Merkmale und Zielgröße 37
3.2 Merkmalsauswahl mit mehrdimensionaler Transinformation für stetige Merkmale 38
3.2.1 Mehrdimensionale Copula-Entropie und mehrdimensionale Transinformation 39
3.2.2 Schätzung der mehrdimensionalen Transinformation basierend auf Copula-Dichte 40
3.3 Normierung 42
3.4 Vergleich von Copula-basierten Maßzahlen mit der klassischen Transinformation und dem Pearson-Korrelationskoeffizienten 43
3.4.1 Deterministische Abhängigkeit zweier Variablen 43
3.4.2 Unabhängigkeit 44
3.5 Vergleich verschiedener Methoden zur Auswahl stetiger Merkmale 45
3.5.1 Erzeugung synthetischer Daten 47
3.5.2 Szenario 1 – fünf relevante Merkmale 48
3.5.3 Szenario 2 – fünf relevante Merkmale, fünf wiederholte Merkmale . . . 51
3.5.4 Schlussfolgerungen aus den Simulationen 52
3.6 Zusammenfassung 52

4 Entwicklung und Umsetzung des Frameworks **53**
4.1 Erweiterungen der READEX Runtime Library 53
4.1.1 Grundlegender Aufbau der READEX Runtime Library 53
4.1.2 Call-Path oder Call-Tree 55
4.1.3 Calibration-Module 57
4.2 Testsystem 58
4.2.1 Architektur 58

4.2.2 Bestimmung des Offsets zur Energiemessung mit RAPL 61
4.3 Verwendete Benchmarks zur Erzeugung der Datengrundlage 61
4.3.1 Datensatz 1: Der Stream-Benchmark 62
4.3.2 Datensatz 2: Eine Sammlung verschiedener Benchmarks 65
4.4 Merkmalsauswahl und Modellgenerierung 69
4.4.1 Datenaufbereitung 69
4.4.2 Merkmalsauswahl Algorithmus 72
4.4.3 Performance-Events anderer Arbeiten zum Vergleich 76
4.4.4 Erzeugen und Validieren eines Modells mithilfe von TensorFlow und Keras 77
4.5 Zusammenfassung 80

5 Evaluierung des Ansatzes 81
5.1 Der Stream-Benchmark 82
5.1.1 Analyse der gewählten Merkmale 82
5.1.2 Ergebnisse des Trainings 87
5.2 Verschiedene Benchmarks 98
5.2.1 Ausgewählte Merkmale 98
5.2.2 Ergebnisse des Trainings 101
5.3 Energieoptimierung einer Anwendung 107

6 Zusammenfassung und Ausblick 110

Literatur 113

Abbildungsverzeichnis 121

Tabellenverzeichnis 123

Quelltextverzeichnis 124

1 Einleitung und Motiovation

Klimaschutz ist eine der Prioritäten der Europäischen Union.[1] Die Bundesregierung hat, um die Fortschritte beim Klimaschutz für Deutschland zu bewerten, den Bruttostromverbrauch als eine Größe gewählt. Ausgegebenes Ziel ist die Reduktion des Bruttostromverbrauchs um 10 % bis 2020, im Vergleich zu 2008. Prognosen aus dem Jahr 2019 stellen fest, dass dieses Ziel wahrscheinlich wurde.[2] Um das nächste Ziel, eine Reduktion des Stromverbrauchs um 25 % bis 2025, zu erreichen, sind Anstrengungen in allen Bereichen der Gesellschaft notwendig.

Dabei spielt auch der IT-Sektor eine wesentliche Rolle: 2018 betrug der Stromverbrauch der IT 11,2 % am gesamten Stromverbrauch (AGEB, 2020, S. 5.1 / S. 5.3). Für alle deutschen Rechenzentren gehen Untersuchungen von einem Energiebedarf von 14 TWh für 2018 aus, was ungefähr einem Viertel des Stromverbrauchs des IT-Sektors entspricht. Die Tendenz ist steigend (Hintemann, 2020).

Da in Rechenzentren oft unterschiedlichste Systeme zusammen stehen, ist es schwierig, den Energieverbrauch von High Performance Computing (HPC) Systemen abzuschätzen. Als konkretes Beispiel kann aber das HPC-System Taurus der TU Dresden dienen: In dem Teil, der für daten- und rechenintensive Anwendung vorgesehen ist, sind mehr als 1 000 einzelne Recheneinheiten, sogenannte Compute-Knoten, miteinander vernetzt.[3] Alle zusammen erzeugen einen Stromverbrauch von ca. 2,7 GWh,[4] wobei der Stromverbrauch des Netzwerks oder der Kühlung nicht mit eingerechnet ist.

Das Besondere an diesem System ist, dass bereits beim Bau des Gebäudes, in dem das System steht, auf Energieeffizienz geachtet wurde. So wurde für das Rechenzentrum der 1. Platz des Deutschen Rechenzentrumspreises in der Kategorie „Energie- und Ressourceneffiziente Rechenzentren“ vergeben (future thinking, 2014). Die Konstruktion des Gebäudes beeinflusst die Energieeffizienz eines HPC-Systems unter anderem über die Möglichkeiten zur effizienten Kühlung oder zur Nachnutzung der Wärme des Systems.

Gleichzeitig lässt sich die Energieeffizienz des Systems selbst steigern. Dazu wird der Energiebedarf zur Lösung eines Problems reduziert, indem die Mechanismen der Hardware selbst besser genutzt werden: Das Leibniz-Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften hat Energie gespart, indem Anwendungen zeigen mussten, dass sie von einer höheren Frequenz des Prozessors profitieren. Ein Programm, das auf dem HPC-System SuperMUC des LRZ zum ersten Mal lief, wurde mit einer reduzierten Frequenz gestartet. Gleichzeitig wurde mit einem Modell vorhergesagt, ob das Programm mit einer höheren Frequenz wesentlich schneller laufen würde. Damit konnten im Januar 2014 6 % Energie gespart werden (Auweter et al., 2014). Es profitieren also nicht alle Programme gleichermaßen von einer hohen Prozessorfrequenz. Dieser Effekt kommt durch unterschiedliche Flaschenhälse in der Hardware oder dem Programm zustande.

Der Ansatz des LRZ kann weiter verbessert werden, indem nicht die gesamte Anwendung, sondern die einzelnen Teile der Anwendung betrachtet werden. Dazu wird ein Programm zunächst anhand seiner Funktionsdefinitionen in unterschiedliche Regionen aufgeteilt. Für

[1] „[...] Wir legen darüber die Prioritäten der Europäischen Union, Klimaschutz und Digitalisierung, [...].“ Ursula von der Leyen, Kommissionspräsidentin der Europäischen Union (Kapern und von der Leyen, 2020).

[2] „Zweiter Fortschrittsbericht zur Energiewende – Die Energie der Zukunft – Berichtsjahr 2017“, veröffentlicht 2019 (Bundesministerium für Wirtschaftund Energie, 2019, S. 65).

[3] Je Compute-Knoten zwei Prozessoren, mit mindestens je 12 Kernen, und bis zu 3,3 GHz (Technische Universität Dresden, 2020).

[4] Basierend auf internen Messungen der Knoten der Inseln 4, 5 und 6 im Jahr 2018.

jede Region wird dann bei einem Testlauf die optimale Prozessorfrequenz ermittelt, indem der Energieverbrauch der Region bei unterschiedlichen Frequenzen gemessen wird. Die Frequenz mit dem niedrigsten Energieverbrauch wird dann für diese Region gespeichert. Wenn das Programm eingesetzt wird, können die Einstellungen für die unterschiedlichen Regionen wieder geladen und angewendet werden. Umgesetzt wurde dieser Ansatz unter anderem in READEX. Dort ist es gelungen, die Einsparungen des LRZ zu übertreffen (Kumaraswamy et al., 2021), wobei der Effekt auf die Laufzeit des Programmes überschaubar bleibt.

Allerdings ist in READEX ein wesentliches Problem aufgetreten: Der Ansatz ist nicht in der Lage, auf unvorhergesehene Situationen zu reagieren, die während der Ausführung des Programmes auftreten. So kann zum Beispiel beim Einsatz des Programmes eine Region auftreten, die bei der Ermittlung der optimalen Frequenzen nicht berücksichtigt wurde. Das kann passieren, wenn beim Testlauf bestimmte Codepfade nicht berücksichtigt wurden oder der Quellcode nach dem Testlauf angepasst wurde. In diesen Fällen muss die Optimierung, also das Bestimmen der optimalen Prozessorfrequenz, erneut durchgeführt werden, um diesen Teil des Programmes mit möglichst geringer Energieaufnahme nutzen zu können. Allerdings kann das Suchen der optimalen Frequenz zeitaufwendig sein. Gelöst wird dieses Problem in dieser Arbeit: Das READEX Framework wird erweitert, um mithilfe von Machine-Learning die Optimierung während der Ausführung des Programmes durchzuführen.

Problemstellungen und Lösungsansätze

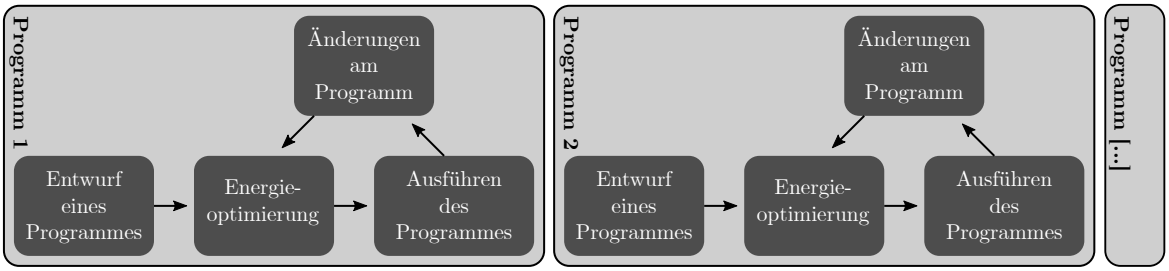


Abbildung 1.1: Klassischer Ablauf einer Energieoptimierung eines Programmes. Die Optimierung findet nach dem Entwurf oder der Bereitstellung eines Programmes statt. Kommt es zu Änderungen an dem Programm, zum Beispiel durch Änderungen am Quellcode, muss es erneut optimiert werden.

Abbildung 1.1 gibt einen Eindruck von dem klassischen Ablauf der Energieoptimierung eines Programmes. Nach dem Entwurf oder der Bereitstellung eines Programmes wird der Energieverbrauch optimiert. Kommt es zu Änderungen am Programm, muss die Optimierung erneut durchgeführt werden.

Mithilfe von Machine-Learning-Methoden kann die Optimierung aber auch während der Ausführung eines Programmes durchgeführt werden, wie in Abbildung 1.2 dargestellt. In dieser Arbeit wird dazu mit einem neuronalen Netz für eine Region ein Modell des Energieverbrauches bei verschiedenen Frequenzen erstellt. Mit diesem Energiemodell kann die optimale Prozessorfrequenz schnell gefunden werden.

Als Merkmale für das Training des Netzes werden auf die Zeit normierte Eventraten prozessorinterner Events verwendet. Die prozessorinternen Events, auch Performance-Events genannt, werden von unterschiedlichen Teilen des Prozessors erzeugt, zum Beispiel wenn eine Instruktion ausgeführt wird oder auf den Arbeitsspeicher zugegriffen werden muss. Sie erlauben damit einen Einblick in die Abläufe in einem Prozessor und können mit sogenannten Performance-Countern gezählt werden.

Performance-Counter können allerdings nicht alle Arten von Events auf einmal zählen, sodass eine Auswahl getroffen werden muss. Das Auswählen der Events von Hand ist aufgrund

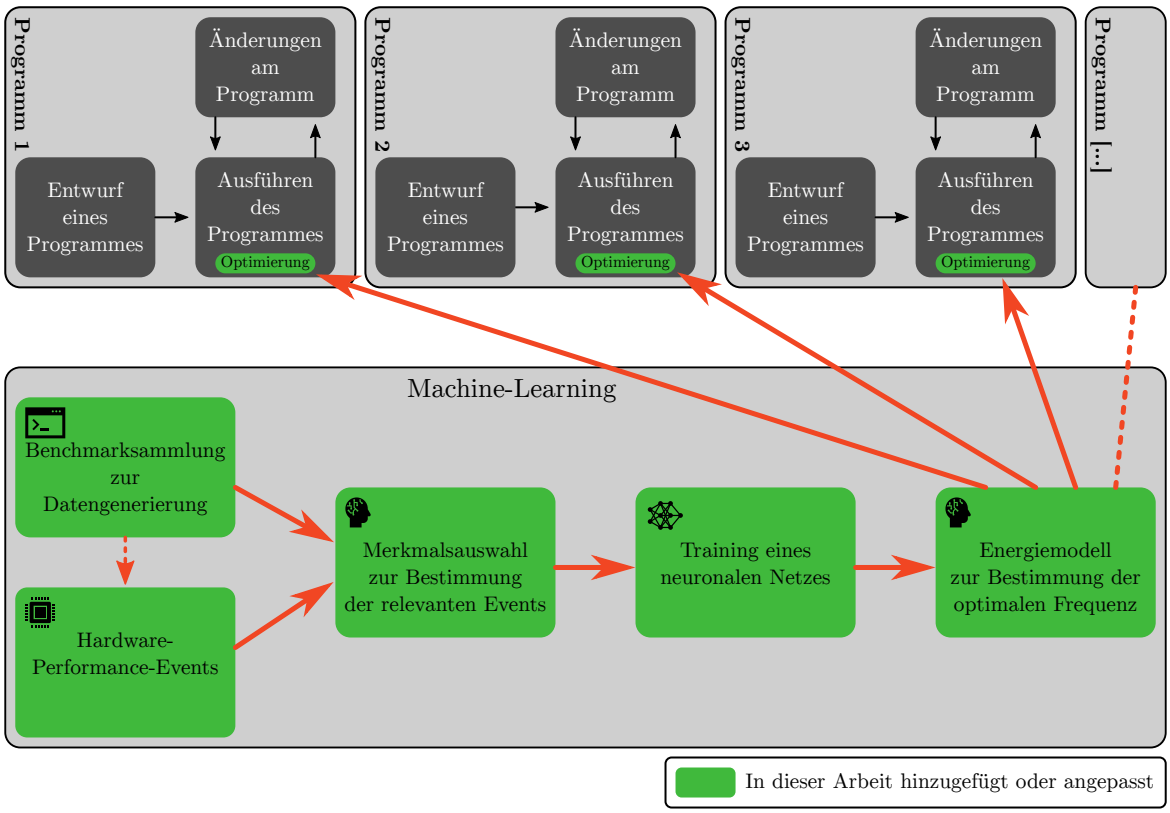


Abbildung 1.2: Ablauf der Optimierung mit dem in dieser Arbeit entworfenen Framework. Zunächst werden mithilfe einer Sammlung von Benchmarks sogenannte Hardware-Performance-Events aufgezeichnet. Aus diesen werden mithilfe eines Merkmalsauswahlverfahrens die relevanten Events bestimmt, um damit ein neuronales Netz zu trainieren. Mit dem daraus resultierenden normierten Energiemodell kann die optimale Frequenz einer Region bestimmt werden. Die Energieoptimierung kann während des Einsatzes des Programmes erfolgen. Änderungen am Programm können ebenfalls direkt berücksichtigt werden.

der Menge sowie einer teils lückenhaften Dokumentation schwierig. Deswegen empfiehlt sich ein automatisiertes Merkmalsauswahlverfahren.

Daraus ergibt sich das zweite Problem dieser Arbeit: Die meisten Verfahren, die im Machine-Learning zur Merkmalsauswahl eingesetzt werden, setzen entweder gaußverteilte Daten voraus oder nutzen diskrete Daten. Beide Eigenschaften treffen auf Eventraten nicht notwendigerweise zu. Um dennoch die relevanten Merkmale auszuwählen, kann der Bereich der Stochastik helfen: In den letzten Jahrzehnten hat sich hier ein breites Verständnis der Abhängigkeitsanalyse zweier oder mehrerer stetiger Zufallsvariablen etabliert (Seth und Príncipe, 2010; Ma, 2021; Blumentritt und Schmid, 2012). Unter der vereinfachenden Annahme, dass Merkmale realisierte Zufallsvariablen sind, lassen sich aus den Abhängigkeitsanalysen Verfahren zur Auswahl von Merkmalen ableiten. Diese können mit neueren Entwicklungen in der Merkmalsauswahl wie den Arbeiten von Brown et al. (2012) und Gocht et al. (2018) kombiniert werden.

Die wesentlichen Beiträge dieser Arbeit lassen sich daraus wie folgt zusammenfassen: 1) Einbettung der aus der Stochastik bekannten Verfahren zur Auswahl von Zufallsvariablen in bekannte Merkmalsauswahlverfahren sowie deren Validierung und Vergleich. 2) Entwicklung eines Frameworks, das unterschiedliche Machine-Learning-Ansätze nutzen kann, um die Energieeffizienz von Anwendungen zu optimieren. 3) Evaluierung des gesamten Frameworks mit unterschiedlichen Merkmalsauswahlverfahren und neuronalen Netzwerken.

Aufbau der Arbeit

Die Arbeit ist dabei folgendermaßen strukturiert: Kapitel 2 gibt einen kurzen Überblick über die Energieeffizienz im Hochleistungsrechnen, bekannte Merkmalsauswahlverfahren sowie das Machine-Learning mit neuronalen Netzwerken. Im Abschnitt über die Energieeffizienz im Hochleistungsrechnen werden sowohl Möglichkeiten zum Messen des Energieverbrauchs und der Performance-Events vorgestellt als auch die Optionen zum Beeinflussen des Energiebedarfs des Prozessors mithilfe der Änderung der Frequenz gezeigt. Darüber hinaus werden einige bereits bekannte Ansätze zur Steigerung der Energieeffizienz von Programmen vorgestellt. Im Rahmen der Merkmalsauswahl werden die wesentlichen Verfahren auf Basis der Transinformation gezeigt. Dazu kommen Verfahren zur Auswahl von Zufallsvariablen auf Basis von Copulas. Am Ende des Abschnittes werden noch weitere bekannte Verfahren zur Merkmalsauswahl mit ihren Vor- und Nachteilen erläutert. Schließlich setzt sich der Abschnitt über das Machine-Learning mit neuronalen Netzwerken kurz mit den mathematischen Grundlagen der neuronalen Netzwerke auseinander.

Kapitel 3 analysiert zunächst die Merkmale und Zielgrößen, die sich aus der Problemstellung ergeben. Dieses Verfahren wird anhand synthetischer Daten mit anderen Verfahren verglichen und analysiert. Dabei werden auch Ansätze zum automatischen Terminieren der Merkmalsauswahlverfahren diskutiert.

Nachdem sich einige geeignete Merkmalsauswahlverfahren herauskristallisiert haben, wird das Framework zum Messen und Trainieren mit realen Daten in Kapitel 4 erläutert. Dabei wird das Aufzeichnen und Verarbeiten der Daten genauso erläutert wie die Umgebung, in der die Experimente durchgeführt wurden, ferner auch das Design und die Anwendung der verwendeten neuronalen Netzwerke.

In Kapitel 5 werden dann die von den Merkmalsauswahlverfahren gewählten Performance-Events ausgewertet. Dazu werden sowohl die Events selbst analysiert als auch der Fehler der damit trainierten neuronalen Netzwerke. Neben den in dieser Arbeit gefundenen Events werden ebenfalls Events analysiert, die sich in anderen Arbeiten als hilfreich herausgestellt haben. Am Ende werden das neuronale Netzwerk und die Menge von Events gewählt, die den geringsten Fehler ergeben haben, und das gesamte Verfahren wird an einer Mini-App evaluiert. Damit wird die Funktionsfähigkeit des Ansatzes in einer realen Umgebung demonstriert.

Abschließend fasst Kapitel 6 die Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf zukünftige Forschungsfragen sowie weitere Verbesserungsmöglichkeiten.

Annahmen und Vereinbarungen

Einige Annahmen und Vereinbarungen ziehen sich durch die gesamte Arbeit. Sie sollen an dieser Stelle zusammengefasst werden.

Annahmen

Annahme 1. *HPC-Programme verhalten sich phasenweise homogen.*

Ein Programm, das sich phasenweise homogen verhält, führt auf allen Kernen die gleiche Arbeit aus. Damit beginnt und endet eine Region auf dem gesamten Knoten zur gleichen Zeit.

Annahme 2. *HPC-Programme belegen einen kompletten Compute-Knoten exklusiv.*

Wenn ein Programm einen Knoten exklusiv belegt, muss es sich die Rechenressourcen nur mit dem Betriebssystem teilen. Dabei wird davon ausgegangen, dass die Ressourcen,

die das Betriebssystem im Vergleich zum Programm benötigt, vernachlässigt werden können. Daraus lässt sich ableiten, dass der gesamte Energieverbrauch eines Compute-Knotens auf das Programm angerechnet werden kann. Gleichzeitig müssen bei der Optimierung des Energieverbrauchs Effekte auf andere Programme nicht berücksichtigt werden.

Annahme 3. *Der Energieverbrauch einer Region kann vorrangig über die Core- und Uncorefrequenz sowie von der Art und Menge der ausgeführten Instruktionen beeinflusst werden.*

Es gibt einige Effekte neben den in Annahme 3 genannten, die den Energieverbrauch eines Systems beeinflussen können. Zwei Beispiele sind die Temperatur und die Operanden. Temperatur: Wird der Prozessor zu warm, gibt es diverse Schutzmechanismen, um den Prozessor vor einer Überhitzung zu schützen. Im Endeffekt führen die Schutzmechanismen zu einer Reduktion der Leistungsaufnahme und damit zu einer Beeinflussung des Energieverbrauchs. Durch Maßnahmen wie eine ausreichende Kühlung kann dem abgeholfen werden. Operanden: Je nach Wert des Operanden verbraucht eine Operation unterschiedlich viel Energie (Schöne et al., 2019). Es lässt sich jedoch annehmen, dass bei einer ausreichenden Menge an Operationen der Effekt im Mittel nicht sichtbar ist.

2 Energieeffizienz und Machine-Learning – eine thematische Einführung

Dieses Kapitel führt in die grundlegenden Techniken und Methoden ein, die in dieser Arbeit verwendet werden, und fasst einige verwandte Arbeiten zusammen.

Begonnen wird dabei in Kapitel 2.1 mit der Energieeffizienz von Programmen im Hochleistungsrechnen. Als Erstes werden dabei Techniken erläutert, um den Energiebedarf eines Compute-Knotens zu ermitteln und zu beeinflussen. Es schließt sich ein kurzer Blick auf die Performanceanalyse an, wobei auch auf Performance-Events und sowie deren Limitierungen eingegangen wird. Basierend auf der Performanceanalyse und den verschiedenen Techniken zum Messen und Beeinflussen des Energiebedarfs haben sich Ansätze gebildet, um die Energieeffizienz von Programmen zu erhöhen. Diese werden insoweit erläutert, als sie eine Grundlage für das hier dargestellte Framework sind. Am Ende des Abschnitts wird die Arbeit im Vergleich zu anderen Ansätzen eingeordnet und abgegrenzt.

Durch die Limitierungen der Performance-Events ist es notwendig, ein Merkmalsauswahlverfahren zu verwenden, um die relevanten Events zu identifizieren. Dazu zeigt und analysiert Kapitel 2.2 verschiedene Merkmalsauswahlverfahren. Dabei liegt der Schwerpunkt auf Verfahren, die die Grundlage für Kapitel 3 darstellen. Zur Abgrenzung werden zwei weitere prominente Verfahren erläutert.

Abgeschlossen wird dieses Kapitel dann mit Kapitel 2.3, das kurz die Grundlagen von neuronalen Netzen zusammenfasst. Diese Netzwerke werden später zum Training und zur Vorhersage eines Energiemodells für verschiedene Programmregionen bei unterschiedlichen Prozessorfrequenzen genutzt. Mit diesem Modell kann dann die optimale Frequenz für eine Region bestimmt werden.

2.1 Energieeffizienz von Programmen im Hochleistungsrechnen

Im Rahmen dieser Arbeit bezieht sich der Begriff *Energieeffizienz* auf den Energieverbrauch zum Ausführen eines bestimmten Teiles eines Programmes oder eines ganzen Programmes. Dabei kann es sich um das Programm selbst wie auch um Programmfunktionen oder andere Regionen innerhalb des Programmes handeln. Ein geringerer Energieverbrauch des entsprechenden Programmteils bedeutet dabei eine höhere Energieeffizienz. Die Funktionalität des Programmes muss dabei erhalten bleiben.

Ziel des Abschnittes ist, unterschiedliche Aspekte des Themas Energieeffizienz im Hochleistungsrechnen zu beleuchten und zugleich die wesentlichen Grundlagen für diese Arbeit zusammenzufassen.

Zunächst werden dazu verschiedene Möglichkeiten gezeigt, den Energieverbrauch eines HPC-Systems zu messen oder abzuschätzen. Diese werden gegeneinander abgewogen, und ein für diese Arbeit zielführender Ansatz wird gewählt. Anschließend werden Technik und Schnittstellen zum Anpassen der Prozessorfrequenzen vorgestellt, mit denen der Energieverbrauch beeinflusst werden kann.

Da die Optimierung der Energieeffizienz auch als Problem der Performanceoptimierung verstanden werden kann, werden anschließend verschiedene Techniken zur Performanceanalyse näher erläutert. Techniken wie Instrumentierung oder Sampling eignen sich als Grundlage für Arbeiten zur Energieeffizienz von Programmen. Ein anderer wesentlicher Aspekt

der Performanceanalyse sind Hardware-Performance-Counter, mit denen prozessorinterne Events gemessen werden können. Diese Events bilden die Datengrundlage dieser Arbeit.

Zusammengeführt werden die Möglichkeiten zum Messen und Beeinflussen des Energieverbrauchs mit den Techniken der Performanceanalyse von den im Anschluss vorgestellten Arbeiten. Die Arbeiten, welche entweder als Grundlage oder Vergleich dienen, werden ausführlich erläutert. Zur Abgrenzung werden kurz weitere Arbeiten zusammengefasst, deren Ergebnisse ebenfalls die Energieeffizienz von Programmen erhöhen können, die aber nicht den hier verwendeten regionsbasierten Ansatz verfolgen.

2.1.1 Techniken zur Energiemessung oder -abschätzung

Um die Energieeffizienz zu verbessern, muss zuerst die Energie gemessen werden. Im HPC kommen dabei diverse Systeme zum Einsatz, die verschiedene Bereiche abdecken und mit unterschiedlicher Genauigkeit arbeiten. Der folgende Abschnitt soll einen kleinen Überblick bieten. Eine ausführliche Analyse unterschiedlicher Messsysteme für den Einsatz im HPC findet sich in der Arbeit von Ilsche (2020).

Energiemessung oder -abschätzung des Prozessor selbst

Die wohl am weitesten verbreitete Technik zur Messung oder Abschätzung des Energieverbrauchs ist Running Average Power Limit (RAPL). Eingeführt wurde sie 2011 von Intel mit der SandyBridge-Architektur. Sie findet inzwischen in den meisten Prozessoren von Intel Verwendung (Desrochers et al., 2016). Externe Messungen an der Intel-SandyBridge-Architektur zeigen, dass RAPL in dieser Prozessorgeneration Modelle verwendet, um den Energieverbrauch des Prozessors zu schätzen (Hackenberg et al., 2013). Seit der Haswell-Architektur wird der reale Energieverbrauch mithilfe eines prozessorinternen Messgeräts gemessen (Hackenberg et al., 2015). Seit der ZEN-Architektur bei AMD wird RAPL auch von AMD unterstützt (AMD Developer Tools Team, 2018, S. 12; AMD, 2017, S. 146). Messungen der Nachfolgerarchitektur ZEN 2 zeigen, dass AMD ein Energiemodell verwendet (Schöne et al., 2021).

Als Schnittstelle zum Auslesen der RAPL-Werte fungieren sogenannte Module Specific Register (MSR). Diese CPU-Register werden dabei mit ungefähr 1 kHz aktualisiert und haben eine Überlaufzeit von ca. 60 Sekunden (Intel, 2019c, S. 14-40 Vol. 3B). Das bedeutet, dass die Register per Software wenigstens einmal pro Minute ausgelesen werden müssen. Um die Leistungsaufnahme eines Programmes über die Zeit darzustellen, müssen die Register entsprechend häufiger ausgelesen werden, was die Programmausführung entsprechend unterbricht und damit Overhead erzeugt. RAPL erlaubt die Bestimmung des Energieverbrauchs der durch den Prozessor verwalteten Komponenten, also des Prozessors selbst und des Arbeitsspeichers (Intel, 2019c, S. 14-38 Vol. 3B).

Lösungen der HPC-Systemintegratoren

Im Gegensatz zu RAPL, das vor allem den Prozessor selbst und den Arbeitsspeicher im Blick hat, gibt es im HPC unterschiedliche Lösungen der Systemintegratoren, die den gesamten Knoten betrachten. Dabei werden in den Knoten integrierte Messgeräte genutzt, die je nach Ausführung sowohl den Energieverbrauch des gesamten Knotens als auch einzelner Komponenten wie Prozessoren, Arbeitsspeicher oder Grafikkarten messen. So nutzt Cray in der XC40 “Cray Advanced Platform Monitoring and Control” (CAPMC), das es erlaubt, Energiemessungen mit einer Auflösung von 10 Hz durchzuführen (Ilsche, 2020, S. 22).

Zeitlich etwas genauere Messungen erlaubt IBM in seinem Power 9 System mit “Automated Measurement of Systems for Temperature and Energy Reporting” (AMESTER). Die Messungen werden mit einer Auflösung von 4 kHz vorgenommen (Ilsche, 2020, S. 23). Es

können neben dem gesamten Knoten auch einzelne Komponenten des Systems gemessen werden.

HDEEM ist als Lösung gemeinsam von Bull/Atos und der TU Dresden entwickelt worden (Ilsche et al., 2018). Bei der Entwicklung wurde besonders Wert auf korrekte Messungen gelegt. Die Messungen des gesamten Knotens werden mit einer Auflösung von 1 kHz bereitgestellt. Darüber hinaus können CPU und DRAM mit einer Auflösung von 100 Hz vermessen werden.

Üblicherweise sind die Energiemessungen über ein externes Interface wie das Intelligent Platform Management Interface (IPMI) zugänglich (Ilsche, 2020, S. 23). Es gibt aber auch Möglichkeiten, die Energiemessungen vom gemessenen System auszulesen, zum Beispiel via PCIe. So ermöglicht es zum Beispiel HDEEM, die Messungen der Leistungsaufnahme über einen bestimmten Zeitraum aufzuzeichnen und die gesammelten Messpunkte am Ende der Messung abzurufen.

Externe Messgeräte

Neben der Messung mittels integrierter Systeme besteht auch die Möglichkeit, externe Geräte zu nutzen, um einzelne Knoten zu vermessen. Ein Beispielgerät ist "Watts Up? Pro", das Messungen mit einer Abtastrate von 1 Hz erlaubt (Ilsche, 2020, S. 20). Bei einer Leistungsaufnahme von über 60 W bietet das Gerät eine Genauigkeit von $\pm 1.5\% + 3 \cdot$ Messauflösung. Unter 60 W sinkt die Genauigkeit.

Andere Geräte gestatten Messungen mit höheren Auflösungen und Genauigkeiten. Das LMG450 zum Beispiel erlaubt das Auslesen der Leistungsaufnahme mit bis zu 20 Hz (Ilsche, 2020, S. 20), der höchste Fehler ist bei einer Signalfrequenz von 20 kHz mit 0,07 % des Messwerts +0,04 % des Messbereiches angegeben (Ilsche, 2020, S. 53). Noch höhere Genauigkeiten und Abtastraten bietet die LMG600-Serie (Ilsche, 2020, S. 20). Hier können die Daten mit einer internen Abtastrate von bis zu 1,21 MHz ausgelesen werden. Die Genauigkeit liegt für Ströme unter 5 A bei 0.015% des Messwerts + 0.01% des Messbereiches.

Eine Kombination zur Anwendung

Die unterschiedlichen vorgestellten Methoden zum Messen von Energie haben diverse Vor- und Nachteile.

So ist bei RAPL, neben der weiten Verbreitung, ein gewichtiger Vorteil, dass die Daten mit einer geringen Latenz verfügbar sind, was für die Bestimmung des Energieverbrauchs eines Programmteils wichtig ist. Es müssen lediglich die entsprechenden Register ausgelesen werden. Dadurch führt das Abrufen während des Programmlaufes zu wenig Overhead. Leider fehlen in RAPL aber meistens Informationen über den Energieverbrauch des Systems als Ganzes, da dies von vielen Herstellern nicht implementiert wird.[1] Damit fehlen auch Informationen über den Energieverbrauch verschiedener Komponenten des Systemes, wie z. B. der Netzwerkkarte oder des Mainboards.

Die Lösungen der HPC-Integratoren messen diese Komponenten mit. Allerdings kommt es hier zu Latenzen beim Zugriff auf die Daten. So wurde im Rahmen dieser Arbiet für den Zugriff auf die Daten von HDEEM via PCIe eine Latenz von wenigstens 5 ms gemessen. Da das IPMI-Interface über das Netzwerk angesteuert wird, ist die Latenz noch höher. Wenn ein Programm oft unterbrochen wird, um für 5 ms Daten auszulesen, führt das zu einem signifikanten Overhead, der nicht mehr vernachlässigt werden kann.

Ähnliche Vor- und Nachteile haben externe Messgeräte. Auch hier führt der Zugriff auf die Daten zu Latenzen. Dazu kommt, dass diese Geräte meist teuer und groß sind, sodass

[1] Mit `MSR_PLATFORM_ENERGY_COUNTER` existiert zwar ein MSR, das die Messung des systemweiten Energieverbrauchs unterstützt, aber: "This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it." (Intel, 2019c, 2-282 Vol 4.).

eine Ausstattung von 1 000 Systemen mit je einem externen Messgerät eher unrealistisch ist (Ilsche, 2020, S. 18).

Ein sinnvoller Kompromiss ist eine Kombination der eingesetzten Methoden. Mit externen Messgeräten oder durch die von HPC-Anbietern integrierten Systeme lässt sich der Unterschied zum mit RAPL ermittelten Energieverbrauch messen. Damit kann ein konstanter Offset geschätzt werden, der zu den Werten der RAPL-Messungen addiert werden kann. Da die Komponenten wie das Mainboard oder die Netzwerkkarte keinen konstanten Energieverbrauch haben, gibt es immer noch einen gewissen Fehler. Allerdings kommt diese Abschätzung der Realität näher, als wenn diese Komponenten einfach ignoriert werden. So ergibt sich für das in Kapitel 4.2 S. 58 beschriebene Testsystem ein Offset von ca. 70 W. Dass sich der Offset nicht ignorieren lässt, zeigt der Vergleich mit den beiden Prozessoren des Testsystems, welche zusammen eine TDP von 240 W aufweisen. Nimmt man vereinfacht an, dass das Gesamtsystem damit eine dauerhafte Leistungsaufnahme von bis zu 310 W aufweist, macht der Offset ca. 22 % der Gesamtleistungsaufnahme des Systems aus und muss in eine Energieoptimierung mit einbezogen werden.

2.1.2 Techniken zur Beeinflussung der Energieeffizienz in der Hardware

Mit einer Leistungsaufnahme von bis zu 240 W stellt der Prozessor dennoch die Komponente mit dem höchsten Stromverbrauch dar. Im READEX-Projekt hat sich gezeigt, dass die Anpassung der Core- und Uncorefrequenz den höchsten Effekt auf die Leistungsaufnahme hat (Gocht et al., 2016). Gleichzeitig impliziert eine geringe Leistungsaufnahme des Prozessors $P_{\text{Prozessor}}$ allein keinen niedrigen Energieverbrauch des Systems E_{System} bei der Ausführung eines Programmes, da für den Energieverbrauch auch die Laufzeit t eine Rolle spielt.

Ein einfaches Modell, bei dem die Änderung der verschiedenen Größen über die Zeit vernachlässigt wird, ist:

$$E_{\text{System}} = (P_{\text{Prozessor}} + P_{\text{DRAM}} + P_{\text{Offset}}) \cdot t.$$

Dabei ist P_{DRAM} die Leistungsaufnahme des Arbeitsspeichers und P_{Offset} die Leistungsaufnahme der restlichen Komponenten im System, und wird hier als konstant angenommen. Die Leistungsaufnahme des Prozessors $P_{\text{Prozessor}}$ kann prinzipiell von zwei Größen beeinflusst werden:[2] von der Spannung V_{DD}, mit der der Prozessor betrieben wird, und der Frequenz f, mit der der Prozessor arbeitet. Da die meisten Prozessoren auf der CMOS-Technologie beruhen, kann deren Leistungsaufnahme über ein Modell für CMOS-Schaltungen abgeschätzt werden (Weste und Harris, 2011, S. 184 ff.):

$$P_{\text{Prozessor}}(V_{DD}, f) = \alpha C V_{DD}^2 f + P_{\text{Kurzschluss}}(V_{DD}, f) + I_{\text{Leckstrom}} V_{DD}. \tag{2.1}$$

Dabei ist α ein Aktivitätsfaktor, der angibt, wie oft ein Schaltkreis zwischen 0 und 1 wechselt, und C die parasitäre Kapazität der verwendeten Transistoren. $P_{\text{Kurzschluss}}$ bezeichnet die Leistungsaufnahme beim Schaltvorgang in dem Moment, in dem die Transistoren schalten. $I_{\text{Leckstrom}}$ gibt den Strom an, der unabhängig davon fließt, ob ein Transistor „offen“ oder „geschlossen“ ist. Eine Reduktion der Spannung hat also wenigstens einen quadratischen Einfluss auf die Leistungsaufnahme und damit auf den Energieverbrauch. Die Frequenz hat wenigstens linearen Einfluss auf die Leistungsaufnahme. Spannung und Frequenz sind aber nicht unabhängig voneinander. Um einen stabilen Betrieb eines Prozessors zu gewährleisten, muss mit der Spannung auch die Frequenz sinken. Umgekehrt kann man sagen, dass bei einer geringeren Frequenz die Spannung gesenkt werden kann.

[2] Bei realen Prozessoren gibt es noch mehr Techniken und Effekte, die die Leistungsaufnahme beeinflussen können. Eine Übersicht geben die Arbeiten von Hackenberg et al. (2015) und Schöne et al. (2019). In dieser Arbeit spielen diese jedoch keine Rolle und werden hier nicht weiter betrachtet.

Gleichzeitig gibt es einen Zusammenhang zwischen Frequenz und Laufzeit:

$$t(f) = \beta(f)\frac{1}{f}.$$

Eine niedrigere Frequenz kann also zu einer längeren Laufzeit eines Programmes führen. Gerade Veränderungen im oberen Frequenzbereich müssen aber nicht immer einen direkten Effekt auf die Laufzeit haben. Besonders wenn der Flaschenhals nicht der Prozessor, sondern zum Beispiel die Bandbreite des Arbeitsspeichers ist, kann die Frequenz ohne Effekt auf t reduziert werden. Das wird durch den Faktor $\beta(f)$ berücksichtigt.

Für den Energieverbrauch des Systems ergibt sich also:

$$E_{\text{System}} = (P_{\text{Prozessor}}(V_{DD}, f) + P_{\text{DRAM}} + P_{\text{Offset}}) \cdot t(f). \tag{2.2}$$

Um E_{System} zu minimieren, muss also eine Frequenz f gefunden werden, bei der die Spannung V_{DD} und damit die Leistungsaufnahme möglichst weit gesenkt werden kann, ohne dass sich die Laufzeit $t(f)$ zu sehr erhöht.

Je nachdem ob die Spannung und die Frequenz eines Prozessorkerns oder des Uncores angepasst werden sollen, spricht man von Dynamic Voltage- and Frequency-Scaling (DVFS) oder Uncore-Frequency-Scaling (UFS). Da DVFS und UFS in dieser Arbeit meistens zusammen auftreten, wird auch von der Core- und Uncorefrequenz gesprochen. Der Hintergrund beider Techniken wird im Folgenden kurz umrissen.

Dynamic Voltage- and Frequency-Scaling

Dynamic Voltage- and Frequency-Scaling (DVFS) ist eine etablierte Technik zum Anpassen der Frequenz und Spannung eines Prozessorkerns. Erste Simulationen dazu wurden bereits 1996 von Weiser et al. (1994) angefertigt. Heute wird DVFS meistens über den Standard Advanced Configuration and Power Interface (ACPI) genutzt (ACPI, 2019, S. 559 ff.). Dieser erlaubt bis zu 16 verschiedene Performance-States (P-States), die vom Prozessor vorgegeben werden können. Jeder P-State definiert ein Paar aus Frequenz und Spannung. Dabei können bestimmte Frequenzen oder Spannungen mehrmals vorkommen, was in der Praxis jedoch selten der Fall ist. Die Einschränkung auf 16 Zustände hat den Nebeneffekt, dass nicht alle vom Prozessor unterstützten Spannungen und Frequenzen angesteuert werden können. Besonders die Turbofrequenzen moderner Intel-Prozessoren werden häufig in einem P-State zusammengefasst.

Diese Limitierung existiert für herstellerspezifische Methoden nicht.[3] Bei modernen Intel-Prozessoren ist es möglich, die Spannung und die Frequenz direkt über prozessorspezifische sogenannte Module-specific Registers (MSRs) zu steuern. Diese können sich aber über Prozessorgenerationen ändern und sind bei anderen Herstellern unter Umständen nicht vorhanden. Deswegen wird in dieser Arbeit der ACPI-Standard verwendet.

Uncore-Frequency-Scaling

Als Uncore werden üblicherweise die Teile eines Multicore-Prozessors bezeichnet, die nicht die Kerne selbst sind. Uncore-Frequency-Scaling (UFS) bezeichnet damit die Technik, mit der die Frequenz und die Spannung des Uncores angepasst werden können. Intel unterstützt eine freie Anpassung des Uncores seit der Haswell-Prozessorgeneration (Hackenberg et al., 2015).[4] Allerdings gibt es für die Anpassung der Spannung und der Frequenz des Uncores keine standardisierten Schnittstellen wie ACPI. Deswegen müssen hier die entsprechenden prozessor- und herstellerspezifischen MSRs genutzt werden.

[3] Details lassen sich in (The kernel development community, 2020b, "`intel_pstate` CPU Performance Scaling Driver") finden.

[4] Die dafür notwendigen Informationen sind inzwischen in den entsprechenden Handbüchern zu finden (Intel, 2019c, S. 2-232 Vol. 4).

2.1.3 Grundlagen zur Performanceanalyse

Die Steigerung der Energieeffizienz eines Programmes kann als Performanceoptimierung verstanden werden. Anstelle der Laufzeit eines Programmes wird auf den Energieverbrauch optimiert. Deshalb können für die Verbesserung der Energieeffizienz auch die Techniken und Werkzeuge [engl. tools] aus dem Bereich der Performanceanalyse verwendet werden.

Prominente Tools, um HPC-Programme selbst untersuchen zu können, sind Tau (Shende und Malony, 2006), ExtraE (Wagner et al., 2017) oder Score-P (Knüpfer et al., 2012) zusammen mit Vampir (Knüpfer et al., 2008; Weber et al., 2019). Für diese Arbeit wird Score-P genutzt, da es mit Plug-ins erweitert werden kann (Schöne et al., 2017). Diese Plug-ins können sowohl zusätzliche Informationen an Score-P geben als auch die von Score-P aufgenommenen Informationen weiter verarbeiten. Score-P unterstützt zwei Techniken zur Untersuchung des Programmes selbst, welche im Folgenden beschrieben werden: Instrumentierung und Sampling.

Um die Interaktion des Programmes mit dem Prozessor untersuchen zu können, wird in der Performanceanalyse oft auf Hardware-Performance-Counter zurückgegriffen. Diese Counter zählen prozessorinterne Events wie Speicherzugriffe. Die Events, oft Performance-Events genannt, können wiederum einen Eindruck davon geben, was im Prozessor selbst vorgeht. Sie eignen sich deshalb als Datengrundlage, mit der später ein neuronales Netzwerk trainiert wird, um die optimale Core- und Uncorefrequenz vorherzusagen.

Sowohl im Rahmen von Instrumentierung und Sampling als auch im Zusammenhang mit Hardware-Performance-Countern wird von Events gesprochen. Diese werden im Folgenden Performance-Events und Programm-Events genannt: Performance-Events sind Events, die vom Prozessor ausgelöst werden, zum Beispiel beim Ausführen einer Floating-Point-Operation oder bei einem Zugriff auf den Speicher. Programm-Events sind Events, die beim Ausführen von instrumentierten Programmen ausgelöst werden, zum Beispiel wenn eine Funktion betreten oder verlassen wird. Für den weiteren Verlauf dieser Arbeit gilt die folgende Vereinbarung: Wenn der Kontext klar ersichtlich ist, wird in dieser Arbeit auf die genaue Spezifikation der Eventart verzichtet. Beide Eventarten werden im Folgenden näher erläutert.

Instrumentierung und Sampling

Um Programme feingranular optimieren zu können, stehen den Tools generell und Score-P im Speziellen zwei Verfahren zur Verfügung: Instrumentierung und Sampling (Ilsche et al., 2015). Instrumentierung kann noch einmal in zwei unterschiedliche Arten aufgeteilt werden: automatische Annotation von Funktionsaufrufen und die Nutzung von Tools-Interfaces. In allen Fällen werden regelmäßig Programm-Events generiert, die von den Tools aufgezeichnet werden.

Die automatische Annotation von Funktionsaufrufen wird von Compilern übernommen. So besitzen zum Beispiel der GCC-, der Intel- oder der LLVM-Compiler die Möglichkeiten, beim Betreten oder Verlassen einer Programmfunktion einen Callback eines Tools aufzurufen (Free Software Foundation, Inc., 2020, Kapitel 3.12; Intel, 2020, S. 244; The Clang Team, 2020). Tools-Interfaces dagegen werden von verschiedenen Programmierparadigmen explizit für die Tools angeboten. So bieten verschiedene, im HPC übliche Standards wie das Message Passing Interface (MPI) (Message Passing Interface Forum, 2015, S. 567 ff.) oder OpenMP (OpenMP Architecture Review Board, 2018, S. 419 ff.) ein solches Interface. Diese Interfaces können unterschiedlich implementiert sein und das eigentliche Programm-Event, wie das Senden einer MPI-Nachricht, kann mit Meta-Informationen wie der Größe der Nachricht angereichert werden. Das Aufzeichnen der wesentlichen Programminformationen funktioniert in beiden Fällen eventbasiert, also z. B. immer wenn eine Funktion aufgerufen wird.

Einen anderen Ansatz verfolgt das Abtasten [engl. sampling]. Dabei gibt es ein taktge-

bendes Signal, das dafür sorgt, dass in regelmäßigen Abständen untersucht wird, welche Programmfunktion gerade ausgeführt wird. Als taktgebendes Signal kann ein Timer oder eine Metrik wie die Anzahl der ausgeführten CPU-Instruktionen genutzt werden (Ilsche et al., 2017). Die gerade ausgeführte Programmfunktion kann dabei zum Beispiel über den Instruction-Pointer bestimmt werden, der nach Abschluss der Ausführung des Programmes den Programmfunktionen zugeordnet wird (Ilsche et al., 2017).

Eine Herausforderung bei beiden Arten der feingranularen Analyse ist, die Analyse nicht zu feingranular zu machen, da sonst der Overhead des Tools zu groß wird. Der Vorteil des Samplings gegenüber der Instrumentierung ist hier, dass der Overhead vor der Untersuchung eines Programmes besser zu bestimmen und zu kontrollieren ist. Bei der Instrumentierung ist vor der ersten Ausführung eines Programmes nicht klar, wie viele Funktionen in welcher Zeit aufgerufen werden. Gerade wenn Funktionen instrumentiert werden, die oft und sehr kurz ausgeführt werden, kann ein großer Overhead entstehen. Mit jedem Aufruf einer Programmfunktion wird auch das Tool aufgerufen. Um dem entgegenzuwirken, werden diese Funktionen gefiltert, das heißt, der Compiler wird angewiesen, bestimmte Funktionen nicht zu instrumentieren.

Ein für diese Arbeit wesentlicher Vorteil der Instrumentierung ist, dass ein Beginn und ein Ende einer Programmfunktion klar bestimmt werden können. Damit können die Regionen mit unterschiedlichen optimalen Frequenzen klar voneinander abgegrenzt werden, was beim Sampling nicht möglich ist.

Hardware-Performance-Counter

Hardware-Performance-Counter, zum Teil auch als Performance-Monitoring-Counter (PMC) oder Performance-Monitoring-Registers bezeichnet, sind fester Bestandteil von Mikroprozessoren. Sie erlauben das Zählen und Filtern von verschiedenen “Performance Monitoring Events” (Intel, 2019c, Vol. 3B 19-1) in Mikroprozessoren. Das können Events wie Cache-Misses oder Speicherzugriffe sein.

Eingeführt wurden die Performance-Events von Intel mit den Intel-Pentium-Prozessoren (Dongarra et al., 2001). Gab es zu Beginn etwa 70 verschiedene Events (Intel, 2019c, 19-258 Vol. 3B), ist deren Anzahl mit der Entwicklung der Prozessoren stetig gewachsen. So ist zum Beispiel mit der Einführung der Multicore-Prozessoren der Uncore hinzugekommen und damit eine Menge von Uncore-Events und Uncore-Countern. Für den Intel-Haswell-Prozessor sind z. B. 364 verschiedene Events in unterschiedlichen Teilen des Prozessors dokumentiert.[5] (Intel, 2015, 2019c)

Die meisten Events haben zusätzlich Filter, sogenannte „Valid Event Masks“ (Umasks), mit denen die Events genauer spezifiziert werden können. Gleichzeitig ist die Anzahl der Counter, die je ein Event mit je einer Umask zählen können, begrenzt. So hat der Caching-Agent an der Schnittstelle zwischen einem einzelnen Kern und dem Ringbus, welcher der Kommunikation im Prozessor dient (Intel, 2015, S. 10), lediglich vier PMCs. Mit diesen können vier verschiedene Events inklusive Umasks gezählt werden. Dem stehen 136 mögliche Events inklusive Umasks gegenüber. Es ist also notwendig eine Auswahl der zu zählenden Events und Umasks zu treffen.

Da die Events selten ohne Umasks verwendet werden, werden in dieser Arbeit mit den Events auch deren Umasks mitgedacht. Ausnahmen werden explizit erwähnt.

PAPI (Dongarra et al., 2001) wurde entwickelt, um ein standardisiertes Set an Events aufzeichnen zu können, das über verschiedene Prozessorarchitekturen die gleichen Informationen liefert. Ein Beispiel sind Floating-Point-Operationen (FLOP). Zusätzlich kann mittels Multiplexing die Beschränkung durch die Anzahl der Counter umgangen werden. Für die fortgeschrittene Anwendung erlaubt PAPI darüber hinaus auch einen einfachen Zugriff auf

[5] Ermittelt mithilfe eines perfmon2-Beispiels auf einem Intel(R) Xeon(R) CPU E5-2680 v3 (Haswell).

die sogenannten nativen, vom Prozessor bereitgestellten Events.

PMCs können aber auch ein Sicherheitsrisiko darstellen. Sie werden über sogenannte Module-specific Registers (MSRs) konfiguriert. Diese MSRs erlauben aber auch andere Einstellungen am Prozessor, wie zum Beispiel das Setzen eines Powerlimits (Intel, 2019c, Vol. 3B 14-37), mit denen die maximal zulässige Leistungsaufnahme des Prozessors spezifiziert werden kann. Deswegen ist der Zugriff auf diese Counter normalerweise gesperrt (The kernel development community, 2020a). Verschiedene Tools wie x86_adapt (Schöne und Molka, 2014) versuchen das Problem zu umgehen, indem sie den Zugriff auf MSRs limitieren. Damit können nur bestimmte vom Administrator festgelegte MSRs mit festgelegten Werten beschrieben werden.

Ein anderes Problem liegt auf der Herstellerseite: Mit neuen Prozessorgenerationen können Counter wegfallen, hinzukommen oder die Bedeutung der einzelnen Counter kann sich ändern (Molka, 2016, S. 121). Zum Teil ist das mit einer sich ändernden Architektur zu begründen, zum Teil bleiben auch PMCs von Fehlern nicht verschont. Damit muss nicht nur für jeden Hersteller, sondern unter Umständen auch für jede Prozessorgeneration neu evaluiert werden, welche Performance-Counter für welche Aussage genutzt werden können.

Dennoch sind PMCs eine hervorragende Möglichkeit, um Anwendungen zu analysieren. So hat Molka (2016) in seiner Arbeit unterschiedliche Events des Prozessorkerns manuell analysiert und verwendet, um verschiedene Leistungsengpässe zu identifizieren. Es kann z. B. festgestellt werden, ob ein Programm durch die Speicherbandbreite limitiert ist, ob eher die Speicherlatenz oder ob doch die Rechenleistung limitierend ist. Daher bieten sich die Daten, die PMCs liefern können, als Datengrundlage für Optimierungen unterschiedlicher Art an, so auch für diese Arbeit.

2.1.4 Regionsbasierte Ansätze zur Erhöhung der Energieeffizienz

Wie bereits geschrieben kann die Optimierung der Energieeffizienz als Performanceoptimierung verstanden werden. Dazu gibt es verschiedene Ansätze, die verfolgt werden können. In dieser Arbeit wird ein regionsbasierter Ansatz verfolgt. Dazu wird eine Region definiert, die optimiert wird.

Diese Region kann aus der gesamten Anwendung oder einem Teil wie einer einzelnen Funktion bestehen. In beiden Fällen sind Anfang und Ende durch die Region gegeben. In den folgenden Arbeiten ist, genau wie in dieser Arbeit, das Optimierungskriterium der Energieverbrauch der Region oder anders ausgedrückt der Energieverbrauch zur Lösung einer Aufgabe. Dies wird auch als Energy to Solution bezeichnet.

Periscope Tuning Framework

Periscope wurde als skalierbares Performanceanalysewerkzeug entwickelt (Benedict et al., 2010). Im Rahmen des AutoTune-Projektes wurde es um einen Autotuningansatz erweitert und in Periscope Tuning Framework (PTF) umbenannt (Miceli et al., 2013). Durch eine Plug-in-API ist es seitdem möglich, PTF einfacher zu erweitern. Neben diversen Plug-ins, die unter anderem die Netzwerkinfrastruktur oder Compiler-Einstellungen optimieren, wurde auch ein Plug-in entwickelt, das die Energieeffizienz einer Anwendung optimiert.

Dieses Plug-in nutzt dabei ein Energiemodell basierend auf Performance-Events (Gerndt et al., 2015, S. 83 ff.). In einem ersten Programmlauf werden bei einer Referenzfrequenz von 2,1 GHz Events wie “instructions per second” oder “L2 cache misses per second” aufgezeichnet. Darüber hinaus wird ermittelt, welche Regionen im Programm für das Optimieren der Corefrequenz geeignet sind. Anschließend wird für die verschiedenen Regionen mithilfe des Energiemodells die optimale Frequenz ermittelt. Im 2. Lauf wird die ermittelte Frequenz angewandt, und im 3. und 4. Lauf wird überprüft, ob die nebenliegenden Frequenzen bessere Ergebnisse liefern. Die Frequenz mit dem niedrigsten Energiebedarf wird dann als optimale

Frequenz gewählt. Nachdem das Optimum ermittelt worden ist, wird dieses als Empfehlung in einer XML-Datei gespeichert. Anschließend wird das Programm neu kompiliert und gegen eine eigene Bibliothek gelinkt, die diese Empfehlung aufnimmt und umsetzt. Die Energieeinsparungen werden mit 40 % gegenüber der schlechtesten Einstellung angegeben (Gerndt et al., 2015, S. 104).

READEX

Im Rahmen von READEX (Schuchart et al., 2017) wurde PTF wiederverwendet und weiterentwickelt. Dazu wurden die Parameter, die zur Optimierung der Energieeffizienz angepasst werden können, erweitert und es wurde auch die Infrastruktur auf Score-P umgestellt. Der Vorteil von Score-P zur Instrumentierung liegt dabei in der breiten Unterstützung unterschiedlicher APIs wie MPI oder OpenMP wie auch verschiedener Programmiersprachen wie C, C++, Fortran oder Python (Gocht et al., 2021).

READEX unterscheidet Design-Time-Analyse (DTA) und Runtime-Analyse (RTA). Die DTA wird nach der Entwicklung eines Programms ausgeführt, um eine optimale Konfiguration für das Programm zu ermitteln. Während des Einsatzes des Programms, also während der Laufzeit (RTA), wird diese Einstellung nur noch angewendet.

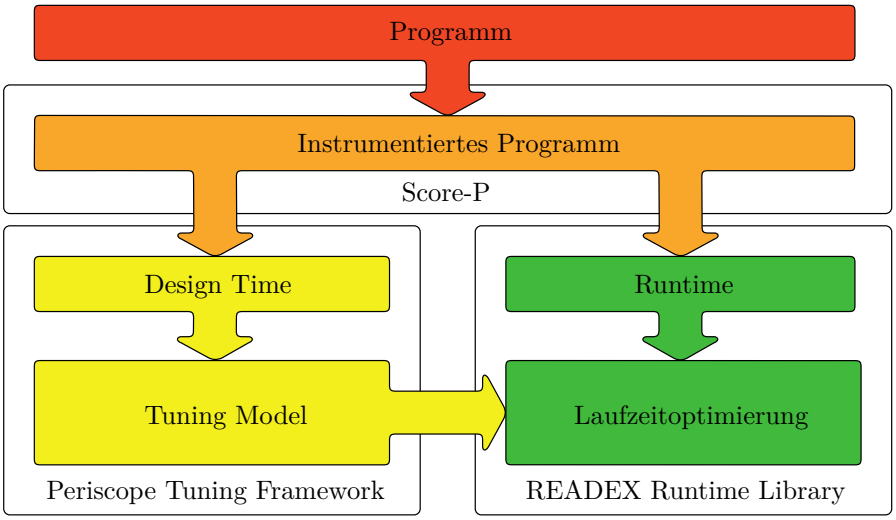


Abbildung 2.1: Vereinfachter Aufbau von READEX. Die Anwendung wird mit Score-P instrumentiert. Zur Design-Time wird mithilfe des Periscope Tuning Frameworks ein Tuning-Model erstellt. Zur Laufzeit wird dieses Model durch die READEX Runtime Library angewendet.

Abbildung 2.1 gibt einen Überblick über die Architektur von READEX. Das Programm muss zuerst mithilfe von Score-P instrumentiert werden. Zur DTA übernimmt PTF die Steuerung und versucht im Zusammenspiel mit Score-P und der READEX Runtime Library (RRL) optimale Einstellungen für die unterschiedlichen Regionen zu finden. Damit eine Region für READEX relevant ist, muss sie wenigstens 100 ms Laufzeit haben. Sobald PTF optimale Einstellungen für alle relevanten Regionen gefunden hat, werden diese in dem Tuning-Model gespeichert. Zur Laufzeit übernimmt die RRL die Steuerung und wendet die gespeicherten Einstellungen an.

Die RRL unterstützt verschiedene sogenannte Parameter-Control-Plug-ins. Diese stellen verschiedene Hard- und Softwareparameter zur Optimierung zur Verfügung und werden genutzt, um die Core- und Uncorefrequenzen sowie die Anzahl der OpenMP-Threads anzupassen. Score-Ps Metrik-Plug-in-Schnittstelle (Schöne, 2017, S. 85 ff.; Schöne et al., 2017) wird genutzt, um Energiemessungen für die verschiedenen Regionen vorzunehmen und die

Messungen an PTF zurückzumelden. Zur Messung der Energie wird, wie in Kapitel 2.1.1 beschrieben, auf RAPL zusammen mit einem konstanten Offset zurückgegriffen.

```
void bar(bool calculate)
{
    if (calculate)
    {
        //Code with more than 100 ms runtime
    } else {
        //Code with 1 ms runtime
    }
}

void foo()
{
    bar(false);
}

int main()
{
    foo();
    bar(true);
}
```

Quelltext 2.1: Beispielcode. Beim ersten Aufruf der Funktion `bar` ist die Laufzeit kleiner als 100 ms, die Funktion ist für READEX nicht von Interesse. Beim zweiten Aufruf der Funktion `bar` ist die Laufzeit größer als 100 ms, die Funktion ist für READEX von Interesse. Beide Aufrufe von `bar` können durch den Call-Path unterschieden werden.

Eine andere Weiterentwicklung im Vergleich zu PTF ist die Verwendung von Call-Paths zur Identifikation von Regionen. Quelltext 2.1 stellt ein Beispiel dar. Es ist möglich, dass die gleiche Funktion in unterschiedlichen Kontexten aufgerufen wird, damit anders verwendet wird und ein anderes Laufzeitverhalten aufweist. Durch die Unterstützung von Call-Paths in READEX können beide in Quelltext 2.1 dargestellten Aufrufe von `bar` unterschieden werden. Da es aber auch möglich ist, dass `bar` von der gleichen Funktion aufgerufen wird, es also keinen Unterschied im Call-Path gibt, unterstützt READEX sogenannte Additional Identifier. Dabei handelt es sich um eine API, die im Rahmen einer manuellen Instrumentierung verwendet werden kann, um zusätzliche Informationen an READEX weiterzureichen. So könnte zum Beispiel der Wert von `calculate` an READEX weitergegeben werden. Diese Information wird dann mit dem Call-Path kombiniert. Diese Kombination wird in READEX Runtime Situation genannt.

Mit dem gesamten READEX-Ansatz ist es gelungen, auf einem Intel-Haswell-System zwischen 7 % und 34 % Energie einzusparen (Kumaraswamy et al., 2021). Im Rahmen dieser Arbeit wurden einige Konzepte und Ideen von READEX wiederverwendet und erweitert.

Neuronale Netze zur Erstellung von Tuning-Modellen

Chadha und Gerndt (2019) erweitern READEX, indem sie ein neuronales Netz als Modell nutzen, um die optimale Core- und Uncorefrequenz vorherzusagen. Als Datenbasis verwenden sie Performance-Events sowie den Energieverbrauch. Die gemessenen Performance-Events werden auf die Menge der Events reduziert, die durch PAPI standardisiert wurden. Damit wird der Messraum klein gehalten.

Für das Training wird ein Teil der gemessenen Events verwendet. Die Auswahl wird mithilfe eines Ansatzes aus einer vorhergehenden Arbeit von Chadha et al. (2017) getroffen, die wiederum auf einer Arbeit von Walker et al. (2017) basiert.

Zunächst wird für alle Events eine multiple lineare Regression in Bezug auf einen normierten Energieverbrauch berechnet. Dazu wird der Energieverbrauch auf den Energieverbrauch bei 2,0 GHz Core- und 1,5 GHz Uncorefrequenz normiert. Mithilfe des Bestimmtheitsmaßes der Regression wird das Event gewählt, welches das Bestimmtheitsmaß am meisten verbessert. Danach wird zwischen allen bereits gewählten Events wieder eine multiple lineare Regression durchgeführt und das Bestimmtheitsmaß ermittelt. Mithilfe des Variance-Inflation-Factors VIF, der sich aus dem Bestimmtheitsmaß R^2 berechnet,

$$\text{VIF} = \frac{1}{1 - R^2}, \tag{2.3}$$

wird entschieden, ob das Event zur Menge der gewählten Events hinzugefügt wird oder ob der Algorithmus abgebrochen wird. Wenn der Algorithmus nicht abgebrochen wird, wird das nächste Event nach dem angegebenen Verfahren gewählt. Damit soll die Multikollinearität zwischen Merkmalen möglichst gering gehalten werden. Chadha und Gerndt (2019) ermittelten so sieben relevante Performance-Events.

Anschließend wird mit dem Energieverbrauch und den ausgewählten Events ein zweilagiges neuronales Netz trainiert. Als Eingabe werden die ausgewählten und auf die Laufzeit einer Funktion normierten Performance-Events sowie die Core- und Uncorefrequenz verwendet, als Ausgabe der normierte Energieverbrauch bei den gegebenen Frequenzen.

Zur Evaluation werden verschiedene Benchmarks verwendet und mittels Kreuzvalidierungsverfahren wird der Fehler des Modells bestimmt. Abschließend wird ein Tuning-Model generiert, das die optimalen Frequenzen speichert und die Energieeinsparungen ermittelt. Dabei wird von Einsparungen zwischen 5 % und 10 % berichtet.

Die Arbeit von Chadha und Gerndt (2019) ist eine konsequente Fortsetzung der Ideen von READEX. Durch die Verwendung eines neuronalen Netzes zum Ermitteln der optimalen Konfiguration kann der Aufwand der Desing-Time-Analyse (DTA) signifikant reduziert werden. Es müssen nicht mehr verschiedene Einstellungen untersucht werden, sondern nur die verschiedenen Performance-Events gemessen werden. Dadurch reduziert sich die Anzahl der nötigen Wiederholungen einer Region.

Der vorgestellte Ansatz benötigt zwar weiterhin eine DTA. Es ist aber denkbar, das neuronale Modell auch zur Laufzeit auszuwerten und so auf die DTA verzichten zu können. Gleichzeitig ist es möglich, mehr als die standardisierten PAPI-Performance-Events zu vermessen, indem die gemessenen Events nicht mit jedem Programmlauf gewechselt werden, sondern mit jeder Ausführung einer Region. Abschließend stellt sich die Frage, ob es Merkmalsauswahlverfahren gibt, die Performance-Events finden, mit deren Hilfe der Energieverbrauch einer Region bei verschiedenen Frequenzen besser vorhergesagt werden kann. An diesen drei Punkten setzt diese Arbeit an.

Auf Reinforcement-Learning basierte Optimierung

Eine andere Weiterentwicklung von READEX hat der Author dieser Arbeit in Gocht et al. (2019) vorgenommen. Der Algorithmus nimmt sich Q-Learning zum Vorbild (Watkins, 1989). Im Gegensatz zu klassischen Q-Learning-Problemen gibt es aber keinen finalen Zustand.

Die verschiedenen durch den Prozessor vorgegebenen Core- und Uncorefrequenzen bilden einen diskreten und endlichen Zustandsraum S_t. Jedem Zustand können mehrere Aktionen A_t zugeordnet werden, bei denen die Frequenz nach oben oder unten geändert werden kann. Aus den normierten Differenzen des Energieverbrauchs einer Region wird dann die Belohnung der Entscheidung abgeleitet R_{t+1} und der Zustands-Aktions-Raum Q entsprechend aktualisiert:

$$Q_{new}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A_t} (Q(S_{t+1}, a)) - Q(S_t, A_t) \right].$$

α ist dabei die Lernrate, während γ eine Gewichtung der Entscheidungen, die im nächsten Schritt zur Verfügung stehen, vornimmt. Um lokale Minima zu vermeiden, wird in einem

gewissen Teil der Fälle die Entscheidung über den nächsten Schritt nicht anhand von Q getroffen, sondern zufällig.

Mit diesem Ansatz ist es mir gelungen, bis zu 15 % Energie zu sparen. Der Ansatz ist in der Lage, ein Optimum zur Laufzeit zu finden und sich auf Änderungen des Programmes einzustellen. Allerdings sind einige Iterationen notwendig, um den optimalen Zustand zu finden. Eine Region muss also mehrmals auftreten, um eine optimale Einstellung zu finden. Bei Programmen mit lang andauernden Regionen und wenigen Wiederholungen dieser kann dieser Ansatz die Energieeffizienz verringern, da das Programm lange mit einer nicht optimalen Einstellung läuft.

2.1.5 Andere Ansätze zur Erhöhung der Energieeffizienz

Neben den bereits beschriebenen regionsbasierten Ansätzen gibt es noch weitere Ansätze, die andere Effekte ausnutzen. So lässt sich bei Anwendungen, die ihre Arbeit nicht gleichmäßig verteilen, die Energieeffizienz steigern, indem die Prozessorkerne, auf denen Prozesse oder Threads laufen, die eher fertig sind als andere, verlangsamt werden. Im Optimalfall erreichen so alle Beteiligten gleichzeitig einen Synchronisationspunkt und der Energieverbrauch sinkt. Beispiele dafür sind in den Arbeiten von Schöne (2017, S. 110 ff.) und Rountree et al. (2009) zu finden. Da es aber Techniken gibt, die es Anwendungen erlauben, ihre Arbeitslast selbstständig zu verteilen (Lieber et al., 2012), ist dieser Ansatz nicht immer zielführend.

Weiter gibt es sogenannte Governor, die versuchen, den optimalen Arbeitspunkt mithilfe von Performance-Events zu ermitteln. Governor werden vom Betriebssystem genutzt, um verschiedene energierelevante Einstellungen, wie z. B. Prozessorfrequenzen, zu verwalten. Ein Beispiel ist der Green Governor von Spiliopoulos et al. (2011). Jedes Mal, wenn der Green Governor aufgerufen wird, werden verschiedene Performance-Events gemessen und mithilfe eines Modells wird der Energieverbrauch bei verschiedenen Frequenzen abgeschätzt. Anhand dieser Information wird versucht, eine Aussage für das nächste Intervall zu treffen, bis der Governor erneut aufgerufen wird. Gegenüber diesem intervallbasierten Ansatz, bei dem eine Entscheidung auf Basis der Vergangenheit getroffen wird, ist ein regionsbasierter Ansatz im Vorteil, da das Optimum der kommenden Region bereits am Beginn dieser geladen werden kann.

2.2 Methoden zur Merkmalsauswahl

Ein Merkmal (engl. feature) ist im Machine-Learning eine Eingabevariable. Je nach Einsatzgebiet kann es sich dabei z. B. um einen Pixel aus einem Bild, einen Buchstaben oder einen Messwert handeln. Eine Reduzierung der Merkmale (engl. feature reduction) kann sinnvoll sein, um das Lernergebnis zu verbessern oder den Lernprozess zu beschleunigen.

Dabei wird die Merkmalsreduktion grundsätzlich in die Merkmalsauswahl (engl. feature selection) und die Merkmalsextraktion (engl. feature extraction) unterschieden (Alpaydin, 2014, Chapter 6). Der Themenbereich der Merkmalsextraktion beschäftigt sich mit der Reduzierung der Eingabedimension durch Erhöhen der Informationsdichte. Dabei werden die ursprünglichen Merkmale durch geeignete Verfahren transformiert und zu neuen Merkmalen kombiniert. Für den Prozess des Lernens und der Inferenz werden immer noch alle ursprünglichen Merkmale benötigt, um diese in die neuen Merkmale zu transformieren. Der Vorteil ist aber, dass keine oder wenig Informationen aus dem ursprünglichen Merkmalsraum verloren gehen. Dazu werden Verfahren wie Principal-Component-Analysis (Jolliffe, 2002, S. 111) verwendet.

Dem steht die Merkmalsauswahl gegenüber, bei der aus der Menge von möglichen Merkmalen nur die relevantesten ausgewählt werden. Nach der Auswahl der relevanten Merkmale wird sowohl während des Trainings als auch während der Inferenz nur mit den ausgewählten

Merkmalen gearbeitet. Dabei gehen Informationen verloren, allerdings können damit andere Randbedingungen erfüllt werden.

Die in Kapitel 2.1.3 beschriebenen Performance-Counter sind ein konkretes Beispiel für eine solche Randbedingung. Wie schon diskutiert, können Performance-Counter nur eine beschränkte Anzahl an Performance-Events zählen. Um die für diese Arbeit relevanten Events anzuwählen, wird ein Merkmalsauswahlverfahren benötigt, das mit diesen Events umgehen kann.

Embedded-, Wrapper- und Filtermethoden

Die Merkmalsauswahl wird in drei Teilbereiche unterteilt: Embedded-, Wrapper- und Filtermethoden (Blum und Langley, 1997). Die Idee hinter Embedded-Methoden ist, Merkmale hinzuzufügen und wieder zu entfernen, um so den Fehler des gesamten Lernprozesses zu minimieren. Dazu werden die Daten, wie im Machine-Learning üblich, in ein Trainings- und ein Evaluationsset geteilt. Jetzt wird eine Menge von Merkmalen gewählt, das Modell trainiert und der Fehler evaluiert. Das wird wiederholt, bis der Fehler des Evaluationssets ausreichend klein ist. Für Pipelines mit einer langen Laufzeit und vielen Merkmalen kann dieses Verfahren problematisch sein.

Wrapper-Methoden verwenden ebenfalls Machine-Learning-Verfahren (Blum und Langley, 1997). Im Gegensatz zu den Embedded-Methoden werden die Merkmale aber ausgewählt, bevor der eigentliche Lernprozess beginnt. Ein Beispiel wäre die Merkmalsauswahl mithilfe von linearer Regression, der sich das Training mit neuronalen Netzwerken anschließt. Die Regression kann genutzt werden, um Abhängigkeiten der Merkmale untereinander zu erkennen und damit die Merkmale auszuwählen (Chadha et al., 2017; Walker et al., 2017). Diese Merkmale können dann für das Training des Netzwerkes verwendet werden. Dabei limitieren die eingesetzten Machine-Learning-Methoden allerdings die Eigenschaften der Merkmale: Wenn als Beispiel lineare Regression verwendet wird, um die Merkmale auszuwählen, können damit die Fähigkeiten neuronaler Netzwerke, Nichtlinearitäten abzubilden, nur bedingt ausgenutzt werden.

Im Kontrast dazu stehen Filtermethoden, bei denen versucht wird, den Zusammenhang zwischen Merkmalen und der vorherzusagenden Größe abzuschätzen (Blum und Langley, 1997). Besonders Ansätze aus der Informationstheorie, die versuchen, die Information eines Merkmals gegenüber der vorherzusagenden Größe abzuschätzen, sind hier zu erwähnen. Andere Ansätze nutzen die Korrelation eines Merkmals mit der vorherzusagenden Größe. Je nach Algorithmus wird dabei auch versucht, die Anzahl der zu wählenden Merkmale automatisch zu bestimmen und die Abhängigkeit verschiedener Merkmale untereinander zu berücksichtigen.

Der Vorteil von Filtermethoden gegenüber Embedded- und Wrapper-Methoden besteht in der Unabhängigkeit gegenüber dem späteren Machine-Learning-Modell. Sie fügen dem Lernprozess wenig Komplexität hinzu, da sie die Anzahl der Hyperparameter, die beim Training des Machine-Learning-Modells beachtet werden müssen, nicht wesentlich erhöhen. In den folgenden Abschnitten wird eine Auswahl an Filtermethoden daher näher beschrieben.

2.2.1 Merkmalsauswahlmethoden basierend auf der Informationstheorie

Das Ziel aller Merkmalsauswahlmethoden ist, die Merkmale zu finden, die am besten für die Vorhersage einer Zielgröße geeignet sind. Anders ausgedrückt sollen genau die Merkmale gewählt werden, die die meisten Informationen über die Zielmetrik haben. Um diesen Zusammenhang zu bestimmen, bieten sich Verfahren an, die aus der Informationstheorie bekannt sind.

Die im Folgenden beschriebenen Methoden bilden eine theoretische Grundlage dieser Arbeit.

Transinformation

Die Transinformation (engl. mutual information) wird genutzt, um Störungen auf einem Kommunikationskanal einzuschätzen (Gottwald, 1993, S. 202 ff.). Eine hohe Transinformation $I(X_k, Y)$ zwischen einem Sender X_k und einem Empfänger Y sagt aus, dass ein Zeichen x_v, das von X_k nach Y gesendet wird, mit einer hohen Wahrscheinlichkeit als y_v erkannt wird. Eine niedrige Transinformation impliziert eine hohe Wahrscheinlichkeit, dass das Zeichen durch eine Störung beim Empfänger als ein anderes Zeichen y_u erkannt wird.

Die Transinformation basiert dabei auf der Entropie H, einer Maßzahl für den mittleren Informationsgehalt eines Zeichens (Gottwald, 1993, S. 191; Brown et al., 2012, S. 29). Für ein diskretes Signal ist sie definiert als:

$$H(X_k) = -\sum_{x_v \in X_k} p(x_v) \log p(x_v),$$

während $p(x_v)$ die Wahrscheinlichkeit des Zeichens x_v in X_k bezeichnet. Es gilt:

$$\sum_{x_v \in X_k} p(x_v) = 1 \tag{2.4}$$

Die Entropie des Senders X_k, bezogen auf den Empfänger Y, ist definiert als:

$$H(X_k|Y) = -\sum_{y_v \in Y} p(y_v) \sum_{x_v \in X_k} p(x_v|y_v) \log p(x_v|y_v),$$

wobei $p(x_v|y_v)$ die bedingte Wahrscheinlichkeit bezeichnet. Die Transinformation $I(X_k, Y)$ zwischen X_k und Y ergibt sich aus der Differenz zwischen der Entropie und der bedingten Entropie:

$$I(X_k, Y) = H(X_k) - H(X_k|Y) \tag{2.5}$$

Hilfreiche Eigenschaften der Transinformation für die Merkmalsauswahl sind $I(X_k, Y) \in [0, 1]$ und $I(X_k, Y) = 0$, wenn X_k und Y unabhängig sind. Darüber hinaus gilt $I(X_k, Y) = I(Y, X_k)$ (Brown et al., 2012, S. 29).

Grafisch lässt sich die Transinformation im Zusammenhang mit der Entropie wie in Abbildung 2.2 zu sehen darstellen. Für das Verständnis im Rahmen der Informationstheorie ist Abbildung 2.2a hilfreich. Im Bereich der Merkmalsauswahl wird aber das Venn-Diagramm in Abbildung 2.2b bevorzugt, weswegen im weiteren Verlauf darauf zurückgegriffen wird.

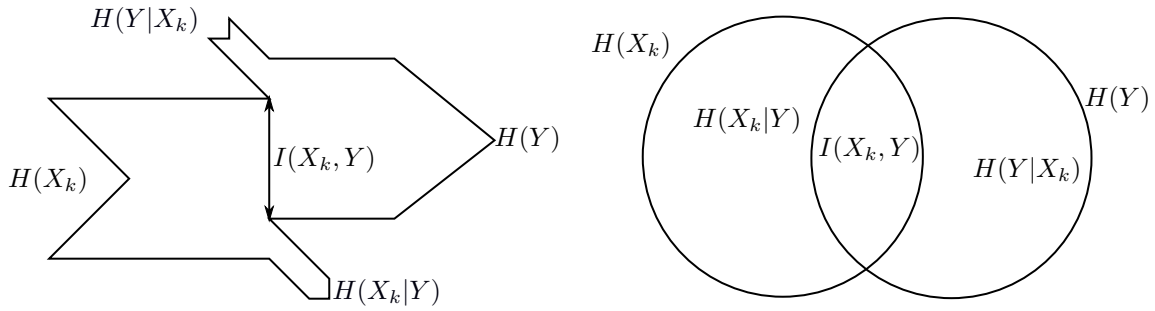
Um die Transinformation aus der Informationstheorie in die Merkmalsauswahl zu übertragen, wird der Sender als Merkmal und der Empfänger als Zielgröße interpretiert. Die Wahrscheinlichkeiten können durch Zählhäufigkeiten geschätzt werden.[6] $p(x_v)$ und $p(x_v|y_v)$ können also über das Verhältnis der Häufigkeit eines Zeichens zur Häufigkeit aller Zeichen bestimmt werden.

Als Erstes wurde dieser Ansatz von Lewis (1992) verfolgt. Er verwendete die Transinformation als Maßzahl für die Qualität eines Merkmals J_{MI}:

$$J_{MI}(X_k) = I(X_k, Y) = H(X_k) - H(X_k|Y) \tag{2.6}$$

Daraus lässt sich ein Algorithmus ableiten. Sei $X = \{X_1, X_2, X_3, ..., X_K\}$ eine Menge von K Merkmalen und l die Anzahl der informativsten Merkmale, wobei $l \leq K$. Dann kann ein Merkmal wie folgt der Menge der gewählten Merkmale S hinzugefügt werden:

[6] Unter der Annahme, dass der datengenerierende Prozess über die Zeit stabil bleibt und unterschiedliche Beobachtungen zu unterschiedlichen Ergebnissen führen.



(a) Darstellung als Informationskanal. $H(X_k)$ und $H(Y)$ repräsentieren die Entropie des Sendesignals X_k und des Empfangssignals Y. $H(Y|X_k)$ kann als Störinformation verstanden werden, während $H(X_k|Y)$ als Informationsverlust betrachtet werden kann. Die Darstellung ist inspiriert von Akribix (2006).

(b) Darstellung als Venn-Diagramm. Die Kreise repräsentieren die Signale X_k bzw. Y. Der gesamte Kreis stellt die Entropie $H(X_k)$ bzw. $H(Y)$ dar. Die Transinformation $I(X_k, Y)$ ergibt sich aus der Schnittmenge.

Abbildung 2.2: Darstellung der Entropie $H(X_k)$ und $H(Y)$ und der Transinformation $I(X, Y)$ als Informationskanal und als Venn-Diagramm.

Algorithmus 1 Algorithmus zur Merkmalsauswahl mittels Transinformation

1: $S \leftarrow \emptyset$
2: **while** $|S| < l$ **do**
3: $\quad X_{\text{tmp}} \leftarrow \text{argmax}_{X_k \in X \setminus S} \left(J_{MI}(X_k) \right)$
4: $\quad S \leftarrow S \cup X_{\text{tmp}}$
5: **end while**

Mehrdimensionale Transinformation

Der Hauptkritikpunkt an der Transinformation als Maßzahl kommt bei Interaktionen zwischen verschiedenen Merkmalen zum Tragen. Angenommen zwei Merkmale haben die gleichen Informationen über die vorherzusagende Zielgröße Y, würden beide Merkmale vom Algorithmus 1 gewählt. Da beide Merkmale sich gleich verhalten, wäre aber für eine Vorhersage für Y ein Merkmal ausreichend.

Über die Zeit wurden verschiedene Ansätze entwickelt, um dieses Problem zu umgehen. In einer ausführlichen Analyse haben Brown et al. (2012) die Merkmalsauswahl mittels Transinformation sowie verschiedene Weiterentwicklungen analysiert. Dazu gehören unter anderem Mutual Information Feature Selection (MIFS) (Battiti, 1994), die Minimum-Redundency Maximum-Relevance (MRMR) (Peng et al., 2005) und die mehrdimensionale Transinformation (engl. Joint Mutual Information, JMI, manchmal auch als verbundene Transinformation oder gemeinsame Transinformation bezeichnet) (Yang und Moody, 1999). Allen drei Verfahren gemein ist, dass sie die bereits gewählten Merkmale berücksichtigen und damit versuchen, Redundanzen zwischen den Merkmalen zu vermeiden.

Brown et al. (2012) analysieren die verschiedenen Merkmalsauswahlverfahren dabei auf die folgenden Kriterien hin:

- Stabilität
- Gleichheit der Kriterien untereinander

- Verhalten bei kleinen Datensätzen
- Verhältnis von Stabilität zu Genauigkeit

Dabei ergibt sich, dass JMI stabiler ist als MIFS oder MRMR und ein besseres Verhältnis von Stabilität zu Genauigkeit aufweist. Darüber hinaus zeigt sich, dass JMI auch mit kleinen Datensätzen umgehen kann. In einer anschließenden Evaluierung mit Daten aus zwei Benchmarks der NIPS Feature Selection Challenge (Guyon et al., 2003, 2004) zeigt sich allerdings, dass die MRMR in der Praxis die Ergebnisse des JMI übertreffen kann. So wird in einem von zwei betrachteten Benchmarks der JMI von der MRMR übertroffen, während das Ergebnis im zweiten Benchmark genau andersherum ist. Damit sind beide Verfahren für diese Arbeit relevant und werden im Weiteren genauer betrachtet.

Die MRMR nutzt die Transinformation zwischen dem gerade betrachteten Merkmal X_k und der Zielgröße Y sowie die Transinformation zwischen dem gerade betrachteten Merkmal X_k und den bereits gewählten Merkmalen X_j.

$$J_{MRMR}(X_k) = I(X_k, Y) - \frac{1}{|S|} \sum_{X_j \in S} I(X_k, X_j). \tag{2.7}$$

Die Menge S enthält die gewählten Merkmale S, $|S|$ ist die Anzahl der gewählten Merkmale. MRMR setzt der Transinformation zwischen X_k und Y die durchschnittliche Transinformation zwischen X_k und allen bereits gewählten Merkmalen entgegen.

Die JMI entwickelt dieses Konzept weiter:

$$I(X_k X_j, Y) = \sum_{x_k \in X_k} \sum_{x_j \in X_j} \sum_{y \in Y} p(x_k, x_j, y) \frac{p(x_k, x_j, y)}{p(x_k, x_j) p(y)} \tag{2.8}$$

$$J_{JMI}(X_k) = \sum_{X_j \in S} I(X_k X_j, Y) \tag{2.9}$$

Für das Verständnis sind hier die Betrachtung des Venn-Diagramms in Abbildung 2.3 und folgende Transformation hilfreich (Brown et al., 2012):

$$I(X_k, X_j | Y) = \sum_{y \in Y} p(y) \sum_{x_k \in X_k} \sum_{x_j \in X_j} p(x_k, x_j | y) log \frac{p(x_k, x_j | y)}{p(x_k | y) p(x_j | y)} \tag{2.10}$$

$$J_{JMI}(X_k) = I(X_k, Y) - \frac{1}{|S|} \sum_{X_j \in S} \left[I(X_k, X_j) - I(X_k, X_j | Y) \right]. \tag{2.11}$$

Die JMI setzt der Transinformation zwischen X_k und Y also die Information entgegen, die über Y bereits abgedeckt ist. Anders formuliert versucht die JMI also nur die Informationen zu berücksichtigen, die in den bereits gewählten Merkmalen noch nicht vorhanden sind.

Es ist offensichtlich, dass das Konzept der JMI auf mehr als zwei Merkmale und eine Zielgröße erweitert werden könnte. Genau das haben Brown et al. (2012) auch versucht. Allerdings stellen sie fest, dass beim Erweitern auf mehr Merkmale die Datenbasis sehr schnell zum Problem wird. Um die bedingte Transinformation zu bestimmen, werden die gemeinsamen Auftretenshäufigkeiten mehrerer Zeichen benötigt, wie in Gleichung 2.10 zu sehen. Bei zu vielen Merkmalen wird hier die Datenbasis für bestimmte Zeichenkombinationen schnell zu dünn, um belastbare Aussagen treffen zu können.

Um JMI oder MRMR verwenden zu können, muss anschließend der Algorithmus 1 angepasst werden:

Automatische Merkmalsauswahl mithilfe der historischen mehrdimensionalen Transinformation

Um Merkmale mit Algorithmus 2 auszuwählen, ist es immer noch notwendig, die optimale Anzahl der auszuwählenden Merkmale, also l, zu bestimmen. Der traditionelle Ansatz ist, das

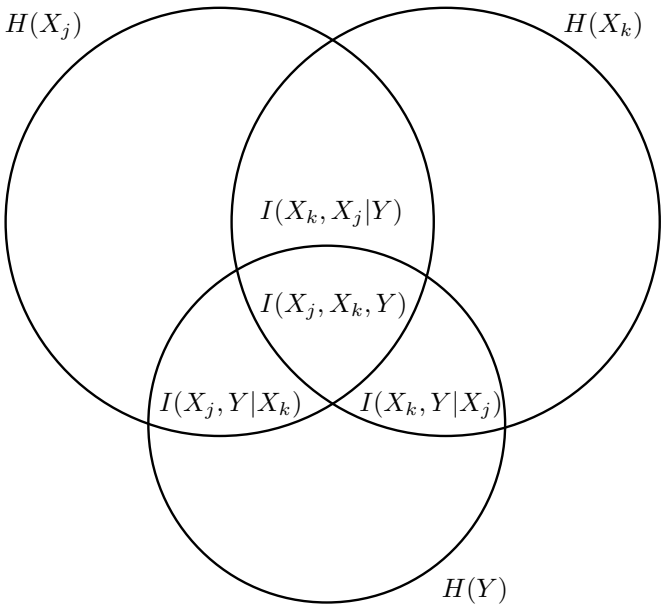


Abbildung 2.3: Venn-Diagramm für die Transinformation unter Berücksichtigung von zwei Merkmalen X_k, X_j und der Zielgröße Y. Vergleichbar zu Abbildung 2.2b repräsentieren die einzelnen Kreise die Entropie $H(X_k)$, $H(X_j)$ bzw. $H(Y)$. Aus den Schnittmengen ergeben sich die bedingte Transinformation $I(X_k, X_j|Y)$, $I(X_k, Y|X_j)$ bzw. $I(X_j, Y|X_k)$ sowie eine Variante der dreidimensionalen Transinformation $I(X_k, X_j, Y)$ (Timme et al., 2014).

Algorithmus 2 Algorithmus zur Merkmalsauswahl mittels JMI oder MRMR $J_{MRMR/JMI}$

1: $X_{\text{tmp}} \leftarrow \text{argmax}_{X_k \in X} \left(J_{MI}(X_k) \right)$
2: $S \leftarrow X_{\text{tmp}}$
3: **while** $|S| < l$ **do**
4: $\quad X_{\text{tmp}} \leftarrow \text{argmax}_{X_k \in X \setminus S} \left(J_{MRMR/JMI}(X_k) \right)$
5: $\quad S \leftarrow S \cup X_{\text{tmp}}$
6: **end while**

Ergebnis des der Merkmalsauswahl nachgelagerten Lernprozesses mit verschiedenen Werten von l zu evaluieren, wie in Brown et al. (2012) zu sehen.

Das kann, je nach Menge der Merkmale und Komplexität des Lernprozesses, sehr aufwendig werden. Eine Abbruchbedingung kann den Aufwand hier verringern und unter Umständen zu besseren Ergebnissen führen, wie der Author dieser Arbeit in Gocht et al. (2018) gezeigt hat. Die JMI kann dazu zur historischen JMI (HJMI) umgeformt werden:

$$J_{HJMI}(X_k, S) = J_H + I(X_k, Y) - \frac{\sum_{X_j \in S} \left(I(X_k, X_j) - I(X_k, X_j|Y) \right)}{|S|}. \tag{2.12}$$

Dabei enthält J_H das Ergebnis von J_{HJMI} für das zuletzt gewählte Merkmal. J_H enthält also historische Informationen.

Aus der Differenz von J_H und J_{HJMI} lässt sich so ableiten, um wie viel die Gesamtinformation angewachsen ist und wie das im Verhältnis zu der bereits vorhandenen Information steht.

$$\delta_{HJMI} > \frac{I(X_k, Y) - \frac{\sum_{X_j \in S} \left(I(X_k, X_j) - I(X_k, X_j|Y) \right)}{|S|}}{J_H}. \tag{2.13}$$

Algorithmus 3 Algorithmus zur Merkmalsauswahl mit historischer mehrdimensionaler Transinformation

1: $X_{\text{tmp}} \leftarrow \text{argmax}_{X_k \in X} \left(J_{MI}(X_k) \right)$
2: $J_H \leftarrow 0$
3: $S \leftarrow X_{\text{tmp}}$
4: **while** $|S| < l$ und $\delta_{HJMI} < 0.03$ **do**
5: $\quad X_{\text{tmp}} \leftarrow \text{argmax}_{X_k \in X \setminus S} \left(J_{HJMI}(X_k, S) \right)$
6: $\quad J_H \leftarrow J_{HJMI}(X_{\text{tmp}}, S)$
7: $\quad S \leftarrow S \cup X_{\text{tmp}}$
8: **end while**

Dadurch ändert sich Algorithmus 2 zu:

Das heißt, der Algorithmus wird beendet, wenn die Information durch ein neues Merkmal nicht um wenigstens z. B. 3 % ($\delta_{HJMI} < 0.03$) im Vergleich zum vorherigen gewählten Merkmal steigt.

2.2.2 Merkmalsauswahl für stetige Merkmale

Während die statistische Schätzung der Transinformation eines diskreten Merkmals relativ einfach ist, ist die direkte Bestimmung der Transinformation eines stetigen Merkmals ungleich schwieriger. Im diskreten Fall kann die Entropie über das Auszählen der Zeichen ermittelt werden. Im stetigen Fall ist das nicht möglich, womit die Bestimmung der Entropie und damit der Transinformation schwierig ist (Ma, 2021). Es werden alternative Ansätze benötigt.

Wie bereits erwähnt, haben sich in der Stochastik verschiedene Methoden zur Bestimmung der Abhängigkeit von Zufallsvariablen etabliert. Unter der Annahme, dass Merkmale als stetige Zufallsvariablen aufgefasst werden können, lassen sich diese Methoden der Stochastik auf die Merkmalsauswahl übertragen. Zwei Verfahren zur Bestimmung der Abhängigkeit von Zufallsvariablen bzw. Merkmalen basieren dabei auf Copulas und sollen nach einem kurzen Überblick über Copulas kurz dargestellt werden.

Copulas

Eine Copula ist eine mehrdimensionale Verteilungsfunktion. Sie ist gleichverteilt in $[0, 1]^d$, wobei d die Anzahl der Dimensionen ist. Sie ermöglicht eine getrennte Betrachtung von Randverteilungen und Abhängigkeitsstruktur.

Nach dem Satz von Sklar gilt: Sei $F_{\mathbf{x},\mathbf{y}}$ eine mehrdimensionale Verteilungsfunktion mit den Randverteilungen $F_{\mathbf{x}}$ und $F_{\mathbf{y}}$ zweier Zufallsvariablen $\mathbf{x}$ und $\mathbf{y}$. Dann existiert eine Copula C, sodass für alle $(x, y) \in \mathbb{R}^2$ gilt (Nelsen, 2006, S. 21 f.):

$$F_{\mathbf{x},\mathbf{y}}(x, y) = C_{\mathbf{x},\mathbf{y}}(F_{\mathbf{x}}(x), F_{\mathbf{y}}(y)). \tag{2.14}$$

Weiterhin gilt:

$$C_{\mathbf{x},\mathbf{y}}(u, v) = F_{\mathbf{x},\mathbf{y}}(F_{\mathbf{x}}^{-1}(u), F_{\mathbf{y}}^{-1}(v)), \tag{2.15}$$

wobei $F_{\mathbf{x}}^{-1}$ und $F_{\mathbf{y}}^{-1}$ die Inversen von $F_{\mathbf{x}}$ und $F_{\mathbf{y}}$ sind und $(u, v) \in [0, 1]^2$. Eine Copula ist eindeutig, wenn die Verteilungen stetig sind.

Merkmalsauswahl mit Maßzahlen der Abhängigkeit von Zufallsvariablen

Die erste Methode nutzt Copulas, um „nichtparametrische Maßzahlen der Abhängigkeit von Zufallsvariablen“ (Schweizer und Wolff, 1981, “Nonparametric Measures of Dependence for

Random Variables") zu definieren. Eine davon ist γ:

$$\gamma(\mathbf{x}, \mathbf{y}) = \sqrt{90 \int_0^1 \int_0^1 (C_{\mathbf{x},\mathbf{y}}(u, v) - uv)^2 du\, dv} \tag{2.16}$$

$$\tag{2.17}$$

Seth und Príncipe (2010) greifen diese Arbeit auf. Sie nehmen an, dass die Werte der Merkmale Realisationen der Zufallsvariablen sind. Vereinfacht gesagt wird angenommen, dass die Werte eines Merkmals $x_1, x_2, ..., x_n$ insgesamt n verschiedene Werte aus der Zufallsvariablen $\mathbf{x}$ sind sowie $y_1, y_2, ..., y_n$ insgesamt n verschiedene Werte der Zufallsvariablen $\mathbf{y}$. Damit kann mithilfe der empirischen Verteilungsfunktion in Gleichung 2.18 γ geschätzt werden:

$$F_{\mathbf{x}}(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(x_i \leq x) \tag{2.18}$$

$$C_{\mathbf{x},\mathbf{y}}(u, v) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(F_{\mathbf{x}}(x_i) \leq u) \mathbb{I}(F_{\mathbf{y}}(y_i) \leq v) \tag{2.19}$$

$$\gamma(\mathbf{x}, \mathbf{y}) = \sqrt{90 \frac{1}{L^2} \sum_{i=1}^{L} \sum_{j=1}^{L} \left(C_{\mathbf{x},\mathbf{y}}(u_i, v_j) - u_i v_j\right)^2} \tag{2.20}$$

Dabei ist $\mathbb{I}$ die Indikatorfunktion, für die $\mathbb{I}(A) = 1$ gilt, wenn die Bedingung A erfüllt ist, und 0 sonst. L gibt die Anzahl der Stützstellen für die numerische Integration an und $1/L \leq u_1 \leq u_2 \leq ... \leq u_L \leq 1$, $1/L \leq v_1 \leq v_2 \leq ... \leq v_L \leq 1$ bezeichnet die Stützstellen.

Für die Auswahl der Merkmale erörtern Seth und Príncipe (2010) verschiedene Verfahren. Sie verweisen schließlich auf die bereits in Kapitel 2.2.1 vorgestellte MRMR. Dabei wird die Transinformation I als Maß für die Abhängigkeit zweier Merkmale durch γ ersetzt. Damit ergibt sich (vergl. Gleichung 2.7):

$$J_{MRMR}(\mathbf{x}_k) = \gamma(\mathbf{x}_k, \mathbf{y}) - \frac{1}{|S|} \sum_{\mathbf{x}_j \in S} \gamma(\mathbf{x}_k, \mathbf{x}_j). \tag{2.21}$$

Dabei ist $\mathbf{x}_k$ das Merkmal unter Betrachtung, $\mathbf{x}_j$ ein bereits gewähltes Merkmal aus S und $\mathbf{y}$ die Zielgröße. S enthält wieder alle bereits gewählten Merkmale.

Wie bereits in Kapitel 2.2.1 erwähnt, kann die MRMR gute Ergebnisse liefern, ist in anderen Fällen aber der JMI oder HJMI unterlegen. Eine Erweiterung von γ in den mehrdimensionalen Raum wäre denkbar, auch wenn sie von Wolff und Schweizer nicht direkt gegeben wird (Schweizer und Wolff, 1981; Wolff, 1977, 1980).

Gleichzeitig stellt sich die Frage, wie γ in die klassische Informationstheorie einzuordnen ist. So beschreiben Timme et al. (2014) eine Menge von mehrdimensionalen Maßzahlen und ordnen diese ein. Allerdings ist keine davon mit dem hier beschriebenen γ vergleichbar.

Merkmalsauswahl mit Copula-Dichte

Die zweite Methode nutzt einen anderen Ansatz: Mithilfe der Dichte einer zweidimensionalen Copula $C_{\mathbf{x},\mathbf{y}}(u, v)$ kann die Entropie der Copula bestimmt werden. Wie sich herausgestellt hat, ist die Copula-Entropie zweier Zufallsvariablen deren Transinformation (Blumentritt und Schmid, 2012; Ma und Sun, 2011; Zeng und Durrani, 2011):

$$c_{\mathbf{x},\mathbf{y}}(u, v) = \frac{\mathrm{d}^2 C_{\mathbf{x},\mathbf{y}}(u, v)}{\mathrm{d}u \mathrm{d}v} \tag{2.22}$$

$$I_{CD}(\mathbf{x}, \mathbf{y}) = \int_0^1 \int_0^1 c_{\mathbf{x},\mathbf{y}}(u, v) \log\left(c_{\mathbf{x},\mathbf{y}}(u, v)\right) \mathrm{d}u\, \mathrm{d}v \tag{2.23}$$

Um die Verwechselung mit der Transinformation aus Kapitel 2.2.1 zu vermeiden, wird I_{CD} für die Transinformation basierend auf der Copula-Dichte verwendet. Ma (2021) nutzt diesen Ansatz zur Auswahl von Merkmalen $\mathbf{x}_k$ auf die Zielgröße $\mathbf{y}$ mit dem in Kapitel 2.2.1 S. 26 für die zweidimensionale Transinformation vorgestellten Algorithmus 1:

$$J_{MI,CD}(\mathbf{x}_k) = I_{CD}(\mathbf{x_k}, \mathbf{y}) \tag{2.24}$$

Gleichung 2.22 und Gleichung 2.23 können auf den mehrdimensionalen Fall erweitert werden. Allerdings kann die Schätzung in höheren Dimensionen problematisch sein. Zum einen steigt mit der Anzahl der Dimensionen die Komplexität der Berechnung, wie Blumentritt und Schmid (2012) erläutern. Zum anderen benötigen Schätzungen in höheren Dimensionen mehr Daten, um verlässliche Aussagen treffen zu können, was auch „Fluch der Dimensionalität“[7] genannt wird.

2.2.3 Andere Verfahren zur Merkmalsauswahl

Neben den bereits vorgestellten Verfahren gibt es weitere Verfahren, die zur Merkmalsauswahl genutzt werden können, wovon im Folgenden drei vorgestellt werden sollen. Für diese Arbeit am wichtigsten ist dabei die Merkmalsauswahl mithilfe von Random Forest, da dieses Verfahren später ebenfalls zum Vergleich herangezogen wird. Die beiden anderen Verfahren, korrelationsbasierte Merkmalsauswahl und Merkmalsauswahl mithilfe der Principle-Component-Analysis, werden regelmäßig im Zusammenhang mit der Merkmalsauswahl genannt und sind deshalb hier mit aufgeführt. Beide Verfahren weisen aber Merkmale auf, aufgrund derer sie für diese Arbeit ungeeignet sind.

Random Forest

Random Forest ist eigentlich ein eigener Algorithmus zum Machine-Learning. Da er aber in der Lage ist, verschiedene Merkmale nach Wichtigkeit zu ordnen, kann er auch zur Merkmalsauswahl genutzt werden. Damit ordnet sich Random Forest als Wrapper-Methode ein.

Prinzipiell basieren Random Forests auf Entscheidungsbäumen. Ein Entscheidungsbaum besteht aus einer Menge Knoten. Jeder Knoten hat keinen, einen oder zwei Kindknoten sowie einen oder keinen Elternknoten. Hat ein Knoten keine Kindknoten, wird er auch Blatt oder finaler Knoten genannt. Hat ein Knoten keinen Elternknoten, wird er auch Wurzel genannt.

Bei der Verwendung eines Entscheidungsbaumes wird an der Wurzel begonnen. Dabei wird anhand der von dem Knoten untersuchten Merkmale entschieden, ob als Nächstes der linke oder der rechte Kindknoten evaluiert wird. Kommt man an einem Blatt an, so enthält dieses den Ausgabewert.

Das Training eines Entscheidungsbaumes funktioniert dabei denkbar einfach: Begonnen wird mit der Wurzel. An dieser wird mit einem oder mehreren Merkmalen eine Entscheidung so festgelegt, dass der gemeinsame Fehler der beiden Kindknoten kleiner ist als der Fehler des aktuellen Knotens (Breiman et al., 1993, S. 230). Der Fehler wird dabei über die mittlere quadratische Abweichung der Daten, die mit dem Kindknoten assoziiert sind, bestimmt. Das wird für jeden Knoten so lange wiederholt, bis der Fehler klein genug ist oder ein Knoten sich nicht mehr spalten lässt.

Der Unterschied von Random Forest zu klassischen Entscheidungsbäumen ist dabei, dass nicht ein Baum, sondern eine Menge Bäume erzeugt wird (Breiman, 2001). Darüber hinaus wird an jedem Knoten der Bäume eine Menge zufälliger Merkmale m aus allen vorhandenen Merkmalen p gewählt, um den Knoten aufzuteilen. Zusätzlich werden die Datenpaare zur Bestimmung des Fehlers zufällig gewählt. Ziel ist es, damit eine Menge unkorrelierter Entscheidungsbäume zu erzeugen. Zur Bestimmung des Ergebnisses nach dem Training werden

[7] (Blumentritt und Schmid, 2012, “curse of dimensionality”).

die Ausgaben der verschiedenen Entscheidungsbäume gemittelt. Der optimale Wert für m hängt vom Problem ab, in der Arbeit von Hastie et al. (2009, S. 592) wird aber $m = \frac{p}{3}$ als Einstiegswert empfohlen.

Dadurch, dass bei Random Forest während des Trainings der Fehler bestimmt werden kann, kann auch der Fehler, der entsteht, wenn ein Merkmal weggenommen wird, evaluiert werden. Damit kann die Wichtigkeit einzelner Merkmale abgeschätzt werden, welche dann als Kriterium für die Merkmalsauswahl genutzt werden kann.

Korrelationsbasierte Merkmalsauswahl

Die korrelationsbasierte Merkmalsauswahl (engl. correlation-based feature selection, CFS) (Hall, 2000) basiert auf einer Heuristik, die ursprünglich von Ghiselli (1964, S. 182) über die Pearson-Korrelation r (Hartung et al., 2005, S. 73) definiert worden ist:

$$Merit_s = \frac{k\mathbf{r}_{\mathbf{y}\hat{\mathbf{x}}}}{\sqrt{k + k(k-1)\mathbf{r}_{\hat{\mathbf{x}}}}}. \tag{2.25}$$

Dabei bezeichnet $\mathbf{r}_{\mathbf{y}\hat{\mathbf{x}}}$ die durchschnittliche Merkmals-Zielgrößen-Korrelation und $\mathbf{r}_{\hat{\mathbf{x}}}$ die durchschnittliche Merkmals-Merkmals-Korrelation.

Die Auswahl der Merkmale erfolgt dabei durch eine Best-first-Suche. Wenn der $Merit_s$ durch das Hinzufügen von weiteren Merkmalen nicht mehr steigt, fällt die Suche zu der nächsten noch nicht betrachteten Untermenge von Merkmalen zurück. Nach einer vorher definierten Anzahl von Schritten wird die Suche abgebrochen und die Menge an Merkmalen gewählt, die den besten $Merit_s$ ergibt. Die nicht gewählten Merkmale werden abschließend erneut evaluiert. Der Menge der gewählten Merkmale werden nun noch die Merkmale hinzugefügt, deren Korrelation mit der Zielgröße größer ist als die höchste Korrelation zu einem der bereits gewählten Merkmale.

Einer der wesentlichen Nachteile der CFS ist genau diese Reevaluation. Dadurch lässt sich die Menge der zu wählenden Merkmale nicht beschränken. Es lässt sich nicht abschätzen, wie viele zusätzliche Merkmale in diesem Schritt gewählt werden. Eine Festlegung einer maximalen Anzahl von Merkmalen unter vollständiger Berücksichtigung des Algorithmus ist daher nicht möglich, weswegen der Algorithmus in dieser Arbeit nicht eingesetzt wird. Darüber hinaus verwendet CFS für stetige Merkmale den Pearson-Korrelationskoeffizienten, welcher nur lineare Abhängigkeiten darstellen kann. Das schränkt die Möglichkeiten einiger Machine-Learning-Methoden ein, nichtlineare Zusammenhänge darzustellen.

Principal-Component-Analysis

Während die Principal-Component-Analysis (PCA), auch Hauptkomponentenanalyse genannt, eigentlich zur Merkmalsextraktion verwendet wird (Jolliffe, 2002, S. 111), gibt es auch Arbeiten, die sie für Merkmalsauswahl nutzen (Malhi und Gao, 2004; Song et al., 2010). Bei der PCA werden zunächst Eigenwerte berechnet. Anschließend werden zu den n größten Eigenwerten die Eigenvektoren bestimmt. In einem Umkehrschritt können nun die Merkmale bestimmt werden, die den größten Einfluss auf die Eigenvektoren haben. Diese werden dann als relevante Merkmale gewählt.

Diese Methode eignet sich besonders, wenn die zu bestimmende Zielgröße nicht bekannt ist, da sie nur die Abhängigkeiten der Merkmale untereinander berücksichtigt. Allerdings bedeutet das auch, dass Merkmale gewählt werden können, die keinen Einfluss auf die Zielgröße haben. Im Umkehrschluss ergibt sich, dass mehr Merkmale als notwendig gewählt werden, wenn die Zielgröße bekannt ist. Deswegen wird diese Methode ebenfalls in dieser Arbeit nicht weiter betrachtet.

2.3 Machine-Learning mit neuronalen Netzen

Ziel der Arbeit ist es, mithilfe von Machine-Learning die Frequenz eines Prozessors zu wählen, bei der der Energieverbrauch am geringsten ist. In Gocht et al. (2019) hat der Author dieser Arbeit bereits gezeigt, dass ein Reinforcement-Ansatz mit dem hier vorgestellten Framework dieses Ziel erreichen kann. Im Rahmen dieser Arbeit soll daher ein anderer Ansatz evaluiert werden: Supervised Learning mithilfe von neuronalen Netzwerken. Dabei soll mithilfe der bereits gewählten Merkmale und eines neuronalen Netzwerks ein Energiemodell für unterschiedliche Frequenzen des Prozessors erstellt werden.

Neuronale Netzwerke wurden aufgrund ihrer Eigenschaft gewählt, nichtlineare Zusammenhänge gut abbilden zu können. Der folgende Abschnitt soll einen kurzen Überblick über die Funktionsweise von neuronalen Netzwerken sowie deren Training geben, um diese Entscheidung zu untermauern.

2.3.1 Neuronale Netze

Für die Popularität neuronaler Netze in den letzten Jahren lassen sich zwei Punkte in der jüngeren Geschichte ausmachen: Die Einführung des Backpropagation-Algorithmus Mitte der 1980er-Jahre (Haykin, 2009, S. 154) und die steigende Nutzung von Grafikkarten (GPUs) bzw. General Purpose Graphics-Processing-Units (GPGPUs) für wissenschaftliche Berechnungen in den letzten 20 Jahren.[8] Während die Einführung des Backpropagation-Algorithmus die rechentechnische Effizienz des Lernprozesses von neuronalen Netzwerken steigerte, führten GPUs zu einer signifikanten Steigerung der Rechenleistung. Damit lassen sich Netzwerke trainieren und auswerten, die zur Zeit der Erfindung des Rosenblatt-Perzeptrons 1958 nicht denkbar waren (Haykin, 2009, S. 77).

Die Grundidee hinter einem neuronalen Netz ist dabei recht einfach: Ein sogenanntes Neuron akkumuliert die gewichteten Eingangssignale und leitet sie durch eine Aktivierungsfunktion (Haykin, 2009, S. 153 ff.). Daraus resultiert ein Ausgangssignal, das an ein folgendes Neuron weitergeleitet wird. Das Konzept ist inspiriert von den Neuronen des Gehirns, die zusammenarbeiten, um verschiedene Aufgaben zu erledigen (Haykin, 2009, S. 33 ff.).

Eine Menge von Neuronen auf einer Ebene wird als Layer bezeichnet. Bei Klassifikationsaufgaben stehen am Ende so viele Ausgabeneuronen, wie es Zielklassen gibt. Das Neuron mit dem höchsten Wert gewinnt. Bei Regressionsproblemen steht am Ende ein einzelnes Neuron, das den Ausgabewert bestimmt. Eine vereinfachte Darstellung ist in Abbildung 2.4 zu sehen.

Beim Training eines neuronalen Netzwerks wird ein Eingangssignal angelegt und das Ausgangssignal bestimmt. Der Fehler zum gewünschten Signal wird jetzt wieder in das Netzwerk zurückgeführt, um ihn durch Anpassungen im Netzwerk selbst zu minimieren. Der dafür verwendete Algorithmus heißt Backpropagation und wird im Folgenden näher beschrieben. Beim erfolgreichen Trainieren wird so der Fehler immer kleiner. Die Hoffnung ist, dass bei der folgenden Anwendung des Netzwerkes, auch als Inferenz [engl. inference] bezeichnet, das Netzwerk in der Lage ist, eine Vorhersage über das gewünschte Signal zu treffen, ohne dieses Signal selbst bestimmen zu müssen.

2.3.2 Backpropagation

Prinzipiell unterscheidet man den Vorwärtsdurchgang und den Rückwärtsdurchgang. In Abbildung 2.5 ist eine Übersicht über den Vorwärtsdurchgang für ein Neuron j dargestellt. Für eine Stichprobe (engl. sample) n werden die m Eingangssignale $y_0(n), y_1(n), \ldots, y_m(n)$ des Neurons mit den entsprechenden Gewichten $\omega_{0j}(n), \omega_{1j}(n), \ldots, \omega_{mj}(n)$ multipliziert und

[8] Die erste Nvidia-Grafikkarte in den Top 500 (Dongarra und Luszczek, 2011) findet sich in der Liste vom November 2008, installiert im Sun Fire x4600/x6250 (Japan), in der Liste von November 2019 besitzen ca. 27 % der Top-500-Systeme eine Nvidia GPU.

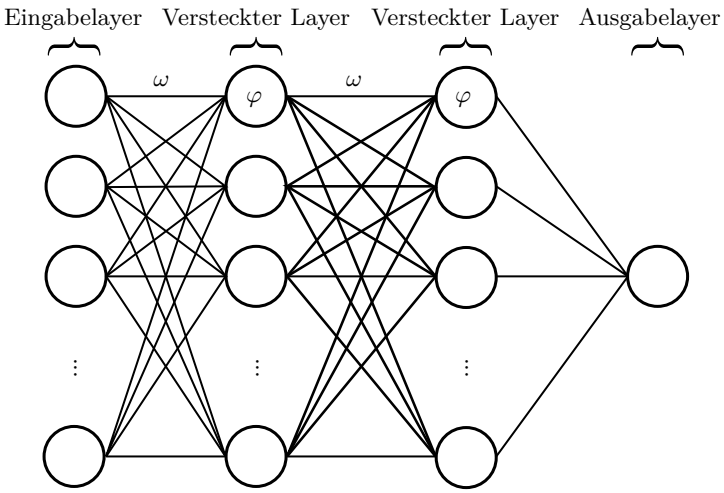


Abbildung 2.4: Vereinfachte Darstellung eines neuronalen Netzes mit zwei versteckten Layern, Aktivierungsfunktionen φ und Gewichten ω.

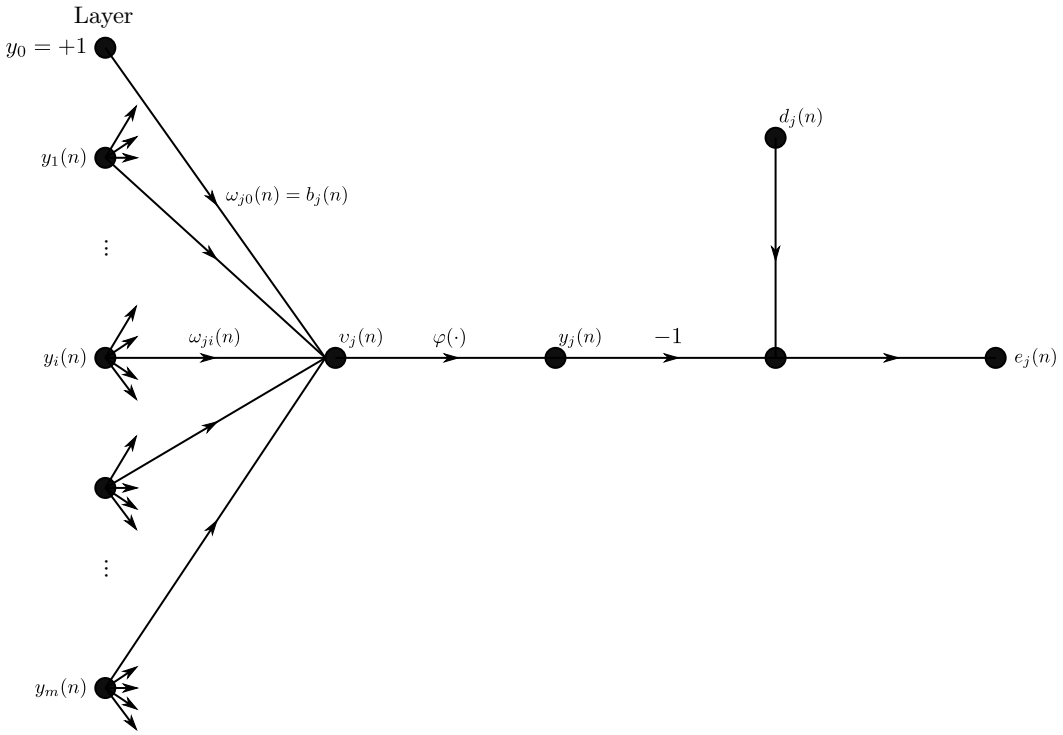


Abbildung 2.5: Signallauf für ein einzelnes Neuron j nach Haykin (2009, S. 159). φ repräsentiert die Aktivierungsfunktion, ω die Gewichte, v die gewichtete Eingabe für die Aktivierungsfunktion, y das Ein- bzw. Ausgabesignal, y_0 eine konstante Verschiebung, d das erwartete Ausgabesignal und e den Fehler.

durch die Aktivierungsfunktion $\varphi(\cdot)$ geleitet (Haykin, 2009, S. 159 ff.). Beim ersten Durchlauf werden die Gewichte mit zufälligen Werten initialisiert. Es ergibt sich der Ausgabewert des Neurons $y_j(n)$.

$$\upsilon_j(n) = \sum_{i=0}^{m} \omega_{ij}(n) y_{ij}(n)$$
$$y_j(n) = \varphi_j(\upsilon_j(n))$$

Wenn das Neuron Teil des letzten Layers ist und damit direkt das Ausgabesignal generiert, lässt sich mithilfe des erwarteten Ausgabewertes $d_j(n)$ der Fehler $e_j(n)$ bestimmen.

$$e_j(n) = d_j(n) - y_j(n)$$

Es folgt der Rückwärtsdurchgang. Diese Phase dient dazu, die einzelnen Gewichte zu korrigieren, um dadurch den Fehler $e_j(n)$ zu minimieren.

$$\Delta\omega_{ij}(n) = \eta\delta_j(n) y_i(n)$$
$$\delta_j(n) = e_j(n)\varphi'_j(\upsilon_j(n))$$

Die notwendige Änderung des einen Gewichtes $\Delta\omega_{ij}(n)$ ergibt sich aus dem lokalen Gradienten $\delta_j(n)$ zusammen mit dem Ausgabewert des Neurons aus dem vorherigen Layer y_i, welches über ω_{ij} mit dem aktuellen Neuron j verbunden ist, sowie der Lernrate η. Der lokale Gradient bezieht sich auf die Aktivierungsfunktion φ_j an der Stelle der gewichteten Eingabe der Aktivierungsfunktion $\upsilon_j(n)$, ergibt sich also aus der Ableitung φ'_j und berücksichtigt den Fehler.

Sollte der Knoten nicht Teil des letzten Layers sein, kann der Fehler nicht direkt bestimmt werden. Die Formel für den lokalen Gradienten passt sich entsprechend an:

$$\delta_j(n) = \varphi'_j(\upsilon_j(n)) \sum_k \delta_k(n)\omega_{kj}(n)$$

Dabei beschreibt k die Neuronen des folgenden Layers, z. B. des Ausgabelayers. Der lokale Gradient hängt also von dem vorhergehenden Gradienten ab, der am Ende vom Fehler des Netzwerkes abhängt.

Das Verfahren wird als Gradientenabstieg [engl. gradient descent] beschrieben. Wird für einen Durchgang nur ein Anteil der zur Verfügung stehenden Trainingsdaten verwendet, spricht man auch von Stochastic Gradient-Descent (Haykin, 2009, S. 210).

Wie schon geschrieben, wurde dieser Backpropagation-Algorithmus schon Mitte der 1980er-Jahre erfunden. Seitdem hat er sich in Teilen weiter entwickelt. So gilt im Moment *Adam* als Standardverfahren und wird auch in dieser Arbeit verwendet. Adam nutzt nicht eine globale Lernrate für alle Gewichte, sondern eine Lernrate pro Gewicht. Berechnet wird diese Lernrate basierend auf dem Mittelwert und der Standardverteilung der Gradienten (Kingma und Ba, 2015). Die Berechnung des Fehlers und die Verwendung der Aktivierungsfunktion ändern sich nicht. Da in dieser Arbeit neuronale Netzwerke vor allem verwendet und nicht weiterentwickelt werden, wird an dieser Stelle auf eine ausführliche Herleitung von *Adam* verzichtet, um die Arbeit nicht unnötig in die Länge zu ziehen.

2.3.3 Aktivierungsfunktionen

Nach dieser kurzen Zusammenfassung des Backpropagation-Algorithmus bleibt die Frage, welche Aktivierungsfunktionen verwendet werden können. Generell ist ein wichtiges Kriterium, dass diese sich ableiten lassen (Haykin, 2009, S. 165). Daraus ergibt sich eine Menge

von möglichen Aktivierungsfunktionen. Im Folgenden werden die Aktivierungsfunktionen zusammengefasst, die später verwendet werden.

Aus der Klasse der Sigmoid-Funktionen wird dabei gerne die logistische Funktion verwendet:

$$\varphi_j(v_j(n)) = \mathrm{sig}(v_j(n)) = \frac{1}{1+e^{[-av_j(n)]}} \qquad |a > 0$$
$$\varphi'_j(v_j(n)) = \mathrm{sig}'(v_j(n)) = \frac{ae^{[-av_j(n)]}}{(1+e^{[-av_j(n)]})^2}$$

Wenn $y_j(n) = \varphi_j(v_j(n))$ gesetzt wird, ergibt sich (Haykin, 2009, S. 166):

$$\varphi'_j(v_j(n)) = ay_j(n)[1-y_j(n)]$$

Das vereinfacht die Berechnung des lokalen Gradienten.

Ähnlich kann der Tangens hyperbolicus als Aktivierungsfunktion verwendet werden (Haykin, 2009, S. 166):

$$\begin{aligned}\varphi_j(v_j(n)) &= a\tanh(bv_j(n))\\ \varphi'_j(v_j(n)) &= ab(1-\tanh^2(bv_j(n)))\\ &= \frac{b}{a}[a-y_j(n)][a+y_j(n)]\end{aligned}$$

Eine andere verwendete Aktivierungsfunktion ist die ReLU-Funktion(Aggarwal, 2018, S. 13):

$$\begin{aligned}\varphi_j(v_j(n)) &= \max(0, v_j(n))\\ \varphi'_j(v_j(n)) &= \begin{cases}1 & \text{wenn } v_j(n) > 0\\ 0 & \text{wenn } v_j(n) < 0\end{cases}\end{aligned}$$

Zwar ist die Ableitung der ReLU-Funktion an der Stelle 0 nicht definiert, allerdings wird darüber gern hinweggesehen und die Ableitung an der Stelle 0 mit 0 angenommen (Aggarwal, 2018, S. 17).

3 Merkmalsauswahl für mehrdimensionale nichtlineare Abhängigkeiten

Wie bereits in Kapitel 2.1.3, S. 18, festgestellt, ist die maximale Aufnahmefähigkeit von Performance-Events durch die Performance-Counter beschränkt. Das stellt eine Beschränkung für die Merkmalsauswahl dar. Gleichzeitig hat der Author dieser Arbeit in Gocht et al. (2018) gezeigt, dass die richtige Anzahl an Merkmalen das Lernergebnis eines Machine-Learning-Verfahrens verbessern kann. Ziel dieses Kapitels ist es also, ein oder mehrere geeignete Merkmalsauswahlverfahren zu identifizieren, mit denen später die relevanten Events gewählt werden können.

Dazu beginnt das Kapitel mit einer Analyse der Problemstellung im Hinblick auf Merkmale und Zielgröße. Im folgenden Abschnitt wird die Transinformation basierend auf der Copula in die Theorie und die klassische Transinformation eingeordnet. Gleichzeitig werden Hinweise zur Bestimmung der Copula-Dichte gegeben. Damit kann die JMI auch für stetige Beobachtungen angewendet werden. Anschließend wird der Vorteil Copula-basierter Maßzahlen gegenüber klassischen Maßzahlen für die Analyse von Abhängigkeiten demonstriert. Das Kapitel schließt mit einer Analyse von kompletten Merkmalsauswahlverfahren.

Dabei stellt sich heraus, dass der Pearson-Korrelationskoeffizient als Maßzahl und die diskrete JMI als Merkmalsauswahlverfahren nicht für die Auswahl von Performance-Events geeignet sind. Gleichzeitig stellt sich die in Gocht et al. (2018) definierte Abbruchbedingung als unbrauchbar für einige hier verwendete Merkmalsauswahlverfahren heraus. Am Ende des Kapitels stehen somit eine Menge von Merkmalsauswahlverfahren sowie ein neues Abbruchkriterium.

3.1 Analyse der Problemstellung, Merkmale und Zielgröße

Ziel der Arbeit ist die Bestimmung der Core- und Uncorefrequenz, f_c und f_u, einer Region r, bei denen deren Energieverbrauch E_r minimal wird:[1]

$$(f_{c,r,\text{opt}}, f_{u,r,\text{opt}}) = \underset{(f_c,f_u)}{\operatorname{argmin}}(E_r(f_c, f_u)) \tag{3.1}$$

Der Energieverbrauch einer Region E_r hängt zusätzlich von der Laufzeit der Region ab, es gilt $E_r \sim t$. Eine Normierung auf den Energieverbrauch bei einer Referenzfrequenz $f_{c,\text{ref}}$ und $f_{u,\text{ref}}$, wie von Chadha und Gerndt (2019) vorgeschlagen, eliminiert diese Abhängigkeit von der Laufzeit und trägt zu einer Vereinfachung des Problems bei:[2]

$$E_n(f_c, f_u, r) = \frac{E_r(f_c, f_u)}{E_r(f_{c,\text{ref}}, f_{u,\text{ref}})} = \left[\frac{\text{J}}{\text{J}}\right] = [\,] \tag{3.2}$$

Es gilt weiterhin:

$$(f_{c,r,\text{opt}}, f_{u,r,\text{opt}}) = \underset{(f_c,f_u)}{\operatorname{argmin}}(E_n(f_c, f_u, r)) \tag{3.3}$$

[1] Ausgehend von Annahme 3 kann der Energieverbrauch einer Region von f_c und f_u sowie der Art und Menge der Instruktionen beeinflusst werden. Da die Instruktionen vom Quellcode abhängen, der wiederum von dem mit dem Programm zu lösenden Problem abhängt, bleiben zur Optimierung f_c und f_u.

[2] $[\,]$ gibt hier eine Abschätzung der Dimension bzw. der physikalischen Einheit der jeweiligen Größe an.

Mit E_n kann also $f_{c,r,\text{opt}}, f_{u,r,\text{opt}}$ genauso bestimmt werden wie mit E_r, womit E_n im Weiteren als Zielgröße verwendet wird. Theoretisch gilt $E_n \in]0, \infty[$, und es kann eine stetige Verteilung von E_n angenommen werden, womit E_n als stetige Zufallsvariable modelliert werden kann. Praktisch zeigt sich, dass $E_n(f_c, f_u, r)$ um 1 schwankt.

Zur Bestimmung von E_n sollen verschiedene Performance-Events p_i verwendet werden, wobei $p_i \in 0, 1, ..., K$ und K die Anzahl aller Performance-Events ist. Diese Events werden über die Ausführungszeit einer Region t_r gezählt, der Zählwert wird als $C_{p_i,r}$ vermerkt. Zur Bestimmung einer Eventrate der Region $R_{p_i}(r)$ wird $C_{p_i,r}$ auf t_r normiert: [2]

$$R_{p_i}(r) = \frac{C_{p_i,r}}{t_r} = \left[\frac{\#\text{Events}}{\text{s}}\right]. \tag{3.4}$$

Da die Eventrate R_{p_i} auf Zählungen basiert, gilt $R_{p_i} \in [0, \infty[$. In der Praxis ergibt sich für einige Events p_i eine ungünstige Häufung in der empirischen Verteilungsfunktion von R_{p_i} bei 0, da bestimmte Events in manchen Regionen nicht auftreten. Um dennoch eine stetige Verteilung annehmen zu können, kann ein gleichverteiltes Rauschen über die Daten gelegt werden. Damit kann R_{p_i} ebenfalls als stetige Zufallsvariable modelliert werden. In Beobachtungen zeigt sich, dass $R_{p_i}(r)$ für verschiedene p_i sehr unterschiedliche Verteilungsfunktionen annimmt.

Wie bereits in Kapitel 2.1.3, S. 18, geschrieben können maximal l Events gemessen werden. Damit ergibt sich:

$$E_n(f_c, f_u, r) = g(R_{p_1}(r), R_{p_2}(r), ..., R_{p_n}(r), f_c, f_u), \tag{3.5}$$

wobei $p_i \in S$ und S die Menge der ausgewählten Merkmale ist. Es gilt $|S| = n \leq l$. Die Funktion g soll durch ein neuronales Netzwerk, wie in Kapitel 2.3, S. 33, beschrieben, approximiert werden. Damit ergeben sich als Merkmale (engl. features) die verschiedenen R_{p_i} sowie f_c und f_u.

Zur Bestimmung der l relevanten Events kann ein Merkmalsauswahlverfahren verwendet werden. Um Vorannahmen über die Verteilungsfunktionen der Merkmale zu vermeiden, werden nichtparametrische Verfahren verwendet.

3.2 Merkmalsauswahl mit mehrdimensionaler Transinformation für stetige Merkmale

Für die Merkmalsauswahl stehen verschiedene Verfahren zur Verfügung. Kapitel 2.2 arbeitet mit den auf Transinformation basierten Verfahren in Kapitel 2.2.1 und den auf Copulas basierenden Verfahren in Kapitel 2.2.2 zwei historisch unterschiedlich gewachsene Ansätze heraus. Wie aber bereits in Kapitel 2.2.2, S. 30, beschrieben, hat sich gezeigt, dass Transinformation Copula-Entropie ist (Blumentritt und Schmid, 2012; Ma und Sun, 2011; Zeng und Durrani, 2011). Damit müssen Merkmale, die als stetige Zufallsvariablen modelliert werden können, nicht diskretisiert werden, um die in Kapitel 2.2.1 beschriebenen Verfahren anwenden zu können. Da bei der Diskretisierung immer auch Informationen verloren gehen, ist das hilfreich.

Allerdings ergeben sich bei der praktischen Anwendung von Copula-Entropie in transinformationsbasierten Merkmalsauswahlverfahren einige Fragen, besonders bei Merkmalsauswahlverfahren wie HJMI, bei denen mehrdimensionale Beziehungen zwischen Zufallsvariablen verwendet werden müssen. Zwar ist die Aussage „Transinformation ist Copula-Entropie“ an Klarheit kaum zu übertreffen. Allerdings ist in der Informations- und Kommunikationstheorie, aus der die Transinformation kommt, nicht ganz klar, was eine mehrdimensionale Transinformation eigentlich ist. So listen Timme et al. (2014) acht verschiedene Möglichkeiten, eine mehrdimensionale Transinformation zu berechnen, welche alle in jeweils einer

Maßzahl mit jeweils spezifischen Eigenschaften münden. Damit ist nicht offensichtlich, wie die Copula-Entropie verwendet werden muss, um sie in Verfahren wie HJMI verwenden zu können.

Eine andere Frage stellt sich bei der Berechnung der Copula-Entropie. So wird die Copula-Dichte benötigt, um die Entropie zu bestimmen. Um die Copula-Dichte bestimmen zu können, muss die Copula differenziert werden können. Bei der in Kapitel 2.2.2, S. 29, diskutierten Schätzung über die empirische Verteilungsfunktion ist die Differenzierbarkeit nicht gegeben. Diese Fragen sollen im Folgenden mit dem Fokus auf praktischen Lösungen diskutiert werden.

Zunächst der Satz von Sklar aus Kapitel 2.2.2, S. 29, erweitert auf drei Dimensionen (Nelsen, 2006, S. 47 f.): Sei $F_{\mathbf{x},\mathbf{y},\mathbf{z}}$ eine mehrdimensionale Verteilungsfunktion mit den Randverteilungen $F_{\mathbf{x}}$, $F_{\mathbf{y}}$ und $F_{\mathbf{z}}$ dreier Zufallsvariablen $\mathbf{x}$, $\mathbf{y}$ und $\mathbf{z}$. Dann existiert eine Copula C, sodass für alle $x, y, z \in \mathbb{R}$ gilt:

$$F_{\mathbf{x},\mathbf{y},\mathbf{z}}(x, z, y) = C_{\mathbf{x},\mathbf{y},\mathbf{z}}(F_{\mathbf{x}}(x), F_{\mathbf{y}}(y), F_{\mathbf{z}}(z)) \tag{3.6}$$

Für stetige Verteilungen lässt sich schreiben:

$$C_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w) = F_{\mathbf{x},\mathbf{y},\mathbf{z}}(F_{\mathbf{x}}^{-1}(u), F_{\mathbf{y}}^{-1}(v), F_{\mathbf{z}}^{-1}(w)), \tag{3.7}$$

wobei $F_{\mathbf{x}}^{-1}$, $F_{\mathbf{y}}^{-1}$ und $F_{\mathbf{z}}^{-1}$ die Inversen von $F_{\mathbf{x}}$, $F_{\mathbf{y}}$ und $F_{\mathbf{z}}$ sind und $u, v, z \in [0, 1]^3$.

Für die Copula-Dichte, die Copula-Entropie und damit die Transinformation ergibt sich (Blumentritt und Schmid, 2012; Ma und Sun, 2011):

$$c_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w) = \frac{\mathrm{d}^3 C_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w)}{\mathrm{d}u\,\mathrm{d}v\,\mathrm{d}w} \tag{3.8}$$

$$I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \int_{[0,1]^3} c_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w) \log\left(c_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w)\right) \mathrm{d}u\,\mathrm{d}v\,\mathrm{d}w \tag{3.9}$$

Um den vor allem historischen Unterschied zwischen diskreten und stetigen Zufallsvariablen zu unterstreichen, werden die stetigen Zufallsvariablen dick geschrieben, z. B. $\mathbf{x}$, während bei diskreten Zufallsvariablen Großbuchstaben verwendet werden, z. B. X. Das erlaubt gleichzeitig eine bessere Nachvollziehbarkeit der folgenden Idee der Einordnung der mehrdimensionalen Copula-Entropie und Copula-Dichte in die klassische Theorie um die Transinformation. Gleichzeitig wird die Transinformation, welche über die Copula-Dichte und damit die Copula-Entropie bestimmt wird, als *Transinformation basierend auf Copula-Dichte*, z. B. I_{CD}, referenziert.

3.2.1 Mehrdimensionale Copula-Entropie und mehrdimensionale Transinformation

Für die Einordnung von $I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in die Theorie zur Merkmalsauswahl mithilfe der klassischen Transinformation ist die Herleitung des Zusammenhangs zwischen Transinformation und Copula-Dichte hilfreich. In der Herleitung gilt (Blumentritt und Schmid, 2012; Ma und Sun, 2011):

$$I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \int_{\mathbb{R}^3} p_{\mathbf{x},\mathbf{y},\mathbf{z}}(x, y, z) \log\left(\frac{p_{\mathbf{x},\mathbf{y},\mathbf{z}}(x, y, z)}{p_{\mathbf{x}}(x) \cdot p_{\mathbf{y}}(y) \cdot p_{\mathbf{z}}(z)}\right) \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z. \tag{3.10}$$

In ihrer Analyse verschiedener Maßzahlen der Transinformationstheorie für diskrete Verteilungen X, Y, Z untersuchen Timme et al. (2014) auch eine sogenannte *Total Correlation* (TC):

$$TC(X, Y, Z) = \sum_{x \in X} \sum_{y \in Y} \sum_{z \in Z} p_{X,Y,Z}(x, y, z) \log\left(\frac{p_{X,Y,Z}(x, y, z)}{p_X(x) \cdot p_Y(y) \cdot p_Z(z)}\right). \tag{3.11}$$

Gleichung 3.11 ist das diskrete Analogon von Gleichung 3.10. Über die TC lässt sich auch der Zusammenhang zur JMI herstellen,

$$TC(X,Y,Z) = I(X,Y) + I(XY,Z), \tag{3.12}$$

und auf die Copula-Dichte-basierte Transinformation übertragen:

$$I_{CD}(\mathbf{x},\mathbf{y},\mathbf{z}) = I_{CD}(\mathbf{x},\mathbf{y}) + I_{CD}(\mathbf{xy},\mathbf{z}) \tag{3.13}$$

Laut Brown et al. (2012) lässt sich schreiben:

$$I(XY,Z) = I(Y,Z) + I(X,Z) - (I(X,Y) - I(X,Y|Z)) \tag{3.14}$$

$$I(X,Y) - I(X,Y|Z) = I(Y,Z) + I(X,Z) - I(XY,Z) \tag{3.15}$$

Mithilfe von Gleichung 3.15 kann die Übertragung aus Gleichung 3.13 genutzt werden, um Gleichung 2.12, S. 28, zur Berechnung von J_{HJMI} für diskrete Merkmale auf stetige Merkmale zu übertragen. Der Einfachheit halber folgt Gleichung 2.12 für ein diskretes Merkmal X_k unter Betrachtung, mit den bereits gewählten diskreten Merkmalen $X_j \in S$ und der diskreten Zielgröße Y:

$$J_{HJMI}(X_k,S) = J_H + I(X_k,Y) - \frac{\sum_{X_j \in S} \left(I(X_k,X_j) - I(X_k,X_j|Y)\right)}{|S|},$$

Dabei stellt J_H das Ergebnis von J_{HJMI} des zuletzt gewählten Merkmals dar.

Für ein stetiges Merkmal unter Betrachtung $\mathbf{x}_k$, die bereits gewählten stetigen Merkmale $\mathbf{x}_j \in S$ und die stetige Zielgröße $\mathbf{y}$ ergibt sich für die historische mehrdimensionale Transinformation basierend auf Copula-Dichte $J_{HJMI,CD}$:

$$J_{HJMI,CD}(\mathbf{x}_k,S) = J_{H,CD} + \frac{1}{|S|} \sum_{\mathbf{x}_j \in S} \left(I_{CD}(\mathbf{x}_j,\mathbf{x}_k,\mathbf{y}) - I_{CD}(\mathbf{x}_j,\mathbf{x}_k) - I_{CD}(\mathbf{x}_j,\mathbf{y})\right). \tag{3.16}$$

Dabei stellt $J_{H,CD}$ das Ergebnis von $J_{HJMI,CD}$ des zuletzt gewählten Merkmals dar. $J_{HJMI,CD}$ kann mit Algorithmus 3 zur Merkmalsauswahl verwendet werden.

3.2.2 Schätzung der mehrdimensionalen Transinformation basierend auf Copula-Dichte

Die Randverteilung für die Merkmale und Zielgröße $\mathbf{x}$, $\mathbf{y}$ und $\mathbf{z}$ lässt sich über die empirische Verteilungsfunktion schätzen:

$$n = |\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| \tag{3.17}$$

$$\widehat{F}_{\mathbf{x}}(x) = \frac{1}{n} \sum_{x_i \in \mathbf{x}} \mathbb{I}(x_i \leq x) \tag{3.18}$$

$$\widehat{F}_{\mathbf{y}}(y) = \frac{1}{n} \sum_{y_i \in \mathbf{y}} \mathbb{I}(y_i \leq y) \tag{3.19}$$

$$\widehat{F}_{\mathbf{z}}(z) = \frac{1}{n} \sum_{z_i \in \mathbf{z}} \mathbb{I}(z_i \leq z) \tag{3.20}$$

Die empirische Verteilungsfunktion kann ebenfalls für die Schätzung der Copula verwendet werden. Allerdings ist die empirische Verteilungsfunktion nicht, wie in Gleichung 2.22, S. 30, gefordert, differenzierbar.

Für die Schätzung der Copula-Dichte werden regelmäßig Histogrammschätzer genutzt (Blumentritt und Schmid, 2012; Ma und Sun, 2011). Ein wesentliches Problem des Histogrammschätzers ist allerdings der sogenannte „Fluch der Dimensionalität“ (Blumentritt und

Schmid, 2012, “curse of dimensionality”). Als Beispiel: Ein dreidimensionales Histogramm mit 10 Klassen in jeder Dimension hat am Ende insgesamt 1 000 Felder, in die Werte eingeordnet werden können. Wenn gleichzeitig nur 1 000 Stichproben gesammelt wurden, ist keine verlässliche Schätzung mehr sichergestellt. Blumentritt und Schmid (2012) illustrieren an einigen Experimenten die Verzerrungen in der Schätzung, die sich dadurch ergeben können. Gleichzeitig führt das Diskretisieren zu einem Verlust an Informationen: Je weniger Klassen, desto weniger Information bleibt erhalten. Genau dieser Verlust an Information soll aber vermieden werden.

Ein alternativer Ansatz ist die Anpassung der empirischen Verteilungsfunktion: Mit der Verwendung der logistischen Verteilung (Haykin, 2009, S. 165 f.), anstelle der Indikatorfunktion, kann die Copula ebenfalls geschätzt werden:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{3.21}$$

$$\frac{\mathrm{d}\sigma(x)}{\mathrm{d}x} = \frac{e^x}{(1+e^x)^2} = \sigma(x)\sigma(-x) \tag{3.22}$$

Gleichzeitig ist die Funktion differenzierbar, sodass sich die Copula-Dichte ermitteln lässt.

Abbildung 3.1 stellt Verläufe der logistischen Funktion $\sigma(-o \cdot x)$ für verschiedene o der Indikatorfunktion gegenüber. Zu sehen ist, dass für steigende o sich die logistische Funktion der Indikatorfunktion annähert und damit der Unterschied zur Indikatorfunktion kleiner wird.

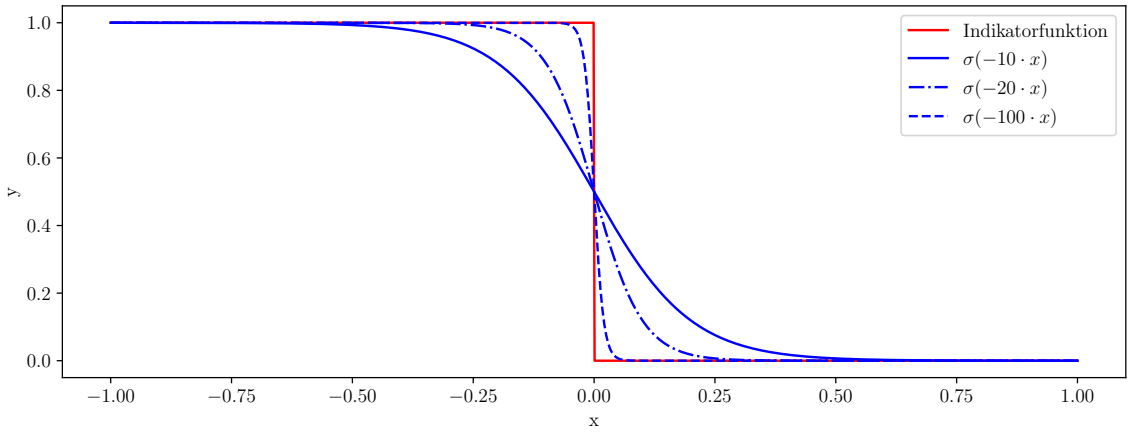


Abbildung 3.1: Indikatorfunktion im Vergleich mit verschiedenen logistischen Funktionen.

Für $\widehat{C}_{\mathbf{x},\mathbf{y},\mathbf{z}}$ ergibt sich dann:

$$\widehat{C}_{\mathbf{x},\mathbf{y},\mathbf{z}}(u,v,z) = \frac{1}{n}\sum_{i=1}^{n} \sigma(-o\cdot(\widehat{F}_{\mathbf{x}}(x_i)-u))\sigma(-o\cdot(\widehat{F}_{\mathbf{y}}(y_i)-v))\sigma(-o\cdot(\widehat{F}_{\mathbf{z}}(z_i)-w)), \tag{3.23}$$

wobei o, wie in Abbildung 3.1 zu sehen, die Annäherung an die Indikatorfunktion beeinflusst (Rinne, 2003, S. 484). Damit lässt sich für den Schätzer $\widehat{c}_{\mathbf{x},\mathbf{y},\mathbf{z}}(u,v,w)$ schreiben:

$$\begin{aligned}\widehat{c}_{\mathbf{x},\mathbf{y},\mathbf{z}}(u,v,w) &= \frac{\mathrm{d}^3\widehat{C}_{\mathbf{x},\mathbf{y},\mathbf{z}}(u,v,w)}{\mathrm{d}u\mathrm{d}v\mathrm{d}w} \\ &= \frac{1}{n}\sum_{i=1}^{n} \frac{o^3 \cdot e^{o\cdot(\widehat{F}_{\mathbf{x}}(x_i)-u)} \cdot e^{o\cdot(\widehat{F}_{\mathbf{y}}(y_i)-v)} \cdot e^{o\cdot(\widehat{F}_{\mathbf{z}}(z_i)-w)}}{(e^{o\cdot(\widehat{F}_{\mathbf{x}}(x_i)-u)}+1)^2(e^{o\cdot(\widehat{F}_{\mathbf{y}}(y_i)-v)}+1)^2(e^{o\cdot(\widehat{F}_{\mathbf{z}}(z_i)-w)}+1)^2}\end{aligned} \tag{3.24}$$

Die Wahl von o ist eine Abwägungsfrage. Ein großes o führt zu einer guten Annäherung an die Indikatorfunktion und einem geringen Fehler im Vergleich zum Ergebnis, das mit der empirischen Schätzung erreicht würde. Gleichzeitig ergibt sich eine Obergrenze von o numerisch aus (IEEE, 2019, S. 18): o steht im Exponenten der eulerschen Zahl. Bei der Verwendung von 64-Bit-Gleitkommazahlen ergibt sich ein maximaler Exponent von 384 sowie ein minimaler Exponent von -383.[3] Wie sich in der Realität zeigt, ist $o = 100$ ein guter Wert.

Mithilfe einer einfachen numerischen Integration kann schließlich $I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ geschätzt werden:

$$\widehat{I}_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} \sum_{k=1}^{n_w} \widehat{c}_{\mathbf{x},\mathbf{y},\mathbf{z}}(i\,\Delta u, j\,\Delta v, k\,\Delta w) \log\left(\widehat{c}_{\mathbf{x},\mathbf{y},\mathbf{z}}(i\,\Delta u, j\,\Delta v, k\,\Delta w)\right) \Delta u \Delta v \Delta w. \tag{3.25}$$

Da $u, v, w \in [0, 1]$ gilt $\Delta u = 1/n_u$, $\Delta v = 1/n_v$ und $\Delta w = 1/n_w$, wobei n_u, n_v, n_w die Anzahl der Stützstellen für die numerische Integration beschreiben. Im Folgenden wird angenommen, dass $n_u = n_v = n_w$. Die Schätzung des zweidimensionalen Falls ist analog zu dem in diesem Kapitel vorgestellten Verfahren.

Im Folgenden wird, so nicht anders angegeben, I_{CD} anstelle von $\widehat{I}_{CD}$ verwendet.

3.3 Normierung

Prinzipiell kann die Transinformation basierend auf Copula-Dichte nach [0,1] normiert werden, was einen Effekt auf die Merkmalsauswahlverfahren haben kann. Deshalb werden im Folgenden zwei Normierungsvarianten diskutiert, die später neben der nicht normierten Variante evaluiert werden.

Bisher wurde implizit angenommen, dass sich Gleichung 3.8 berechnen lässt. In einigen Fällen kann es aber sein, dass sich $C_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w)$ nicht nach $u\,v\,w$ ableiten lässt. Ein Beispiel ist die perfekte Abhängigkeit zwischen $\mathbf{x}, \mathbf{y}$ und $\mathbf{z}$, die einen der Randfälle der Copula darstellt (Blumentritt und Schmid, 2012, vergl. Fréchet–Hoeffding bounds). Per Definition ist in diesen Fällen $I_{CD} = \infty$. Um I_{CD} nach $[0, 1]$ normieren zu können, schlagen Blumentritt und Schmid (2012) die hier nur für den zwei- und dreidimensionalen Fall notierte Normierung vor:

$$I_{CD,\text{NormGauss}}(\mathbf{x}, \mathbf{y}) = \sqrt{1 - e^{-2 \cdot I_{CD}(\mathbf{x},\mathbf{y})}} \tag{3.26}$$

$$I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = -\frac{1}{2} \log\left[\left(1 - I_{CD,\text{NormGauss}}(\mathbf{x}, \mathbf{y}, \mathbf{z})\right)^2 \cdot \left(1 + 2 I_{CD,\text{NormGauss}}(\mathbf{x}, \mathbf{y}, \mathbf{z})\right)\right]. \tag{3.27}$$

Für Gleichung 3.27 wird eine numerische Lösung zur Bestimmung von $I_{CD,\text{NormGauss}}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ vorgeschlagen. `NormGauss` wird zur Bezeichnung genutzt, da die Grundlage für diese Normierung Gauss-Copula sind.

Ein zweiter Ansatz ergibt sich aus der Wahl des Schätzers $\widehat{C}$ (vergl. Gleichung 3.23). Der Schätzer wurde so gewählt, dass sich $C_{\mathbf{x},\mathbf{y},\mathbf{z}}(u, v, w)$ immer nach $u\,v\,w$ ableiten lässt. Damit lassen sich $\widehat{c}$ und $\widehat{I}_{CD}$ immer bestimmen und I_{CD} wird nie ∞. Dadurch lässt sich für diesen Schätzer das Maximum im Zweidimensionalen und Dreidimensionalen numerisch bestimmen, indem $\mathbf{x}, \mathbf{y}$ und $\mathbf{z}$ so gewählt werden, dass sie perfekt abhängig sind. Am einfachsten kann das erreicht werden, wenn die Abhängigkeit einer beliebigen Zufallsvariablen $\mathbf{w}$ zu sich selbst bestimmt wird:

$$I_{CD,\text{NormMax}}(\mathbf{x}, \mathbf{y}) = \frac{I_{CD}(\mathbf{x}, \mathbf{y})}{I_{CD}(\mathbf{w}, \mathbf{w})} \tag{3.28}$$

$$I_{CD,\text{NormMax}}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \frac{I_{CD}(\mathbf{x}, \mathbf{y}, \mathbf{z})}{I_{CD}(\mathbf{w}, \mathbf{w}, \mathbf{w})} \tag{3.29}$$

[3] $\widehat{F}_{\mathbf{x}}(x_i) - u$, $\widehat{F}_{\mathbf{y}}(y_i) - v$ und $\widehat{F}_{\mathbf{z}}(z_i) - w$ schwanken zwischen -1 und 1.

Im Gegensatz zu $I_{CD,\text{NormGauss}}$ hat $I_{CD,\text{NormMax}}$ damit eine lineare Abhängigkeit von I_{CD}.

Beide Normierungsvarianten haben einen Einfluss auf die Bestimmung von Maßzahlen wie $J_{HJMI,CD}$: Sie beeinflussen das Verhältnis der einzelnen Terme zueinander (vergl. Gleichung 3.16). Im Folgenden wird deshalb sowohl I_{CD} als auch $I_{CD,\text{NormGauss}}$, $I_{CD,\text{NormMax}}$ näher betrachtet.

3.4 Vergleich von Copula-basierten Maßzahlen mit der klassischen Transinformation und dem Pearson-Korrelationskoeffizienten

Für ein gutes Merkmalsauswahlverfahren werden Maßzahlen benötigt, die die Abhängigkeit einer stetigen Zielgröße von stetigen Merkmalen möglichst gut abbilden können. Die folgenden beiden Randfälle sollen dazu die Vorteile Copula-basierter Maßzahlen gegenüber der klassischen Transinformation und dem Pearson-Korrelationskoeffizienten r zeigen. Zur bereits diskutierten Copula-Dichte wird noch das von Schweizer und Wolff (1981) definierte γ hinzugezogen (vergl. Kapitel 2.2.2, S. 29), da dieses ebenfalls auf der Copula basiert. Rangkorrelationskoeffizienten wie Spearmans ρ oder Kendalls τ werden in dieser Arbeit nicht betrachtet.

Zur numerischen Berechnung von γ und der Transinformation basierend auf der Copula-Dichte werden 100 Stützstellen für die jeweiligen numerischen Integrationen genutzt. Für die Transinformation werden 10 verschiedene Klassen gewählt, $I_{MI,10}$. Auf die Analyse der Transinformation mit mehr Klassen wird verzichtet: Für die Schätzung der zweidimensionalen Transinformation werden zweidimensionale Histogramme verwendet, um die reale Häufigkeit einer Kombination von Klassen zu schätzen. Bei einem zweidimensionalen Histogramm mit z. B. 100 Klassen je Dimension ergeben sich 10000 Felder. Zusammen mit den im Folgenden verwendeten Wertepaaren mit zwischen 100 und 2000 Elementen ist keine sinnvolle Schätzung einer realen Häufigkeit mehr möglich.

3.4.1 Deterministische Abhängigkeit zweier Variablen

Der erste Fall soll verschiedene Fälle einer deterministischen Abhängigkeit simulieren. Dazu eignen sich funktionale Zusammenhänge. Für eine Variable $\mathbf{x}$ werden aus dem Bereich $[0, 6]$ in gleichmäßigem Abstand 1000 Werte generiert. Die Werte der Variablen $\mathbf{y}$ bestimmen sich über die angegebenen funktionalen Zusammenhänge aus $\mathbf{x}$.

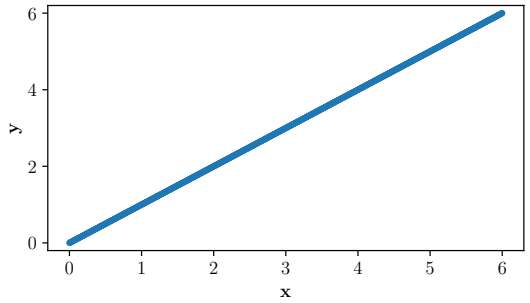


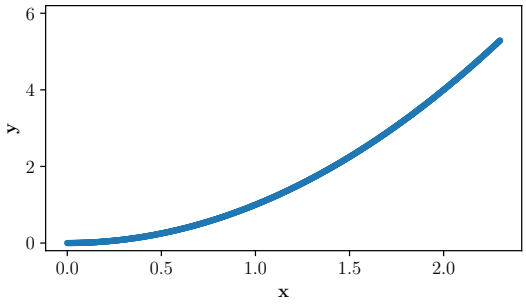
Abbildung 3.2: $\mathbf{x}$ und $\mathbf{y}$ sind perfekt linear voneinander abhängig: $\mathbf{x} = \mathbf{y}$.

Für die in Abbildung 3.2 dargestellten Merkmale gilt $\mathbf{x} = \mathbf{y}$. Tabelle 3.1 zeigt die Ergebnisse der unterschiedlichen Maßzahlen. Wie bereits geschrieben ist I_{CD} offensichtlich nicht auf $[0, 1]$ normiert. Für $I_{CD,\text{NormGauss}}$ ist zu beobachten, dass der Maximalwert 1 nicht erreicht wird. Wie in Kapitel 3.3 geschrieben ist der Schätzer I_{CD} so gewählt worden, dass sich die Copula ableiten lässt. Damit kann der Fall, dass I_{CD} per Definition ∞ wird, und $I_{CD,\text{NormGauss}} = 1$ nicht erreicht werden.

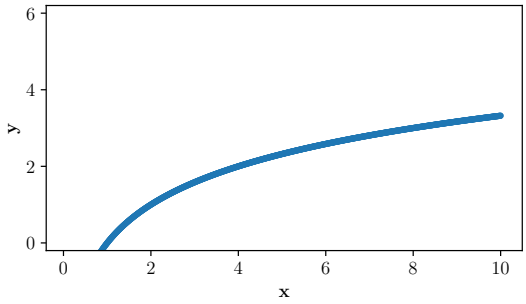
	Perfekt lineare Abhängigkeit
I_{CD}	2.201
$I_{CD,\text{NormGauss}}$	0.994
$I_{CD,\text{NormMax}}$	1.000
$I_{MI,10}$	1.000
γ	1.000
r	1.000

Tabelle 3.1: Vergleich verschiedener Maßzahlen für eine perfekt lineare Abhängigkeit: $\mathbf{x} = \mathbf{y}$.

Abbildung 3.3a und Abbildung 3.3b zweigen zwei Funktionen, bei denen die Abbildung von $\mathbf{x}$ auf $\mathbf{y}$ und von $\mathbf{y}$ auf $\mathbf{x}$ ebenfalls eindeutig, also bijektiv, ist. Die Funktionen sind monoton.



(a) Die Wertepaare wurden aus einer quadratischen Funktion generiert: $\mathbf{y} = \mathbf{x}^2$ mit der Bedingung $\mathbf{x} \geq 0$.



(b) Die Wertepaare wurden aus einer logarithmischen Funktion generiert: $\mathbf{y} = log_2\mathbf{x}$.

Abbildung 3.3: Wertepaare aus unterschiedlichen monotonen Funktionen.

Wie in Tabelle 3.2 zu sehen, besteht für die Copula-basierten Methoden kein Unterschied zur perfekt linearen Abhängigkeit. Anders sieht das für den Pearson-Korrelationskoeffizienten aus. Hier zeigt sich, dass der Pearson-Korrelationskoeffizient vor allem lineare Abhängigkeit bewertet.

$I_{MI,10}$ gibt ebenfalls einen Wert kleiner als 1 an, trotz eindeutigen Zusammenhangs. Erklären lässt sich der Unterschied mit der Diskretisierung in gleich große Klassen. Als Beispiel: Im Falle der logarithmischen Funktion gibt es eine Häufung von Werten für $\mathbf{y}$ in den Klassen, die die Werte größer 1 fassen. Gleichzeitig sind die Werte von $\mathbf{x}$ gleichmäßig über die Klassen verteilt. Dadurch scheinen die Werte von $\mathbf{y}$ ab einem bestimmten Punkt von $\mathbf{x}$ unabhängig zu sein. Mit einem anderen Diskretisierungsverfahren könnte das Problem adressiert werden, allerdings erhöht das wieder die Komplexität der Merkmalsauswahl.

3.4.2 Unabhängigkeit

Der zweite Randfall ist die Unabhängigkeit zweier Variablen. Eine bei realen Werten auftretende Unabhängigkeit lässt sich am besten durch eine zufällige Generierung der Werte für die Variablen $\mathbf{x}$ und $\mathbf{y}$ simulieren. Dazu werden für $\mathbf{x}$ und $\mathbf{y}$ verschiedene Verteilungsfunktionen angenommen. Aus dieser Verteilung wird eine Stichprobe gezogen und die jeweilige Maßzahl berechnet. Da durch die zufällige Generierung von $\mathbf{x}$ und $\mathbf{y}$ die Maßzahlen schwanken, wird der Versuch mehrmals wiederholt.

Für die in Abbildung 3.4 dargestellten Ergebnisse wurden eine Normalverteilung $N(0; 1)$

Wertepaare generiert aus einer	quadratischen Funktion	logarithmischen Funktion
I_{CD}	2.201	2.201
$I_{CD,\text{NormGauss}}$	0.994	0.994
$I_{CD,\text{NormMax}}$	1.000	1.000
$I_{MI,10}$	0.701	0.584
γ	1.000	1.000
r	0.968	0.905

Tabelle 3.2: Vergleich verschiedener Maßzahlen für Abhängigkeit von Wertepaaren generiert aus unterschiedlichen monotonen Funktionen.

und eine Gleichverteilung $U(0;6)$ verwendet. Aus den Verteilungen wurden 100 und 500 Stichproben gezogen. Der Versuch wurde 1 000-mal wiederholt. Für den Pearson-Korrelationskoeffizienten r wurde der Betrag $|r|$ gewählt, da für die Merkmalsauswahl lediglich relevant ist, ob zwei Merkmale zusammenhängen oder nicht. Ob der Zusammenhang positiv oder negativ ist, ist nicht relevant.

Keine Maßzahl erreicht immer exakt den Wert 0 für Unabhängigkeit. Da es sich bei den verwendeten Daten um aus den Verteilungen gezogene Stichproben handelt, ist dieses Verhalten zu erwarten. Abbildung 3.4 zeigt für alle Maßzahlen außer $I_{CD,\text{NormGauss}}$, dass mit steigender Stichprobengröße die Verteilung schmaler wird und weiter an 0 heranrückt. Bei $I_{CD,\text{NormGauss}}$ hingegen wird die Verteilung mit steigender Stichprobengröße breiter, rückt aber ebenfalls näher an 0 heran. Begründet ist dieser Effekt durch die nichtlineare Normalisierungsfunktion.

Die Verteilungen der Copula-basierten Maßzahlen sowie $|\rho|$ sind unabhängig von den zugrunde liegenden Verteilungsfunktionen. Bei I_{10}, also der klassischen Transinformation, ist eine leichte Abhängigkeit von der Verteilungsfunktion je nach zugrunde liegender Wahrscheinlichkeitsverteilung zu sehen. Vermutlich liegt diese Abhängigkeit in der Diskretisierung begründet. Bei einer steigenden Anzahl von Stichproben verliert dieser Effekt an Bedeutung.

Bei gleicher Stichprobengröße zeigt sich, dass $|\rho|$ die breiteste Verteilung hat, während $I_{CD,\text{NormGauss}}$ am weitesten vom angestrebten Wert 0 für die Unabhängigkeit entfernt ist. Von den Copula-basierten Merkmalen ist die Verteilung von γ für 100 Stichproben am weitesten links und damit der 0 nahe. Bei 500 stehen I_{10}, I_{CD} und $I_{CD,\text{NormMax}}$ am weitesten links.

Generell lässt sich sagen, dass alle Maßzahlen mit zunehmender Anzahl an Stichproben zwei unabhängige Variablen besser als unabhängig bewerten können als bei wenigen Stichproben.

Daraus lässt sich auch ein Abbruchkriterium für die Merkmalsauswahl ableiten: Die in dieser Arbeit diskutierten Merkmalsauswahlverfahren nutzen die Maßzahlen, um Abhängigkeiten der Zielgröße von einem Merkmal zu bestimmen. Wenn nun ein Merkmal hinzugefügt wird, das sich aufgrund der Stichprobengröße nicht von einem unabhängigen Merkmal unterscheiden lässt, kann das Merkmalsauswahlverfahren abgebrochen werden, da die zugrunde liegenden Maßzahlen bei der vorhandenen Stichprobengröße keine zuverlässige Schätzung der realen Abhängigkeiten mehr zulassen. Als Schwellwert kann das 95 %-Perzentil der Abhängigkeiten einer Menge von zufällig generierten Merkmalen verwendet werden.

3.5 Vergleich verschiedener Methoden zur Auswahl stetiger Merkmale

Mithilfe einer weiteren Simulation können die Merkmalsauswahlverfahren direkt verglichen werden. Für den Vergleich werden die folgenden Verfahren herangezogen:

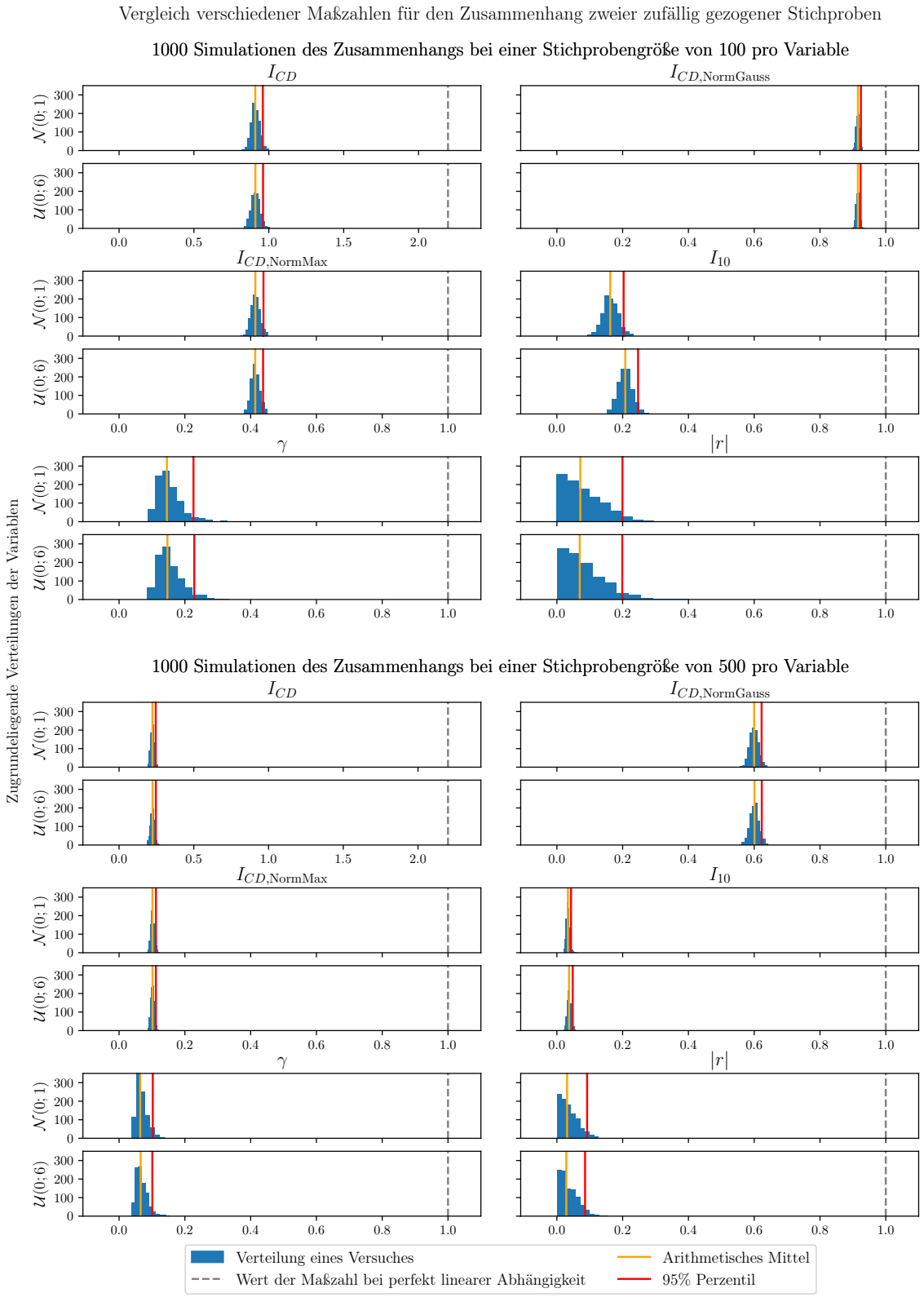


Abbildung 3.4: 1 000 Simulationen des Zusammenhangs zweier zufällig generierter Variablen mit 100 oder 500 Stichproben pro Variable. Die zugrunde liegenden Verteilungen sind eine Normalverteilung $N(0;1)$ und eine Gleichverteilung $U(0;6)$.

- J_{HJMI}, mit 10 Klassen (I_{10}), vergl. Kapitel 2.2.1, S. 27
- $J_{HJMI,CD}$ mit 100 Stützstellen, vergl. Kapitel 3.2, S. 38
- $J_{HJMI,CD,\text{NormGauss}}$ mit 100 Stützstellen, vergl. Kapitel 3.2, S. 38
- $J_{HJMI,CD,\text{NormMax}}$ mit 100 Stützstellen, vergl. Kapitel 3.2, S. 38
- $J_{MRMR,\gamma}$ mit 100 Stützstellen, vergl. Kapitel 2.2.2, S. 29
- $J_{MRMR,CD}$ mit 100 Stützstellen, nutzt das Verfahren Minimum-Redundancy Maximum-Relevance zusammen mit der Copula-Dichte, vergl. Kapitel 2.2.1, S. 26 und Kapitel 2.2.2, S. 30
- Merkmalsauswahl mittels Random Forests, vergl. Kapitel 2.2.3, S. 31

3.5.1 Erzeugung synthetischer Daten

Wie am Anfang des Kapitels und in Gleichung 3.5 zusammengefasst, soll in dieser Arbeit mithilfe eines neuronalen Netzwerks eine Funktion gelernt werden, um einen normierten Energieverbrauch auf Basis von Eventraten zu bestimmen. Hinter den verschiedenen Eventraten stehen Events in unterschiedlichen Teilen des Prozessors. Die Events hängen wiederum vom ausgeführten Programm ab. Gleichzeitig können sich die durch die Events abgebildeten Hardwarekomponenten gegenseitig beeinflussen. Als Beispiel: Wenn die Verbindung zum DRAM ausgelastet ist, kann das Laden von Daten in den L1-Cache entsprechend länger dauern. Diese Zusammenhänge können beliebig komplex werden. Deswegen wird zunächst die Annahme getroffen, dass die Eventraten voneinander unabhängig sind, sich aber über funktionale Zusammenhänge auf die Zielgröße wie den Energieverbrauch auswirken.

Um die Merkmalsauswahlverfahren in dieser Hinsicht zu vergleichen, eignet sich eine Simulation mithilfe von mehrdimensionalen nichtlinearen Funktionen. Ein Beispiel ist:

$$\mathbf{y} = f(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \mathbf{x}_0^{\mathbf{x}_1} \cdot \frac{1}{\mathbf{x}_2} \cdot \mathbf{x}_3 + \mathbf{x}_4.$$

Die Zielgröße $\mathbf{y}$ hängt von fünf relevanten Merkmalen, $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$, ab, welche damit als relevante Merkmale gelten. Zur Generierung von Stichproben wird für jedes Merkmal ein zufälliger, unabhängiger und gleichverteilter Wert zwischen 0 und 1 gezogen und der Funktionswert $\mathbf{y} = f(\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ bestimmt. Diese Merkmale sollen nun mithilfe der Merkmalsauswahlverfahren von einer Menge zufällig gezogener Merkmale, die aber nicht im Zusammenhang mit $\mathbf{y}$ stehen, unterschieden werden.

Guyon et al. (2004) schlagen im Rahmen der Feature Selection Challenge vor, die relevanten Merkmale als *wiederholte Merkmale* erneut unterzumischen. Dadurch kann geprüft werden, wie gut die verschiedenen Verfahren mit Dopplungen in den Daten umgehen können. Für die spätere Auswahl von Events ist das eine wichtige Eigenschaft, da es vorkommen kann, dass mit unterschiedlichen Events das Gleiche gezählt wird. Ein Beispiel: Wenn zwei Prozessoren miteinander kommunizieren, wird der eine Daten senden und der andere empfangen. Beides wird Events auslösen, die das Gleiche zählen, nur von unterschiedlicher Seite. Deshalb wird die oben erwähnte vereinfachende Annahme dahingehend angepasst, dass Eventraten entweder unabhängig oder direkt linear abhängig voneinander sind.

Die Realität wird zwischen diesen beiden Randfällen liegen. Wieder ein Beispiel: Daten, die im L1-Cache geladen werden und dort bearbeitet werden, müssen aus dem DRAM geladen werden. Es wird also Events für das Laden im DRAM und aus dem L1-Cache geben. Da die Daten im L1-Cache aber mehrfach verwendet werden können, kann es mehr Events aus dem L1-Cache geben als aus dem DRAM. Es kann aber auch sein, dass die Daten nach einer Bearbeitung verworfen werden und neue Daten geladen werden. In beiden Fällen wird

es zu einem unterschiedlichen Zusammenhang zwischen Events aus dem L1-Cache und dem DRAM kommen. Die folgende Simulation wird aber auf die beiden dargestellten Randfälle beschränkt, um eine gewisse Übersichtlichkeit zu gewährleisten.

Zur Generierung der Zielgröße $\mathbf{y}$ werden die folgenden Funktionen verwendet:

1. $\mathbf{y} = f_1(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{x}_0 \cdot \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 \cdot \mathbf{x}_4$
2. $\mathbf{y} = f_2(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{x}_0^{\mathbf{x}_1} \cdot \frac{1}{\mathbf{x}_2} \cdot \mathbf{x}_3 + \mathbf{x}_4$,

Für sie werden je 2 000 Stichproben genommen.

Im Folgenden werden zwei Szenarien vorgestellt, um die unterschiedlichen Eigenschaften der Verfahren zu demonstrieren. Im ersten gibt es neben den fünf relevanten Merkmalen keine wiederholten Merkmale. Im zweiten werden die fünf relevanten Merkmale wiederholt, sodass $\mathbf{x}_5 = \mathbf{x}_0$, $\mathbf{x}_6 = \mathbf{x}_1$, $\mathbf{x}_7 = \mathbf{x}_2$, $\mathbf{x}_8 = \mathbf{x}_3$ und $\mathbf{x}_9 = \mathbf{x}_4$. In beiden Fällen werden die Merkmale mit zufällig gezogenen Merkmalen auf insgesamt 100 Merkmale aufgefüllt. Die zufällig generierten Merkmale sind zur Hälfte normalverteilt und zur anderen Hälfte gleichverteilt.

Abbildung 3.5 und Abbildung 3.6 zeigen einen Vergleich der Merkmalsauswahlverfahren im ersten respektive zweiten Szenario. Auf der x-Achse werden die gewählten Merkmale in der Auswahlreihenfolge dargestellt. Für die Merkmalsauswahlverfahren, die die Maßzahlen J_{HJMI}, $J_{HJMI,CD}$ sowie $J_{HJMI,CD,\text{NormGauss}}$ und $J_{HJMI,CD,\text{NormMax}}$ nutzen, ist es egal, ob ein Merkmal über die Maßzahl, die Änderung der Maßzahl oder die normierte Änderung der Maßzahl ausgewählt wird. Gleichzeitig erlaubt die Darstellung der normierten Änderung der Maßzahl eine Einschätzung der in Kapitel 2.2.1, S. 27, definierten Abbruchbedingung, weshalb diese in Abbildung 3.5 und Abbildung 3.6 angegeben ist. Diese Abbruchbedingung bricht das Auswahlverfahren ab, wenn die normierte Änderung der Maßzahl unter einem Schwellwert von $0,03$ liegt. Sie ist nur bei den HJMI-basierten Verfahren sinnvoll, da die Maßzahl mit jedem hinzugefügten Merkmal potenziell größer wird. Die MRMR-basierten Maßzahlen sind so angelegt, dass sie maximal 1 und minimal 0 ergeben. Bei Random Forest wird die Wichtigkeit der Merkmale so normiert, dass die Summe über alle Merkmale 1 ergibt. Deshalb ist für $J_{MRMR,\gamma}$, $J_{MRMR,CD}$, $J_{MRMR,CD,\text{NormGauss}}$ und Random Forest nur die Maßzahl dargestellt.

Da alle Maßzahlen, außer der Wichtigkeit bei Random Forest, den Zusammenhang eines Merkmals mit der Zielgröße berücksichtigen, lässt sich für diese Merkmalsauswahlverfahren eine weitere Abbruchbedingung, wie in Kapitel 3.4.2 beschrieben, definieren: Das Verfahren kann abgebrochen werden, wenn sich der Zusammenhang eines gewählten Merkmals mit der Zielgröße nicht mehr von zufällig generierten Merkmalen unterscheidet. Dazu wird, wie in Kapitel 3.4.2 beschrieben, das 95 %-Perzentil der Maßzahl von mehreren zufällig generierten Merkmalen verwendet.

3.5.2 Szenario 1 – fünf relevante Merkmale

Abbildung 3.5 zeigt den Vergleich der Merkmalsauswahlverfahren, wenn fünf relevante Merkmale vorhanden sind. Idealerweise werden zunächst die relevanten Merkmale x_0 bis x_4 ausgewählt, bevor eines der zufälligen Merkmale ausgewählt wird. Zusätzlich sollte der Wert der angegebenen Maßzahl zur Auswahl oder für den Zusammenhang zwischen dem Merkmal und der Zielgröße unter den Schwellwert der jeweiligen Abbruchbedingung sinken, nachdem alle relevanten Merkmale gewählt worden sind.

Die HJMI-basierten Merkmalsauswahlverfahren $J_{HJMI,CD}$, $J_{HJMI,CD,\text{NormMax}}$ sowie die MRMR-basierten Verfahren $J_{MRMR,CD,\text{NormGauss}}$, $J_{MRMR,CD}$, $J_{MRMR,CD,\gamma}$ erkennen sowohl für Funktion f_1 als auch f_2 alle relevanten Merkmale. $J_{HJMI,CD,\text{NormGauss}}$ mischt unter die relevanten Merkmale ein nicht relevantes. Mit beiden Abbruchbedingungen würde das zu einem Stopp der Merkmalsauswahl an dem nicht relevanten Merkmal führen, womit die

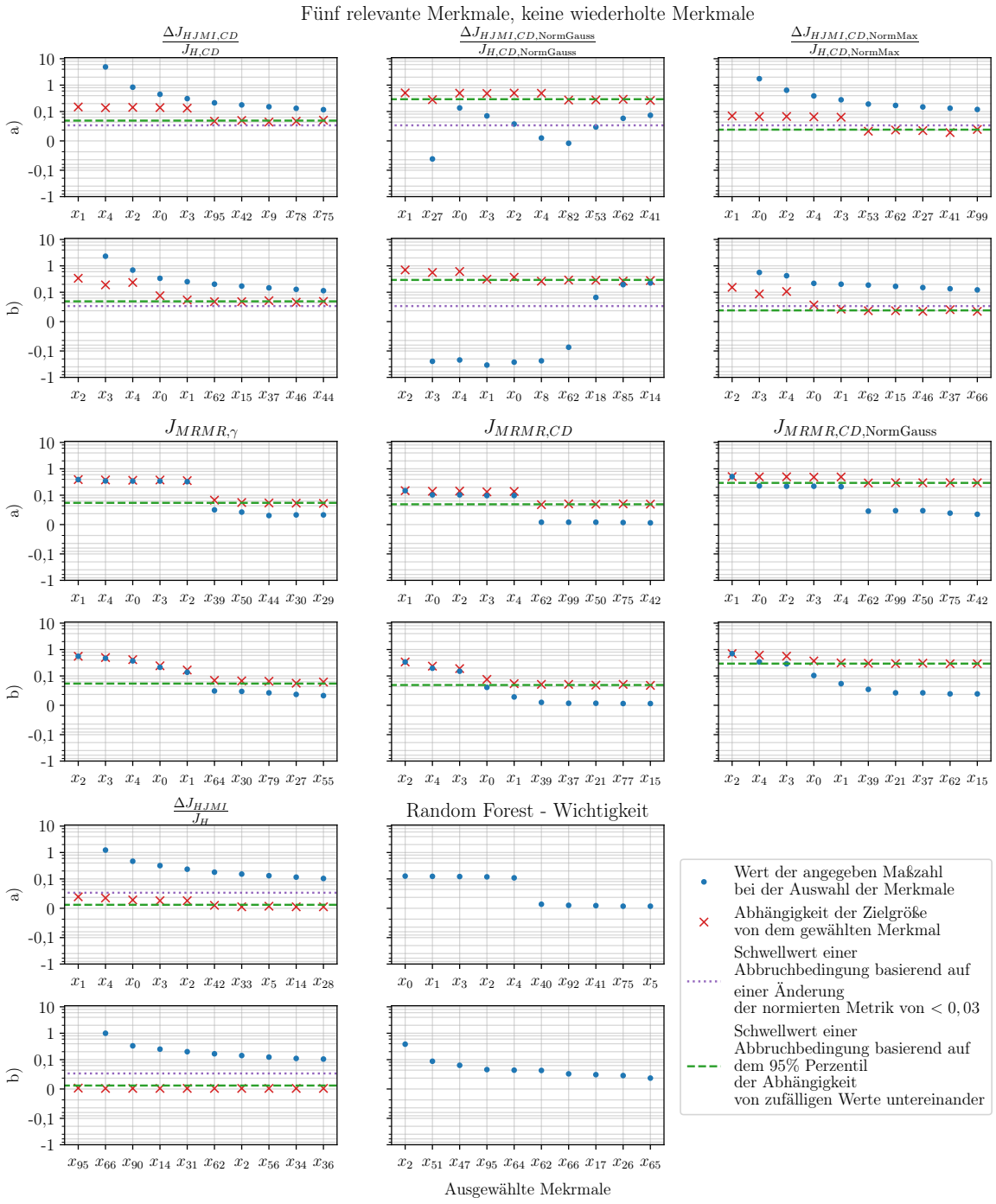


Abbildung 3.5: Vergleich verschiedener Merkmalsauswahlverfahren. Die x-Achse repräsentiert die ausgewählten Merkmale in der Reihenfolge, in der sie gewählt wurden. Die Merkmale 0 bis 4 sind relevante Merkmale $\mathbf{x}_0, ..., \mathbf{x}_4$, die über $f_1(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 \cdot \mathbf{x}_4$ bzw. $f_2(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{y} = \mathbf{x}_0^{\mathbf{x}_1} \cdot \frac{1}{\mathbf{x}_2} \cdot \mathbf{x}_3 + \mathbf{x}_4$ mit der Zielgröße $\mathbf{y}$ zusammenhängen. Alle weiteren Merkmale sind zufällig generiert. Der Bereich zwischen $0,01$ und $-0,01$ der y-Achse ist linear dargestellt.

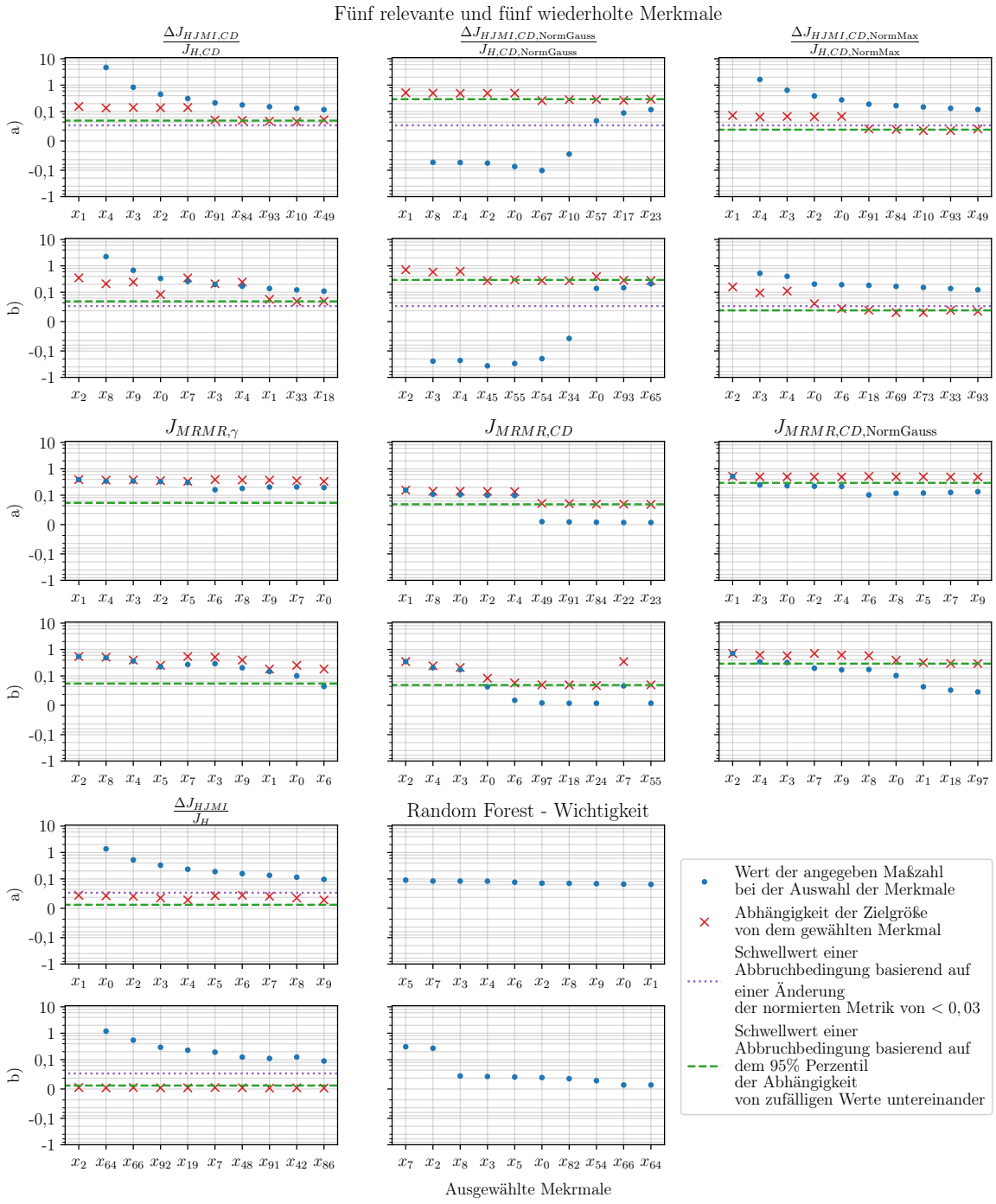


Abbildung 3.6: Vergleich verschiedener Merkmalsauswahlverfahren. Die x-Achse repräsentiert die ausgewählten Merkmale in der Reihenfolge, in der sie gewählt wurden. Die Merkmale 0 bis 4 sind relevante Merkmale $\mathbf{x}_0, ..., \mathbf{x}_4$, die über $f_1(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 \cdot \mathbf{x}_4$ bzw. $f_2(\mathbf{x}_0, ..., \mathbf{x}_4) = \mathbf{y} = \mathbf{x}_0^{\mathbf{x}_1} \cdot \frac{1}{\mathbf{x}_2} \cdot \mathbf{x}_3 + \mathbf{x}_4$ mit der Zielgröße $\mathbf{y}$ zusammenhängen. Die Merkmale x_5 bis x_9 sind Wiederholungen der Merkmale x_0 bis x_4. Der Bereich zwischen $0,01$ und $-0,01$ der y-Achse ist linear dargestellt.

übrigen relevanten Merkmale nicht gewählt würden. Der Merkmalsauswahl mit Random Forest gelingt es, für f_1 alle und für f_2 ein relevantes Merkmal zu finden. J_{HJMI} findet nur für Funktion f_1 alle relevanten Merkmale.

Für die HJMI-basierten Verfahren zeigt sich, dass die Abbruchbedingung über die Änderung der normierten Maßzahl nicht rechtzeitig greift. So sorgen die nicht relevanten Merkmale immer noch für eine ausreichende Änderung der Maßzahl für die Merkmalsauswahl. Wie Gocht et al. (2018) feststellen, kann die Abbruchbedingung dennoch hilfreich sein, da sich bei vielen relevanten Merkmalen die Frage stellt, wie viel mehr nützliche Information ein weiteres relevantes Merkmal bringt.

Bei der Abbruchbedingung über das 95 %-Perzentil kann es ebenfalls vorkommen, dass nicht nur relevante Merkmale gewählt werden, wie für $J_{MRMR,\gamma}$ deutlich zu sehen ist. Da der Schwellwert für die Abbruchbedingung aus dem 95 %-Perzentil des Zusammenhangs von zufällig gewählten Merkmalen generiert wird, ist es sehr wahrscheinlich, das einige der 95 ebenfalls zufällig generierten Merkmale einen Zusammenhang von mehr als dem Schwellwert mit der Zielgröße haben. Zwar kann die Wahrscheinlichkeit dafür verringert werden, wenn anstelle des 95 %-Perzentils das 99 %-Perzentil oder das 99,9 %-Perzentil verwendet wird, allerdings wird dann der Schwellwert weniger stabil.

Dennoch ist der Schwellwert über das 95 %-Perzentil hilfreich, da er sich sowohl für die HJMI-basierten Maßzahlen als auch für die MRMR-basierten Maßzahlen verwenden lässt. Für Random Forest lässt sich die Abbruchbedingung nicht direkt verwenden, kann aber abgewandelt werden: Es gibt für f_1 eine klare Schwelle zwischen den relevanten und zufällig generierten Merkmalen. Durch Bestimmung des 95 %-Perzentils der Wichtigkeit der zufälligen Merkmale kann für diese Funktion ein Schwellwert definiert werden, ab wann ein Merkmal als relevant bewertet wird. Generalisiert kann eine Menge Merkmale zufällig generiert werden und dann zu den zu untersuchenden Merkmalen hinzugefügt werden. Nachdem die Merkmale bewertet worden sind, kann die Wichtigkeit der hinzugefügten Merkmale genutzt werden, um den entsprechenden Schwellwert zu bestimmen. Im Gegensatz zu den anderen Maßzahlen kann der Schwellwert aber nicht im Vorhinein bestimmt werden, da die Wichtigkeit eines Merkmals, durch die Normierung der Summe aller Merkmale auf eins, von der Menge der zu untersuchenden Merkmale abhängt.

3.5.3 Szenario 2 – fünf relevante Merkmale, fünf wiederholte Merkmale

Abbildung 3.6 zeigt die Ergebnisse, sobald es zu Dopplungen in den Merkmalen kommt. Idealerweise wird entweder jeweils eines der relevanten Merkmale x_0 bis x_4 oder der entsprechenden wiederholten Merkmale x_5 bis x_9 ausgewählt, bevor eines der zufälligen Merkmale ausgewählt wird. Zusätzlich sollte der Wert der angegebenen Maßzahl zur Auswahl oder für den Zusammenhang zwischen dem Merkmal und der Zielgröße unter den Schwellwert der jeweiligen Abbruchbedingung sinken, nachdem alle relevanten Merkmale bzw. deren Wiederholung gewählt worden sind.

$J_{HJMI,CD}$, $J_{HJMI,CD,\text{NormMax}}$, $J_{MRMR,\gamma}$, $J_{MRMR,CD}$ und $J_{HJMI,CD,\text{NormGauss}}$ erkennen alle relevanten Merkmale. Bei einigen Verfahren werden die wiederholt eingeführten Merkmale gewählt, anstelle der Originale. Da die Merkmale aber theoretisch identisch sind, sind hier eher numerische Effekte entscheidend. $J_{HJMI,CD}$, $J_{MRMR,\gamma}$ und $J_{HJMI,CD,\text{NormGauss}}$ wählen mehr als die fünf relevanten Merkmale aus. Die Maßzahlen können also Redundanzen nur bedingt identifizieren, womit mehr Merkmale ausgewählt werden als notwendig.

$J_{HJMI,CD,\text{NormGauss}}$ findet für f_2 nur drei relevante Merkmale, bevor nicht relevante Merkmale gewählt werden. Ähnlich sieht es für J_{HJMI} aus: Für f_2 wird nur ein relevantes Merkmal gefunden. Mithilfe von Random Forest werden für f_2 nur drei Merkmale mit deren Wiederholungen gefunden. Die Merkmale x_1 und x_4, bzw. ihre Wiederholungen, fehlen.

3.5.4 Schlussfolgerungen aus den Simulationen

Die beiden Szenarien zeigen klar die Grenzen von J_{HJMI} und Random Forest auf. Sobald die Zusammenhänge zwischen den Merkmalen und der Zielgröße nichtlinar werden, wie in f_2, haben diese Merkmalsauswahlverfahren Probleme, alle relevanten Merkmale zu finden. Die Copula-basierten Verfahren schneiden, mit Ausnahme von $J_{HJMI,CD,\text{NormGauss}}$, besser ab. $J_{HJMI,CD,\text{NormGauss}}$ scheint weniger zuverlässig als die anderen Merkmalsauswahlverfahren zu sein, wie in Funktion f_1 in Szenario 1 und Funktion 2 in Szenario 2 zu sehen. In beiden Simulationen werden nicht alle relevanten Merkmale gefunden. Das Verfahren bietet demnach nur bedingt Vorteile gegenüber den klassischen Verfahren und wird deshalb im Weiteren nicht mehr betrachtet. Die anderen Copula-basierten Verfahren sind in der Lage, in den untersuchten Simulationen alle relevanten Merkmale zu identifizieren. Sie sollen daher an den Eventraten im Zusammenspiel mit verschiedenen neuronalen Netzen evaluiert werden. J_{HJMI} und Random Forest werden als klassische Vertreter für Merkmalsauswahlverfahren ebenfalls evaluiert.

3.6 Zusammenfassung

In diesem Kapitel wurden ausführlich die theoretischen Grundlagen der Arbeit diskutiert. Nach einer Analyse der Problemstellung und der sich daraus ergebenden Herausforderungen an die Merkmalsauswahlverfahren wurde eines dieser Merkmalsauswahlverfahren weiterentwickelt. So kann nun neben dem MRMR auch der JMI und HJMI mithilfe der Copula-Entropie bzw. der Copula-Dichte bestimmt werden. Dabei wurden gleichzeitig pragmatische Lösungen zur Berechnung der Copula-Entropie vorgeschlagen.

Es schloss sich eine Evaluation der verschiedenen Maßzahlen an. Zunächst wurden an einigen zweidimensionalen Beispielen die Vorteile der Copula-basierten Maßzahlen gegenüber der klassischen Transinformation oder dem Pearson-Korrelationskoeffizienten gezeigt. Anschließend fand eine Evaluation der Merkmalsauswahlverfahren an sich statt, um die Unterschiede der Verfahren zu analysieren. Das nächste Kapitel wird sich nun mit den Details zur Implementierung des gesamten Frameworks beschäftigen.

4 Entwicklung und Umsetzung des Frameworks

Dieses Kapitel beschäftigt sich mit der Entwicklung und Umsetzung des Frameworks.

Der vielleicht wichtigste Teil ist die READEX Runtime Library (RRL). Neben der Einteilung eines Programmes in relevante und nicht relevante Regionen übernimmt sie auch die Vermessung der relevanten Regionen, das Sammeln der Daten für das Training der Machine-Learning-Modelle sowie schließlich die Optimierung mit diesen Modellen. Die in dieser Arbeit verwendete RRL basiert dabei auf meinen Vorarbeiten aus dem READEX-Projekt, diese wurden jedoch signifikant erweitert, wie in diesem Kapitel dargestellt wird.

Für das Vermessen, Datensammeln und die Optimierung und die Evaluierung des Frameworks wird ein Testsystem benötigt. Dieses wird mit seinen Eckdaten ebenfalls in diesem Kapitel beschrieben. Dabei wird auch kurz auf den Zusammenhang zwischen dem Prozessor und den Performance-Events, die auf diesem gezählt werden können, eingegangen.

Neben der RRL und dem Testsystem sind aber auch die verwendeten Benchmarks relevant. So werden neben einer Menge von etablierten Anwendungsbenchmarks auch speziell angepasste Kernel verwendet. Mithilfe der Kernel lassen sich die durch die Merkmalsauswahlverfahren ausgewählten Performance-Events besser evaluieren, da Kernel meist wohldefinierte Randfälle abbilden.

Es folgt eine Erläuterung, wie die Daten, die mittels RRL gesammelt wurden, aufbereitet werden, um daraus mittels Merkmalsauswahl die relevanten Events zu wählen. Dabei werden auch kurz Implementierungsdetails der Merkmalsauswahl in Bezug auf die Komplexität der verwendeten Algorithmen diskutiert. Schließlich werden die verwendeten neuronalen Netzwerke beleuchtet.

4.1 Erweiterungen der READEX Runtime Library

Ursprünglich wurde die RRL von mir im Rahmen des READEX-Projektes (Kjeldsberg et al., 2017) umgesetzt. Im Rahmen dieser Arbeit wurde sie um den Call-Tree und die Calibration-Module erweitert. Da es für das Verständnis hilfreich ist, wird aber zunächst auf den gesamten Aufbau der RRL eingegangen, bevor die Änderungen im Rahmen dieser Arbeit im Detail erläutert werden.

4.1.1 Grundlegender Aufbau der READEX Runtime Library

Die READEX Runtime Library (RRL) nutzt Score-P (vergl. Kapitel 2.1.3, S. 17), welches ein sogenanntes Substrat-Interface (Schöne et al., 2017) bereitstellt. Das Substrat-Interface wurde entworfen, um Score-P mittels Plug-ins erweitern zu können. Es leitet alle Programm-Events, die Score-P aufnimmt, an das entsprechende Plug-in, in diesem Fall die RRL, weiter.

Abbildung 4.1 stellt den prinzipiellen Aufbau der RRL sowie das Zusammenspiel der verschiedenen Komponenten dar. Ein Programm, das mit Score-P instrumentiert ist, erzeugt Enter- und Leave-Events für unterschiedliche Regionen wie Funktionen oder OpenMP-parallelisierte Schleifen. Metrik-Plug-ins wiederum können verschiedene Informationen aufzeichnen und an Score-P weitergeben (Schöne et al., 2017), welches diese dann mit den Enter- und Leave-Events verknüpft. Die Events werden von Score-P zusammen mit gesammelten

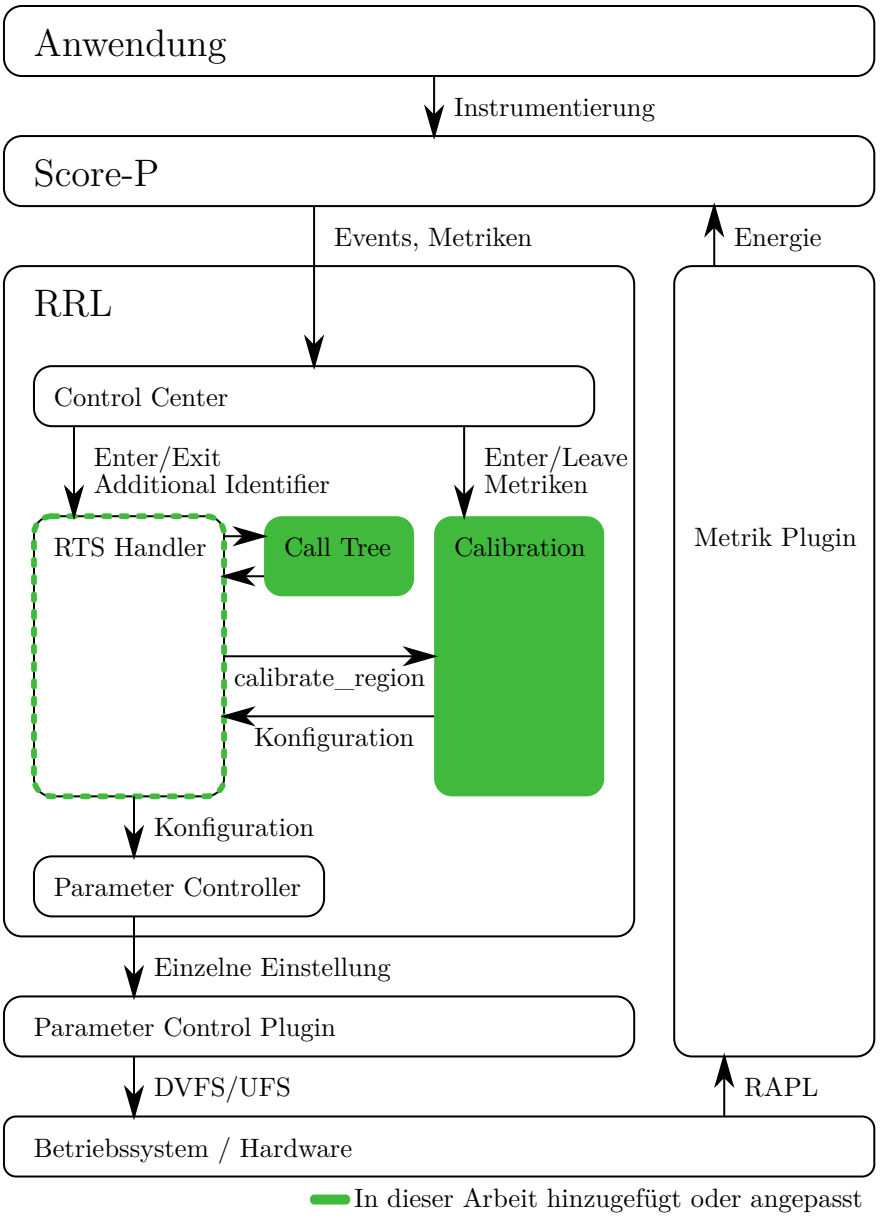


Abbildung 4.1: Prinzipieller Aufbau der RRL, mit den Anpassungen für diese Arbeit.

Metriken an die RRL weitergeleitet. So können auch die in Kapitel 2.1.1, S. 13, beschriebenen Energiemessungen an die RRL weitergereicht werden.

Das *Control-Center* filtert die ankommenden Events. Da nach Annahme 1 die untersuchten Programme phasenweise homogen sind, können alle Events, die nicht vom Master-Thread kommen, verworfen werden. Darüber hinaus können bestimmte Regionen, die sich aus technischen Gründen nicht von Score-P ignorieren lassen, hier gefiltert werden. Die übrig gebliebenen Informationen werden danach sowohl an den *RTS-Handler* als auch an das gerade aktive *Calibration-Modul* weitergeleitet. Die unterschiedlichen Calibration-Module und deren Funktion werden in Kapitel 4.1.3 beschrieben. Im Wesentlichen sind sie für das Aufzeichnen der Daten sowie das Finden einer optimalen Frequenz für den Prozessor zuständig.

Der Name *RTS-Handler* leitet sich von der Runtime-Situation (RTS) ab. Eine RTS wird in READEX genutzt, um eine Region eindeutig zu identifizieren. Der *RTS-Handler* nimmt eine Konfiguration wahlweise vom Calibration-Modul oder vom Call-Tree entgegen. Der Call-Tree, welcher in Kapitel 4.1.2 beschrieben wird, speichert die Konfiguration des Calibration-Moduls, wenn dieses signalisiert, dass die Calibration abgeschlossen ist.

Die Konfiguration wird schließlich an den *Parameter-Controller* weitergegeben. Dieser verwaltet die einzelnen *Parameter-Control-Plug-ins*. Diese Plug-ins nutzen Betriebssystemschnittstellen, Bibliotheken oder den direkten Zugriff auf die Hardware, um verschiedene Einstellungen des Compute-Knotens zu ändern.

Um die verschiedenen Frequenzen ändern zu können, wird *libfreqgen* (Schöne, 2020) verwendet. Die Bibliothek prüft, welche Schnittstellen vorhanden sind, und bietet ein einheitliches Interface, diese zu nutzen. Als Schnittstellen werden likwid (Treibig et al., 2010), x86_adapt (Schöne und Molka, 2014), msr-save („libMSR library and msr-safe kernel module“, 2013) oder das Linux Sysfs Cpufreq System unterstützt. Zwar ist das Linux Sysfs Cpufreq System auf allen Linux-Systemen verfügbar, allerdings erlauben Kernel vor 5.6 nur das Anpassen der Corefrequenz, nicht aber der Uncorefrequenz (Larabel, 2020). Bei älteren Kerneln ist es daher notwendig, die entsprechenden MSRs im Prozessor direkt zu schreiben, vergl. Kapitel 2.1.2, S. 15.

4.1.2 Call-Path oder Call-Tree

Wie schon in Kapitel 2.1.4, S. 20, geschrieben, besteht eine RTS aus dem Call-Path der Programmfunktionen sowie den Additional Identifiern. In READEX nutzt die RRL ein Tuning-Modell, aus dem für die unterschiedlichen RTS verschiedene Konfigurationen abgerufen werden können. Das Tuning-Modell wird während der Designtime-Analysis (DTA) von PTF generiert (vergl. Kapitel 2.1.4, S. 20). Für jede RTS wird die Konfiguration bestimmt, die am wenigsten Energie benötigt. Gleichzeitig wird bestimmt, für welche RTS die Konfigurationen geändert werden und welche ignoriert werden.

Da das Finden der optimalen Konfiguration in dieser Arbeit automatisiert wird, muss auch die Einordnung, ob eine RTS relevant ist oder ignoriert werden soll, während des Einsatzes des Programms erfolgen. Deshalb wird das Konzept des Call-Paths zu einem Aufrufbaum (engl. call tree) erweitert, der zur Laufzeit aufgebaut wird. Funktionen und Additional Identifier werden als Knoten im Call-Tree repräsentiert. Für jede Funktion, die betreten wird, wird geprüft, ob sie schon als Kindknoten des aktuellen Knotens vorhanden ist. Wenn das nicht der Fall ist, wird ein neuer Knoten erzeugt und die Laufzeit der Funktion bestimmt. Gleiches gilt beim Auftreten von Additional Identifiern.

Damit lassen sich durch Traversieren vom aktuellen Knoten zur Wurzel des Baumes klassische RTS im Sinne von READEX erzeugen. Gleichzeitig werden für jeden Knoten Metainformationen wie Laufzeit oder der Zustand gespeichert, wie in Tabelle 4.1 und im Folgenden beschrieben. Nur der RTS-Handler ändert den Zustand eines Knotens.

Wenn ein Knoten angelegt wird, ist er im Zustand `unknown`, also unbekannt. Mit dem Anlegen des Knotens wird die Zeitmessung der Laufzeit gestartet und der Zustand auf

Zustand	Beschreibung	Folgender Zustand
`unknown`	Initialer Zustand des Knotens	`measure_duration`
`measure_duration`	Die Laufzeit der Funktion, die der Knoten repräsentiert, wird gemessen	`calibrate`, `known`
`calibrate`	Der Knoten wird kalibriert	`calibrate`, `known`
`known`	Der Knoten besitzt nach der Calibration eine bekannte Konfiguration oder muss nicht kalibriert werden und besitzt keine Konfiguration	`known`

Tabelle 4.1: Unterschiedliche mögliche Zustände eines Knotens.

`measure_duration` geändert. Wird die zugehörige Funktion verlassen, wird die Zeitmessung angehalten, und es wird entschieden, ob der Knoten optimiert werden soll oder nicht. Abbildung 4.2 gibt eine Übersicht über den Entscheidungsprozess.

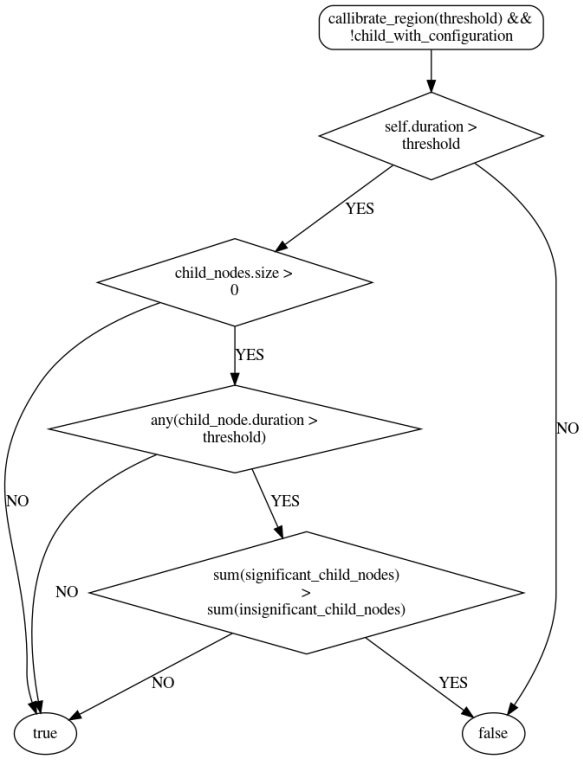


Abbildung 4.2: Entscheidungsprozess, ob ein Knoten im Call-Tree der RRL optimiert werden soll.

Als Erstes wird evaluiert, ob die Laufzeit des aktuellen Knotens länger ist als ein gegebener Grenzwert. Wenn das nicht der Fall ist, muss der Knoten nicht optimiert werden. Der Grenzwert für die Laufzeit einer Funktion wurde in READEX mit 100 ms festgesetzt. Für die Wahl dieser Zeit ist sowohl der Overhead von Score-P und der RRL als auch das Aktualisierungsintervall von RAPL relevant gewesen. Wie in Kapitel 2.1.1, S. 13, beschrieben, werden die RAPL-Register ungefähr alle 0,001 s aktualisiert. Allerdings sind diese Aktualisierungen nicht synchron mit dem Beginn oder Ende einer Funktion. Damit können bei einer Funktion mit exakt 100 ms Laufzeit nur 99 Messpunkte vorliegen, und es ergibt sich ein entsprechender Fehler in der Energiemessung. Der fehlende Messpunkt kann aber bei einer Laufzeit von 100 ms vernachlässigt werden.

Nach der Laufzeit wird geprüft, ob ein Knoten Kinder hat, also ob weitere Funktionen von der aktuellen Funktion aufgerufen werden. Wenn das nicht der Fall ist, kann der Knoten optimiert werden. Wenn es Kinder gibt, muss geprüft werden, ob eines der Kinder eine ausreichende Laufzeit hat und sich zur Optimierung eignen würde. Trifft das nicht zu, muss noch abschließend geklärt werden, ob durch das Optimieren der Kinder Optimierungspotenzial verloren geht. Ein Beispiel: Vier Kindknoten haben eine Laufzeit von je 100 ms, der aktuelle Knoten eine Laufzeit von 1 s. Würden die Kindknoten optimiert, würden deren 400 ms berücksichtigt, die restlichen 600 ms würden außer Acht gelassen. Es lohnt also mehr, den aktuellen Knoten zu optimieren, als die Kindknoten.

Wenn ein Knoten optimiert wird, wird er in den Zustand `calibrate` bewegt und das Calibration-Modul wird involviert. Wird die Funktion wieder verlassen, wird die Rückgabe des Calibration-Moduls geprüft. Wenn eine weitere Optimierung notwendig ist, bleibt der Knotenzustand erhalten, ansonsten wechselt der Knoten in den Zustand `known`, und die ermittelte Konfiguration wird im Knoten gespeichert. Wird ein Knoten nicht optimiert, wird er in den Zustand `known` bewegt und es wird eine leere Konfiguration gespeichert. In beiden Fällen kann beim nächsten Besuch des Knotens die Konfiguration geladen und angewendet werden. Wenn die Konfiguration, die angewendet werden soll, leer ist, passiert nichts.

Beim Verlassen einer Region wird, je nach Einstellung, die aktuelle Konfiguration beibehalten oder die Konfiguration der Elternknoten wiederhergestellt.

4.1.3 Calibration-Module

Um Daten für das Training der Machine-Learning-Mechanismen zu sammeln und die trainierten Modelle anzuwenden, wurde die RRL im Rahmen dieser Arbeit um die sogenannten Calibration-Module erweitert.

Zum Aufzeichnen gibt es drei unterschiedliche Module: Das erste erzeugt eine Referenzmessung der Laufzeit. Das zweite gibt am Beginn jeder Region, die optimiert werden soll, eine zufällige, unabhängige und gleichverteilte Kombination von Core- und Uncorefrequenz an den RTS-Handler. Gleichzeitig wird der aktuelle Energieverbrauch des Programmes an jedem Programm-Event mithilfe eines Metrik-Plug-ins von Score-P gemessen. Ziel ist es, das Verhalten der Funktion im Hinblick auf den Energieverbrauch zu charakterisieren. Dabei können unterschiedliche Score-P Metrik-Plug-ins (Schöne, 2017, S. 85 ff.) zum Messen der Energie zum Einsatz kommen.

Das dritte Modul ist zum Konfigurieren und Auslesen der unterschiedlichen Performance-Counter zuständig. Dazu werden am Beginn einer Region zufällig verschiedene Performance-Events ausgewählt und diese mithilfe der Counter bei einer festgelegten Kombination von Core- und Uncorefrequenz vermessen. Die Idee dahinter ist, dass gleiche Regionen die gleiche Charakteristik aufweisen. Damit können bei jedem Auftreten einer Region andere Events gemessen werden. So kann mit weniger Programmläufen ein wesentlich größerer Raum an Events abgedeckt werden als in der Arbeit von Chadha und Gerndt (2019), da hier die unterschiedlichen Events in verschiedenen Läufen des Programmes gemessen werden. Zum Auslesen der Counter werden PAPI und x86_adapt verwendet (vergl. Kapitel 2.1.3, S. 18).

Zum Anwenden des finalen Machine-Learning-Modells gibt es ein weiteres Modul. Dieses verwendet die TensorFlow C API (Martín Abadi et al., 2015, Version 1.14.0), um ein trainiertes neuronales Netz zu laden. Jedes Mal, wenn der RTS-Handler entscheidet, dass eine Kalibrierung notwendig ist, konfiguriert das Modul die entsprechenden Performance-Counter mit den Events, die zum Training verwendet wurden und die mittels Merkmalsauswahlverfahren bestimmt wurden. Gleichzeitig werden die Core- und Uncorefrequenzen, bei denen die Events gemessen werden sollen, an den RTS-Handler zurückgegeben. Wenn die Messung beendet ist, werden die Daten auf die Laufzeit normiert und die Eventrate wird ermittelt. Anschließend werden die Eventraten um die gespeicherten Mittelwerte und Standardabweichungen bereinigt. Schließlich werden diese normierten Eventraten zusammen mit

unterschiedlichen Core- und Uncorefrequenzen an das Machine-Learning-Modell gegeben. Die Kombination, bei der laut Modell der geringste Energieverbrauch zu erwarten ist, wird an den RTS-Handler zurückgegeben, und das Training wird als beendet markiert.

Andere Module

Das Interface der Calibration-Module ist so ausgelegt, dass es einfach um neue Module ergänzt werden kann. Ein Beispiel ist der in Kapitel 2.1.4, S. 22, beschriebene Reinforcement-Ansatz (Gocht et al., 2019). Dieser wurde ebenfalls von mir entwickelt und verwendet die gleiche Infrastruktur. Durch diese Erweiterbarkeit ist es möglich, unterschiedliche Ansätze zu entwickeln und zu testen.

4.2 Testsystem

Das verwendete Testsystem entspricht hinsichtlich der verwendeten Hardware einem Compute-Knoten des HPC-Systems Taurus der TU Dresden. Aussagen, die für dieses System getroffen werden, lassen sich also auf das HPC-System verallgemeinern. Das Testsystem wurde mit LMG 450 instrumentiert (vergl. Kapitel 2.1.1, S. 14), um detaillierte Analysen der Leistungsaufnahme und des Energieverbrauchs durchführen zu können.

Das Testsystem verwendete zwei Intel® Xeon® Processor E5-2680 v3 Prozessoren mit je 12 Kernen. Jeder Kern besitzt wiederum einen 32 KB großen L1-Daten-Cache sowie einen 32 KB L1-Befehls-Cache. Der L2-Cache jedes Kerns wird für Daten und Befehle genutzt und ist 256 KB groß. Darüber hinaus besitzt der Prozessor einen 30 MB großen L3-Cache, der über die Kerne aufgeteilt ist (Intel, 2019b, S. 2-23 f.; Intel, 2017, S. 11). Die Basisfrequenz des Prozessors liegt bei 2,5 GHz, die maximale Turbofrequenz bei 3,3 GHz. Neben der Basis- und der Turbofrequenz für normale Befehle hat Intel mit der Haswell-Generation eine Basis- und Turbofrequenz für Programme eingeführt, die den AVX-Befehlssatz nutzen (Hackenberg et al., 2015). Die Advanced-Vector-Extensions(AVX)-Befehle erlauben das gleichzeitige Bearbeiten mehrerer Daten. So können zum Beispiel mittels AVX-2 vier 64-Bit- oder acht 32-Bit-Floating-Point-Operationen gleichzeitig ausgeführt werden. Die AVX-Basisfrequenz liegt bei 2,1 GHz, die maximale AVX-Turbofrequenz für 12 Kerne bei 2,8 GHz (Intel, 2017, S. 11). Aufgrund der in Kapitel 2.1.2 beschriebenen Beschränkungen des ACPI-Standards auf 16 P-States sind die Turbofrequenzen nicht einzeln ansteuerbar, sondern in der Frequenz 2,501 GHz zusammengefasst und werden vom Prozessor frei gewählt. Darüber hinaus kann sich der Prozessor über die via ACPI spezifizierte Frequenz hinwegsetzen, wenn er zu warm wird oder die Leistungsaufnahme die Spezifikation übersteigt (Intel, 2017, S. 11).

Dem Prozessor stehen 128 GB Hauptspeicher vom Typ Micron MTA36ASF2G72PZ-2G1A2 zur Seite. Dabei handelt es sich um 8 Module zu je 16 GB DDR4-2133 Speichern mit ECC, die gleichmäßig über die beiden Prozessoren und die vorhandenen Speicherkanäle verteilt sind.

4.2.1 Architektur

Im Folgenden wird der Aufbau der beiden verwendeten Prozessoren kurz umrissen. Damit lassen sich die verschiedenen gemessenen Performance-Events besser einordnen. Eine ausführliche Beschreibung der Prozessoren kann den entsprechenden Intel-Handbüchern entnommen werden (Intel, 2015; Intel, 2019b).

Abbildung 4.3 zeigt den schematischen Aufbau des verwendeten Haswell-Prozessors aus Intel (2015, S. 10). Die 12 Kerne sind in zwei Blöcke zu acht und vier Kernen aufgeteilt. Innerhalb des Blocks sind die Kerne über die sogenannten *Caching-Agents* (CBox) mit zwei Ringbussen und dem *Last-Level-Cache* (LLC) verbunden. Die beiden Ringe kommunizieren in entgegengesetzte Richtungen. Über *Ring-Transfers* (SBox) kommunizieren die Ringe der

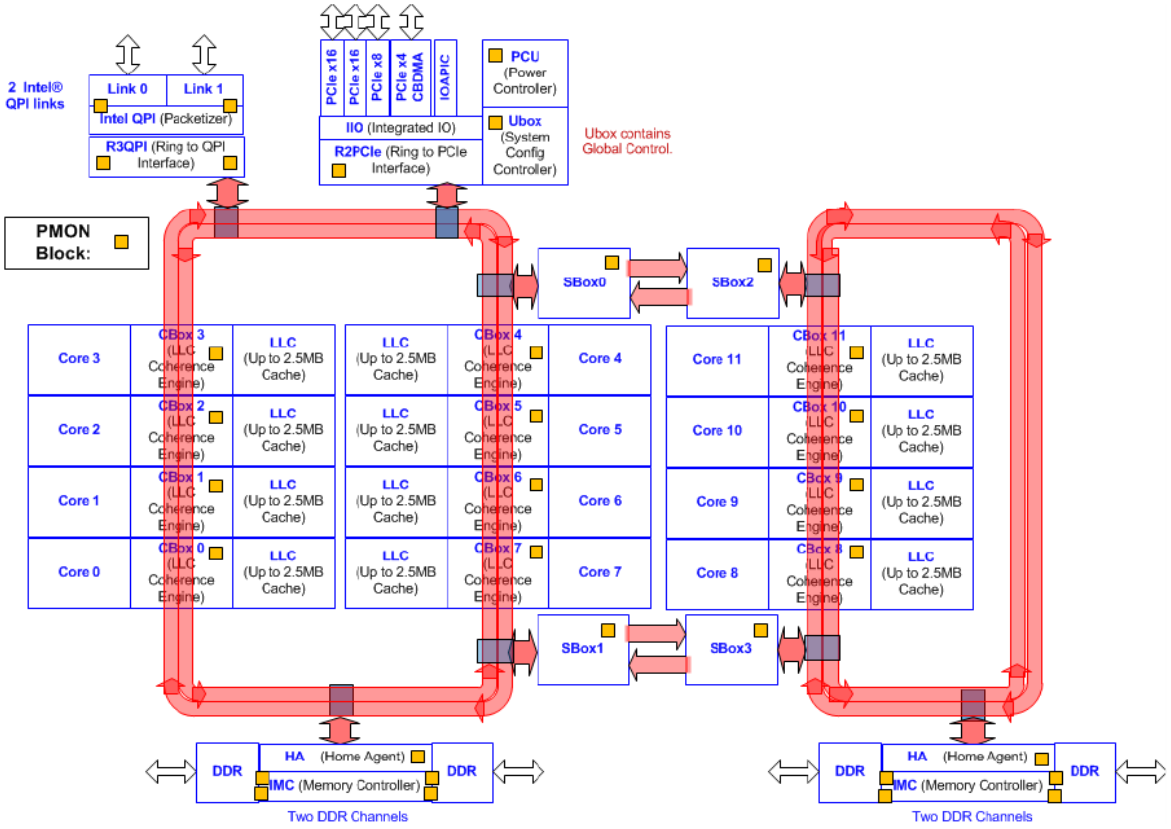


Abbildung 4.3: Schematischer Aufbau der verwendeten Intel-Haswell-Prozessoren aus Intel (2015, S. 10). Jeder PMON-Block stellt Performance-Counter zum Zählen von Performance-Events bereit. Es sind nur die PMON-Blöcke im Uncore dargestellt.

verschiedenen Blöcke miteinander. An den Ringen jedes Blocks hängt je ein *Home-Agent* (HA), der zusammen mit dem *Memory-Controller* (IMC) die Kommunikation zum Hauptspeicher übernimmt.

Nur an den Ringen des ersten Blocks mit acht Kernen hängt ein *Ring-to-QPI-Interface* (R3QPI). Intels *Quick-Path-Interconnect* (QPI) wiederum, das am R3QPI angeschlossen ist, dient als Schnittstelle zum zweiten installierten Prozessor. Ebenfalls am ersten Ring können über das *Ring-to-PCIe-Interface* (R2PCIe) PCIe-Geräte wie Netzwerkkarten angeschlossen werden. An der gleichen Schnittstelle hängen auch die UBox, die als Systemkonfigurationskontroller ("[...] system configuration controller [...]" Intel, 2015, S. 30) dient, und die *Power-Control-Unit* (PCU), die sich um die Energie- und Temperaturverwaltung des Prozessors kümmert.

Die in Abbildung 4.3 gezeigten Performance-Monitoring-Blöcke (PMON-Blöcke), auch Boxen genannt, sind Einheiten im Prozessor, die die Performance-Counter zum Messen der Performance-Events bereitstellen. Neben den dargestellten PMON-Blöcken im Uncore besitzt noch jeder Core einen PMON-Block. Es zeigt sich, dass alle wesentlichen Komponenten eines Prozessors einen PMON-Block besitzen.

Abbildung 4.4 zeigt einen schematischen Aufbau eines einzelnen Kerns der Intel-Haswell-Architektur. Eine Instruktion wird aus dem L2-Cache in den L1-Instruction-Cache geladen. Die *Branch-Prediction-Unit* (BPU) versucht die nächste benötigte Instruktion vorherzusagen und veranlasst das Laden aus dem L1-Instruction-Cache in die *Instruction-Queue*. Dort wird die Instruktion decodiert und in der *Instruction-Decode-Queue* (IDQ) gespeichert. Die IDQ kann auch über den Micro-Operation-Cache (uOp Cache) von der BPU bestückt werden. Der uOp Cache dient der Zwischenspeicherung bereits decodierter Befehle.

Aus der IDQ gelangen die Instruktionen in die Out-of-Order-Engine des Prozessors, die

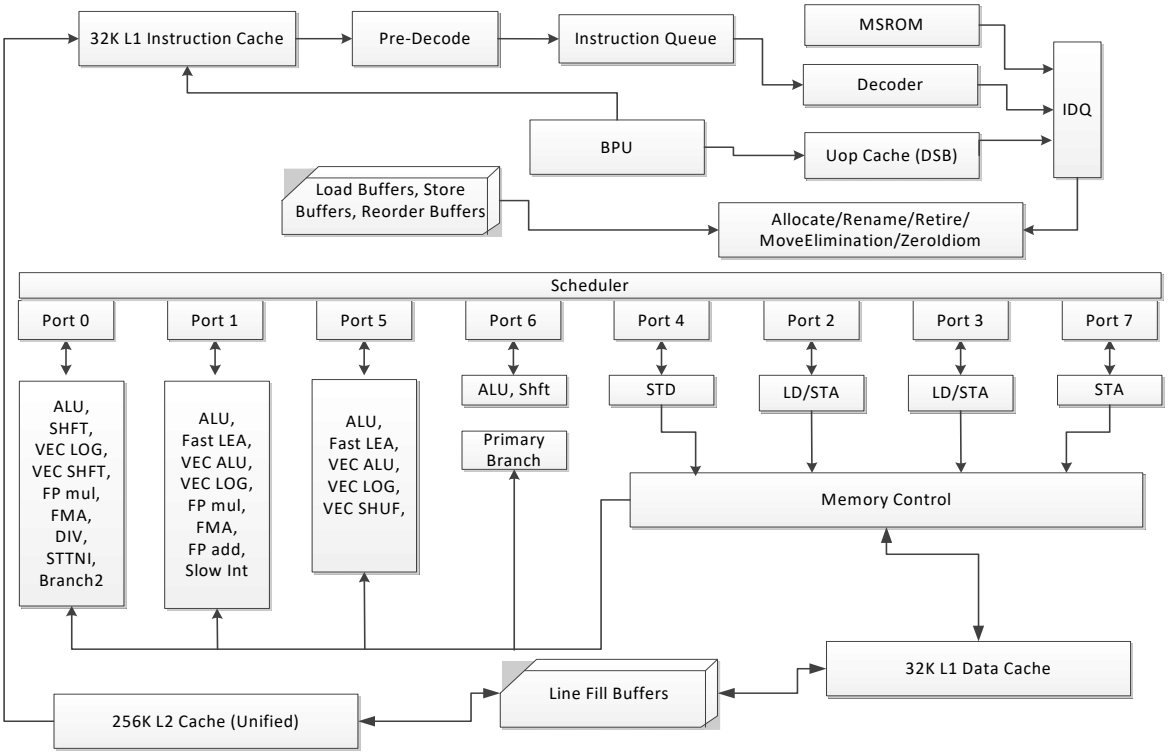


Abbildung 4.4: Schematischer Aufbau eines Kerns der verwendeten Intel-Haswell-Prozessoren aus Intel (2019b, S. 2-23).

unter anderem aus dem Renamer, Scheduler und Reorder-Buffer besteht. Von dort können die Instruktionen an die einzelnen Ports weitergeleitet werden. Dort findet die eigentliche Berechnung oder das Laden und Speichern der Daten statt.

Die verschiedenen Teile des Prozessors haben unterschiedlich viele Performance-Counter. Beim Prozessorkern unterscheidet sich die Anzahl der Counter, die verfügbar sind, je nachdem, ob Hyperthreads im Prozessor beim Starten des Systems aktiviert oder deaktiviert wurden. Wurde Hyperthreading beim Starten aktiviert, stehen vier Counter pro Hyperthread zur Verfügung. Auch wenn die Hyperthreads nach dem Start durch das Betriebssystem deaktiviert werden, ändert sich diese Zahl nicht. Wurde das Hyperthreading bereits beim Start des Systems deaktiviert, stehen pro Thread und damit pro Kern acht Counter zur Verfügung (Intel, 2019c, S. 3504, 18-48 Vol. 3B). Dazu kommen noch drei sogenannte Fixed Counter, bei denen das gezählte Performance-Event nicht geändert werden kann:

```
INST_RETIRES       :ANY
CPU_CLK_UNHALTED   :THREAD
                   :CORE
                   :THREAD_ANY
CPU_CLK_UNHALTED   :REF_TSC
```

Diese zählen die zurückgestellten Instruktionen, die Anzahl der Kern/Thread-Zyklen, in denen der Kern/Thread nicht angehalten war, und Time-Stamp-Counter(TSC)-Zyklen.

Da vier die mit allen Einstellungen kompatible Anzahl an Countern ist, werden nur diese im Weiteren verwendet. Darüber hinaus werden die Fixed Counter wie gewöhnliche Counter behandelt.

Tabelle 4.2 zeigt die pro PMON-Block verfügbaren Counter. Da der R3QPI vor allem als Brücke zwischen dem QPI-Interface und dem Ring dient und selbst keine Daten verarbeitet, wird er in der weiteren Betrachtung nicht berücksichtigt. Gleiches gilt für R2PCIe, da die

PMON-Block	Counter/PMON-Block
CBox	4
HA	4
IMC	4 (per Channel)
PCU	4 + 2
QPI	4 (per Port)
R2PCIe	4
R3QPI	3 (per Link)
SBox	4

Tabelle 4.2: Anzahl der verfügbaren Performance-Counter pro PMON-Block. Die meisten PMON-Blöcke sind mehrfach vorhanden, wie in Abbildung 4.3 zu sehen. Die Zahl hinter dem + markiert fixe Einträge (Intel, 2015, S. 12).

PCI-Geräte nicht optimiert werden.

4.2.2 Bestimmung des Offsets zur Energiemessung mit RAPL

Wie in Kapitel 2.1.1 beschrieben, ist eine Kombination von RAPL mit einem statischen Offset für diese Arbeit zweckmäßig. Der Offset wurde durch das Bestimmen des Gesamtenergieverbrauchs verschiedener Programme bestimmt. Als externes Messgerät wurde das LMG 450 (vergl. Kapitel 2.1.1, S. 14) verwendet. Aus der Laufzeit der Programme und der Differenz zwischen Energiemessung mit RAPL und dem externen Messgerät ergibt sich ein Offset von 70 W für dieses Testsystem. Dieser deckt unter anderem den Energieverbrauch des Mainboards und der Netzwerkkarte ab.

4.3 Verwendete Benchmarks zur Erzeugung der Datengrundlage

Um die Datengrundlage für die Merkmalsauswahl und das Training zu generieren, werden unterschiedliche Benchmarks verwendet. Einige dieser Benchmarks, wie Stream (McCalpin, 1995) oder die Matrix-Multiplikation, besitzen spezifische Kernel, die eine bestimmte Arbeitslast wiederholt ausführen. Diese Kernel werden zum Teil mit unterschiedlichen Konfigurationen genutzt, um verschiedene Arbeitslasten zu generieren. Andere Benchmarks bestehen aus Anwendungen oder Teilen von Anwendungen. Damit soll das Verhalten eines Compute-Knotens unter möglichst realitätsnahen Bedingungen untersucht werden.

Die in dieser Arbeit vorgestellten Merkmalsauswahlverfahren und neuronalen Netzwerke sollen mit zwei unterschiedlichen Datensätzen analysiert werden. Als Erstes wird ein modifizierter Stream-Benchmark verwendet (McCalpin, 1995), der vier unterschiedliche Kernel nutzt. Für den Benchmark kann die verwendete Speichermenge konfiguriert werden. Die unterschiedlichen Eventraten der Performance-Events, die je nach verwendeter Speichermenge auftreten, können nach der Speichermenge geordnet und damit analysiert werden. So lässt sich zum Beispiel analysieren, wie sich eine Eventrate verhält, wenn auf Daten im L1-Cache des Prozessors zugegriffen wird oder wenn Daten aus dem DRAM geladen werden müssen.

Als Zweites wird eine Reihe von Benchmarks verwendet, die nicht notwendigerweise Gemeinsamkeiten haben. Dabei werden sowohl anwendungsbasierte als auch kernelbasierte Benchmarks verwendet. Diese Benchmarks können durch sehr unterschiedliche Flaschenhälse limitiert sein.

Während mit dem ersten Datensatz eine ausführliche Analyse der durch die Merkmalsauswahlverfahren gewählten Performance-Events möglich ist, ist der zweite Datensatz vielfältiger und damit eine bessere Grundlage für das Training des finalen neuronalen Netzes. Beide Datensätze werden im Folgenden näher beschrieben.

Für alle Experimente wurde die GCC Compiler Suite, bestehend aus `gcc`, `g++` sowie `gfortran`, in der Version 8.2.0 verwendet. Darüber hinaus wurden in allen Fällen die folgenden Flags genutzt:

- `-fopenmp`
- `-O3`
- `-march=haswell`
- `-mcmodel=medium`

Zur Instrumentierung wurde die in READEX verwendete Entwicklungsversion von Score-P verwendet, welche auf Score-P 4.0 basiert. Für die Messung der Core-Events wurde PAPI in der Version 5.6.0 verwendet, für die Uncore-Events x86_adapt.

Um die Menge der generierten Daten zu erhöhen, wurde die minimale Laufzeit einer relevanten Funktion von 100 ms, die in READEX festgesetzt wurden, auf 50 ms reduziert. Wie in Kapitel 4.1.2, S. 55, beschrieben, wurde dieser Wert unter der Berücksichtigung des Overheads durch RRL und Score-P und des Messintervalls von RAPL festgesetzt. Durch die Reduktion der minimalen Laufzeit als Kriterium, um zu entscheiden, ob eine Funktion optimiert wird oder nicht, steigt der Fehler der Daten durch den Overhead und das RAPL-Messintervall. Gleichzeitig steigt die Menge der Funktionen, für die, durch die Calibration-Module, Daten gesammelt werden. Da mehr Daten die Trainingsergebnisse der später verwendeten Machine-Learning-Methoden wieder verbessern können, ist die Wahl der minimalen Laufzeit eine Abwägungsfrage, bei der hier 50 ms festgesetzt wurden.

4.3.1 Datensatz 1: Der Stream-Benchmark

Der Stream-Benchmark (McCalpin, 1995) wird genutzt, um die Speicherbandbreite der verschiedenen Speicherhierarchien zu vermessen. Dazu wird die Dauer unterschiedlicher Kernel über eine vorgegebene Arraygröße und damit angegebene Datenmenge gemessen. Daraus wird die Speicherbandbreite in Byte/s berechnet. Es kommen die folgenden Kernel mit den Vektoren a, b und c zum Einsatz:

- Copy: $c = a$
- Scale: $b = \text{scalar} \cdot c$
- Add: $c = a + b$
- Triad: $c = b + \text{scalar} \cdot c$

Die einzelnen Kernel sind über eine `for`-Schleife realisiert und per OpenMP parallelisiert, wie beispielhaft in Quelltext 4.1 zu sehen. Besonders bei kleinen Datenmengen, die in den L1-, L2- oder L3-Cache passen, laufen die Kernel zu kurz, um später verlässliche Aussagen über den Energieverbrauch treffen zu können. Deshalb wurde der Benchmark modifiziert und eine weitere Schleife hinzugefügt, die die gleiche Operation mehrfach ausführt, wie in Quelltext 4.2 zu sehen. Darüber hinaus wurde der Benchmark um Score-P User-Regionen erweitert, um die relevanten Regionen vermessen zu können, ohne die OpenMP selbst instrumentieren zu müssen. Dadurch wird der Overhead durch Score-P begrenzt.

Für die Bandbreite des verwendeten Systems ergibt sich dabei die Kurve, die in Abbildung 4.5 zu sehen ist. Die unterschiedlichen Datenmengen werden für jeden Kernel im Folgenden als eine Region gewertet. Die Grenze zwischen L3-Cache und DRAM ist in den Messungen klar zu erkennen. Entgegen der Erwartung ist die Speicherbandbreite im L1- bzw. im L2-Cache kleiner als im L3-Cache. Wie in Quelltext 4.2 zu sehen, beinhaltet die

```
    times[0][k] = mysecond();
    [...]
#pragma omp parallel for
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        c[j] = a[j];
    times[0][k] = mysecond() - times[0][k];
```

Quelltext 4.1: Originale Version des Stream-Copy-Kernels.

```
    times[0][k] = mysecond();
    SCOREP_USER_REGION_BEGIN(copy, "copy", SCOREP_USER_REGION_TYPE_COMMON)
    for (i = 0; i < NTIMES2; i++)
    {
    [...]
#pragma omp parallel for
        for (j = 0; j < STREAM_ARRAY_SIZE; j++)
            c[j] = a[j];
#endif
    }
    SCOREP_USER_REGION_END(copy)
    times[0][k] = mysecond() - times[0][k];
```

Quelltext 4.2: Für diese Arbeit modifizierte Version des Stream-Copy-Kernels. Um die innere `for`-Schleife wurde eine weitere `for`-Schleife gelegt, um die Laufzeit für kleine Arrays `a` und `c` zu erhöhen. Gleichzeitig wurde eine Score-P User-Instrumentierung hinzugefügt, um auf eine Instrumentierung der OpenMP-Regionen durch Score-P verzichten zu können und den Overhead durch Score-P zu begrenzen.

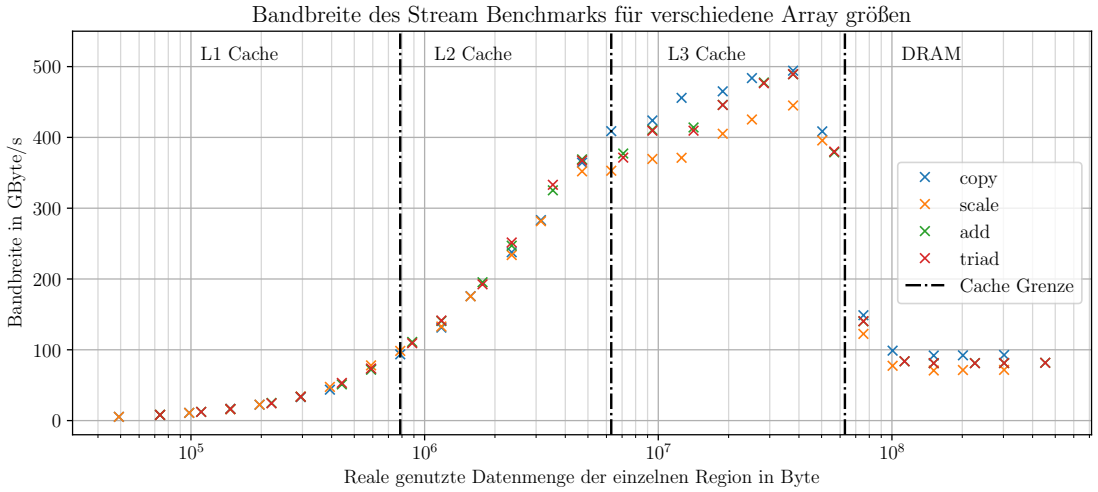


Abbildung 4.5: Speicherbandbreite der vier modifizierten Stream-Kernel `copy`, `scale`, `add` und `triad`, die mittels Score-P User-Instrumentierung und der RRL vermessen wurden. Gemessen wurde auf dem in Kapitel 4.2, S. 58, beschriebenen Testsystem. Die theoretische Grenze der Daten, die in einen Cache passen, ist dargestellt. Für die Messung wurde die Core- und Uncorefrequenz auf 2,5 GHz respektive 3,0 GHz gesetzt. Entgegen der Erwartung ist die Speicherbandbreite im L1- bzw. im L2-Cache kleiner als im L3-Cache. Wie in Schöne et al. (2008) beschrieben ist der OpenMP-Overhead ursächlich.

Zeitmessung das Verteilen der Daten und Parallelisieren der Berechnung durch OpenMP. Da bei kleinen Datenmengen der Overhead durch OpenMP im Verhältnis groß ist, sinkt die gemessene Bandbreite. Durch eine Aufteilung der Daten vor der Ausführung der parallelen Region, wie in Schöne et al. (2008) beschrieben, lässt sich das Problem adressieren. Gleichzeitig weicht man damit vom eigentlichen Stream-Benchmark ab, weswegen hier darauf verzichtet wird.

Um eine Überrepräsentation des DRAMs in den späteren Trainingsdaten zu vermeiden, werden im Folgenden nur Arraygrößen bis 8 MB pro Array und Prozessorkern verwendet.

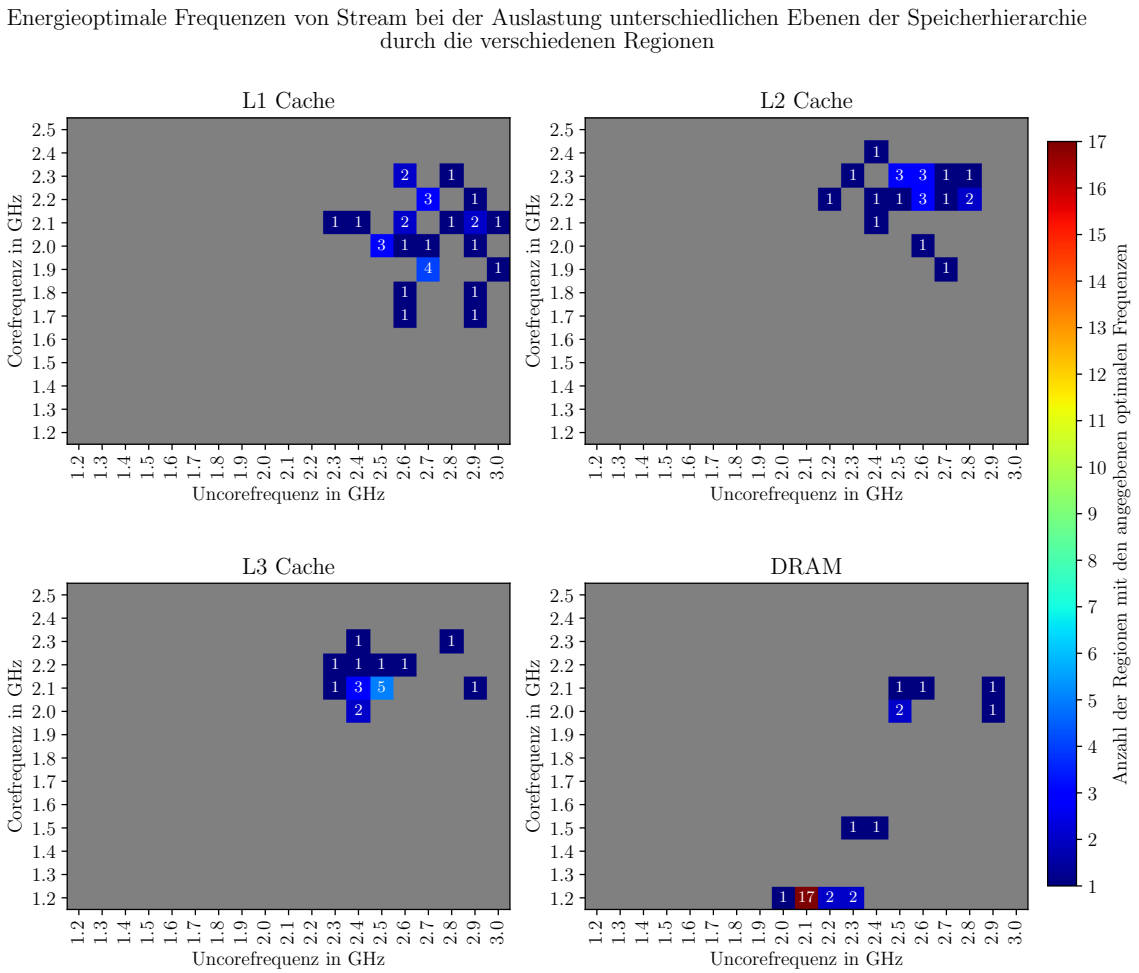


Abbildung 4.6: Anzahl von Regionen der vier Stream-Kernel `copy`, `scale`, `add` und `triad` mit der angegebenen optimalen Frequenz abhängig davon, in welche Speicherebene die verwendeten Daten des Kernels passen. Gemessen wurde auf dem in Kapitel 4.2, S. 58, beschriebenen Testsystem. Die Zahlen geben an, wie viele Regionen die jeweilige optimale Frequenz haben. Daten, die in den L1-, L2- und L3-Cache passen, profitieren von einer hohen Core- und Uncorefrequenz, während für Daten, die in keinen der Caches passen, eine niedrige Core- und eine mittlere Uncorefrequenz vorteilhaft sind. Beim L1-Cache ist die optimale Frequenz nicht eindeutig.

Abbildung 4.6 gibt eine Übersicht über die optimalen Frequenzen der Regionen, je nachdem, in welche Ebene der Speicherhierarchie die Region fällt. Die Zahlen geben an, wie viele Regionen die jeweilige optimale Frequenz haben. Zu beachten ist dabei, dass `copy` und `scale` zwei Arrays verwenden, während `add` und `triad` drei Arrays verwenden und damit mehr Daten bewegen. In Abbildung 4.6 zeigt sich, dass Daten, die in den L2- und L3-Cache passen, stark von einer hohen Core- und Uncorefrequenz profitieren. Sobald auf den DRAM zurück-

gegriffen werden muss, sind eine niedrige Core- und eine mittlere Uncorefrequenz besser. Zu beachten ist, dass es, wie bei der Bandbreite in Abbildung 4.5 zu sehen, auch bei den optimalen Frequenzen zu Übergangseffekten zwischen den Caches sowie zum DRAM kommen kann.

4.3.2 Datensatz 2: Eine Sammlung verschiedener Benchmarks

Als weiterer Datensatz werden Benchmarks aus SPEC OpenMP 2012 (Müller et al., 2012) sowie die NAS Parallel Benchmarks (Bailey et al., 1991)[1] verwendet. Dazu wird eine Auswahl von Stream-Kerneln und Matrixmultiplikationskerneln hinzugefügt. Die Benchmarks wurden danach selektiert, dass sie Annahme 1, S. 10, und Annahme 2, S. 10, nicht verletzen.

SPEC OpenMP 2012 Konkret wurden die folgenden Benchmarks aus SPEC OpenMP 2012 ausgewählt (Whitney, 2012):

- md - Molecular Dynamics, simuliert ionisierte Atome via Coulomb-Interaktion
- nab - Molecular Modeling, Berechnungen von unstrukturierter Molekulardynamik bis zu linearer Algebra
- bt331 - Computational Fluid Dynamics, simuliert eine Computational-Fluid-Dynamics-Anwendung, nutzt einen Block-Tri-diagonal-Solver
- ilbdc - Fluid Mechanics, Teil eines 3-D-Lattice-Boltzmann-Strömungslösers, zeichnet sich vor allem durch sparse Daten aus
- fma3d - Mechanical Response Simulation, simuliert nicht elastische, flüchtige und dynamische Reaktionen von dreidimensionalen Strukturen
- swim - Weather, Wettervorhersagebenchmark

Alle Benchmarks wurden in der Referenzgröße ausgeführt.

NAS Parallel Benchmarks Aus den NAS Parallel Benchmarks (NPB) wurden die folgenden Benchmarks verwendet:

- bt - Block Tri-diagonal Solver, Pseudoanwendung
- lu - Lower-Upper Gauss-Seidel Solver, Pseudoanwendung
- sp - Scalar Penta-diagonal Solver, Pseudoanwendung
- cg - Conjugate Gradient, unregelmäßige Speicherzugriffe und Kommunikation
- mg - Multi-Grid auf einer Sequenz von Netzen, kurze und weite Kommunikationsabstände, speicherintensiv

Die Benchmarks wurden alle mit der Größe D konfiguriert. Dabei ist zu beachten, dass es sich bei `bt` aus NPB um den gleichen Code wie in `bt331` aus SPEC OpenMP handelt. Allerdings unterscheidet sich die verwendete Datenmenge.

Alle Benchmarks, sowohl die aus dem SPEC OpenMP als auch die aus dem NPB Set gewählten, wurden mit dem Score-P GCC Compiler Plug-in instrumentiert. Danach wurden die Benchmarks und die Laufzeit der einzelnen Funktionen vermessen. Funktionen, die eine zu kurze Laufzeit, also kleiner als 50 ms, hatten, wurden beim erneuten Kompilieren ignoriert.

[1] Version 3.3.1 OpenMP.

Matrixmultiplikation Für die Matrixmultiplikation wurde sowohl eine nicht optimierte, direkt in C geschriebene Version verwendet als auch eine optimierte, bereitgestellt durch den `dgemm`-Aufruf der MKL (Intel, 2019a). Der Code dafür stammt ursprünglich aus (NERSC, 2018). Wie bei dem Stream-Benchmark wurde der wesentliche Code wieder explizit instrumentiert und mit zusätzlichen Wiederholungen versehen. Er ist in Quelltext 4.3 zu sehen. Für die Matrizen wurden die in Tabelle 4.3 angegebenen Größen verwendet, mit dem Ziel, die unterschiedlichen Speicherebenen des Testsystems anzusprechen.

Matrixgröße	Speicherbedarf	Speicherbedarf pro Kern	Cache-Ebene
128x128	384 KByte	16 KByte	L1
256x256	1,5 MByte	64 KByte	L2
512x512	6 MByte	256 KByte	Übergang L2/L3
1024x1024	24 MByte	1 MByte	L3
2048x2048	96 MByte	4 MByte	DRAM

Tabelle 4.3: Gesamter Speicherbedarf für die drei in der Matrixmultiplikation verwendeten Matrizen. Der Speicherbedarf pro Kern sowie die Cache-Ebene ergeben sich für das in Kapitel 4.2, S. 58, beschriebene Testsystem.

```
#ifdef SCOREP_USER_ENABLE
  SCOREP_USER_REGION_BEGIN( handle, "dgemm", SCOREP_USER_REGION_TYPE_LOOP)
#endif
    for(int kr = 0; kr < KERNEL_REPEATS; kr++)
    {
#if defined( USE_MKL ) || defined (USE_CBLAS)
      cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
        N, N, N, alpha, matrixA, N, matrixB, N, beta, matrixC, N);
#elif [...]
      [...]
#else
      #pragma omp parallel for private(sum)
        for(i = 0; i < N; i++) {
          for(j = 0; j < N; j++) {
            sum = 0;

            for(k = 0; k < N; k++) {
              sum += matrixA[i*N + k] * matrixB[k*N + j];
            }

            matrixC[i*N + j] = (alpha * sum) + (beta * matrixC[i*N + j]);
          }
        }
#endif
    }
#ifdef SCOREP_USER_ENABLE
  SCOREP_USER_REGION_END( handle )
#endif
```

Quelltext 4.3: Matrixmultiplikation nach NERSC (2018). Die äußere `for`-Schleife sowie die Score-P User-Instrumentierung wurden ergänzt, um eine ausreichende Laufzeit zu erreichen und um auf die OpenMP-Instrumentierung durch Score-P verzichten zu können.

Tabelle 4.4 zeigt die optimalen Core- und Uncorefrequenzen der unterschiedlichen Kernel. Es fällt auf, dass ohne MKL die beiden Größen 1024x1024 und 2048x2048 ihr Optimum bei einer sehr niedrigen Uncorefrequenz haben. Gleichzeitig kommen diese Kernel mit einer einzigen Wiederholung aus, um die notwendigen 50 ms für eine Messung zu erreichen. Die

MKL zeigt den gleichen Effekt nicht, benötigt aber signifikant mehr Wiederholungen, um die notwendige Dauer des Kernels zu erreichen.

Größe	Ohne MKL (Corefrequenz / Uncorefrequenz / Wiederholungen)	Mit MKL (Corefrequenz / Uncorefrequenz / Wiederholungen)
128x128	2,4 GHz / 1,6 GHz / 600	2,3 GHz / 2,5 GHz / 5000
256x256	2,3 GHz / 2,8 GHz / 50	2,3 GHz / 2,2 GHz / 1000
512x512	2,3 GHz / 2,8 GHz / 5	2,3 GHz / 1,9 GHz / 150
1024x1024	2,4 GHz / 1,2 GHz / 1	2,3 GHz / 1,9 GHz / 20
2048x2048	2,5 GHz / 1,2 GHz / 1	2,3 GHz / 1,9 GHz / 20

Tabelle 4.4: Optimale Frequenzen der unterschiedlichen Matrix-Multiplikationen sowie Anzahl der Wiederholungen der eigentlichen Matrix-Multiplikation.

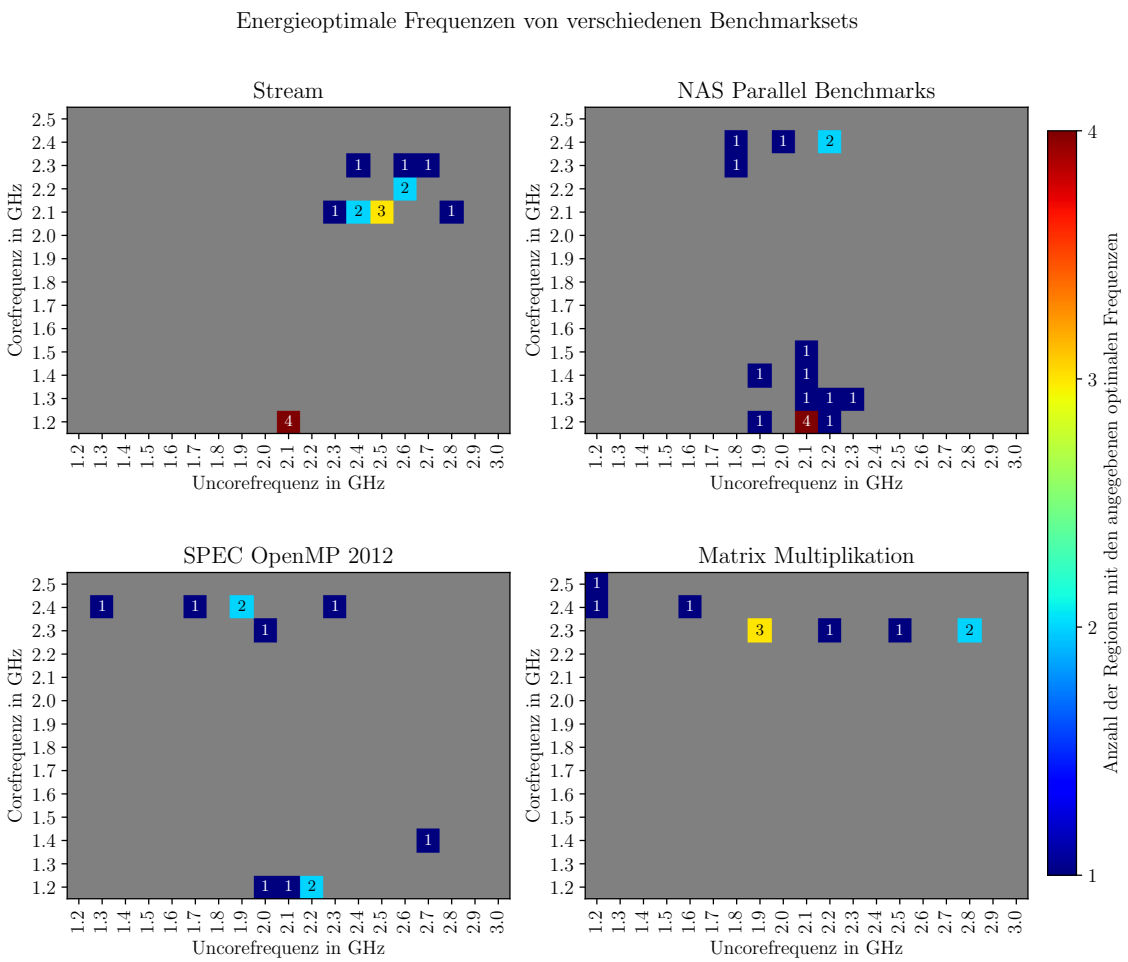
Stream Als Letztes wird der Stream-Benchmark mit den in Kapitel 4.3.1 beschriebenen Anpassungen verwendet. Es werden vier verschiedene Arraygrößen für die Kernel gewählt, um die vier Speicherebenen des Prozessors des Testsystems zu repräsentieren. Tabelle 4.5 gibt einen Überblick über die gewählte Arraygröße, den daraus resultierenden Speicherbedarf und die damit angesprochene Cache-Ebene.

Elemente	Speicherbedarf pro Array	Speicherbedarf pro Array und Kern	Speicherbedarf für zwei / drei Arrays pro Kern	Cache-Ebene
24576	192 KByte	8 KByte	16 KByte / 24 KByte	L1
196608	1,5 MByte	64 KByte	128 KByte / 192 KByte	L2
786432	6 MByte	256 KByte	512 KByte / 768 KByte	L3
12582912	96 MByte	4 MByte	8 MByte / 12 MByte	DRAM

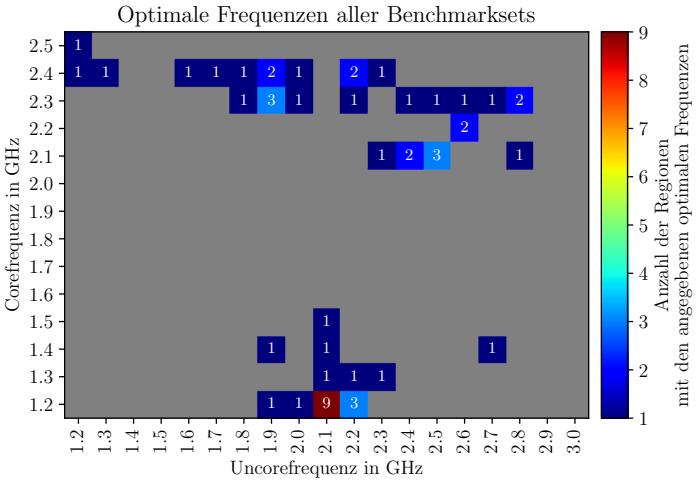
Tabelle 4.5: Gesamter Speicherbedarf für die drei in der Matrixmultiplikation verwendeten Matrizen. Der Speicherbedarf pro Kern sowie die Cache-Ebene ergeben sich für das in Kapitel 4.2, S. 58, beschriebene Testsystem.

Übersicht über alle verwendeten Benchmarks Die aus den unterschiedlichen Benchmarks resultierenden optimalen Frequenzen in Abbildung 4.7a zeigen das Spektrum, das aus den unterschiedlichen Anwendungscharakteristiken resultiert. Die Zahlen geben an, wie viele Regionen des jeweiligen Benchmarksets die jeweilige optimale Frequenz haben. Dabei fällt auf, dass eine niedrige Core- und Uncorefrequenz in keinem der Fälle optimal ist. Dieser Effekt liegt darin begründet, dass eine niedrige Frequenz eine längere Zeit zur Lösung eines Problems nach sich ziehen kann. Da es neben der dynamischen Leistungsaufnahme des Prozessors auch eine statische Leistungsaufnahme gibt, führt eine längere Laufzeit zu einer höheren Energieaufnahme. Es lohnt sich also, das Problem schneller zu lösen.

Werden alle Benchmarksets zusammen genommen, ergibt sich nach Auswahl der relevanten Regionen das in Abbildung 4.7b gezeigte Bild. Es zeigt, dass die verschiedenen Regionen der Benchmarks zusammen eine breite Verteilung der optimalen Frequenzen haben. Das legt nahe, dass sich diese Regionen eignen, um ein möglichst allgemeingültiges Modell zu trainieren. Mit diesem können dann verschiedene Programme mit unterschiedlichen optimalen Frequenzen optimiert werden.



(a) Optimale Frequenzen nach Benchmark-Sammlung. Jede Sammlung hat eine andere Verteilung der optimalen Frequenzen und deckt nur einen begrenzten Teil des Frequenzbereiches ab.



(b) Optimale Frequenzen der kombinierten Benchmark-Sammlung. Die Kombination aller Benchmark-Sammlungen ist besser verteilt und erhöht damit die Abdeckung des möglichen Frequenzbereiches.

Abbildung 4.7: Verschiedene optimale Frequenzen der unterschiedlichen Benchmarks. Die Benchmarks sind, wie in Kapitel 4.4.1 beschrieben, in Regionen aufgeteilt, da unterschiedliche Regionen unterschiedliche optimale Frequenzen haben. Die Zahlen geben an, wie oft eine optimale Core- und Uncorefrequenz vorkommt.

4.4 Merkmalsauswahl und Modellgenerierung

4.4.1 Datenaufbereitung

Nachdem die Daten mit den in Kapitel 4.1.3 beschriebenen Modulen aufgezeichnet worden sind, müssen sie aufbereitet werden. Als Erstes werden dazu die relevanten Regionen nach einem vereinfachten Verfahren ermittelt. Danach werden der Energieverbrauch sowie die optimale Frequenz per Region festgestellt sowie die aufgezeichneten Performance-Events aufbereitet. Anschließend werden die relevanten Events mittels Merkmalsauswahl ausgewählt. Diese Schritte werden im Folgenden im Detail beschrieben.

Relevante Regionen

Da die Merkmalsauswahl und das Training des neuronalen Netzwerks auf bekannten Benchmarks erfolgen, kann eine vereinfachte Definition einer Region verwendet werden: Eine Region ist alles, was zwischen zwei Programm-Events liegt, unabhängig davon, ob es sich um das Betreten oder das Verlassen einer Funktion handelt. Zur eindeutigen Identifikation einer Region werden die Art des Programm-Events sowie eine ID für die Funktion, die das Event ausgelöst hat, gespeichert. Darüber hinaus wird der Name des Programms mit dieser Region assoziiert. Bei dieser Definition einer Region können bestimmte Muster und Verschachtelungen von Regionen nicht korrekt auseinandergehalten werden. Da aber mit einer bekannten Menge von Benchmarks gearbeitet wird, wird dieses Problem durch vorherige Analyse ausgeschlossen. Gleichzeitig vereinfacht diese Definition die Verarbeitung der Daten und erhöht die Menge der signifikanten Regionen.

Anhand einer ersten Referenzmessung wird ermittelt, welche Regionen im Mittel länger als 50 ms laufen. Um einzelne Ausreißer zu erkennen und zu verwerfen, wird für jedes Vorkommen einer Region ermittelt, ob die Laufzeit mehr als das Dreifache der Standardabweichung der mittleren Laufzeit der Region entfernt ist. Als Nächstes wird die Änderung der Arbeitslast einiger Benchmarks berücksichtigt. Durch die Änderung der Arbeitslast kann es vorkommen, dass sich die Laufzeit einer Region über die Zeit verändert. Um dem Rechnung zu tragen, werden die Vorkommen einer Regionen zusätzlich in Cluster eingeteilt, bis die Standardabweichung einer Region in einem Cluster weniger als 6 % von deren Mittelwert ist. Dafür wird der in SciPy implementierte k-Means-Algorithmus verwendet (Arthur und Vassilvitskii, 2007)SciPy2021. Nun kann angenommen werden, dass jede Region innerhalb eines Clusters bei jedem Auftreten ungefähr die gleiche Arbeit leistet.

Jede Wiederholung einer einzelnen Region in einem Cluster kann nun genutzt werden, um den Energieverbrauch unterschiedlicher Frequenzkonfigurationen oder Performance-Events zu analysieren. Durch Wiederholungen des Benchmarks kann sichergestellt werden, dass genug Datenpunkte pro Cluster-Region vorhanden sind.

Energieverbrauch

Wie schon in Kapitel 4.1.3 geschrieben, wird der Energieverbrauch einer Region bei einer zufälligen Konfiguration von Core- und Uncorefrequenz ermittelt. Dadurch kann es dazu kommen, dass für bestimmte Konfigurationen keine Messwerte vorliegen, während andere mehrere Messwerte haben. Um das auszugleichen, wird für jede Region ein regionsspezifisches Energiemodell in Abhängigkeit von der Core- und Uncorefrequenz erstellt. Dieses Modell basiert darauf, dass Frequenz, Spannung und damit auch der Energieverbrauch kontinuierliche Größen sind, die nur durch die ACPI-Spezifikation in feste Schritte geteilt sind. Basierend auf Gleichung 2.2, S. 16, als Modell für den Energieverbrauch und der Annahme, dass die Spannung linear mit der Frequenz gesenkt werden kann, kann abgeschätzt werden, dass der Energieverbrauch einer Region quadratisch von der Frequenz abhängt. Der Energieverbrauch E einer Region lässt sich demnach allgemein mit der folgenden Funktion in

Abhängigkeit von der Frequenz f beschreiben:

$$E(f) = af^2 + bf + c \tag{4.1}$$

Für zwei Frequenzen, wie die Core- und Uncorefrequenz f_c und f_u, wird demzufolge der Energieverbrauch einer Region allgemein wie folgt beschrieben:

$$\begin{aligned} E(f_c, f_u) = af_c^2 f_u^2 & \\ + bf_c^2 f_u + cf_c^2 & \\ + df_c f_u^2 + ef_u^2 & \\ + gf_c f_u + hf_c + if_u + j & \end{aligned} \tag{4.2}$$

Mittels „Trust Region Reflective“ (Branch et al., 1999) Fitting aus SciPy (SciPy, 2020) werden die Parameter bestimmt.

Abbildung 4.8 stellt exemplarisch zwei Regionen sowohl mit gemessenen Daten als auch mithilfe des regionsspezifischen Energiemodells dar. Bei den gemessenen Daten zeigen graue Felder einen fehlenden Messwert für diese Konfiguration an. Wenn mehr als ein Messwert vorlag, wurden diese Messwerte zur Erstellung der Abbildung gemittelt. Der mittlere Fehler, der sich zwischen gemittelten Messwerten und dem Modell ergibt, liegt in beiden Beispielen bei 1 % und der maximale Fehler bei 5 % respektive bei 4 %.

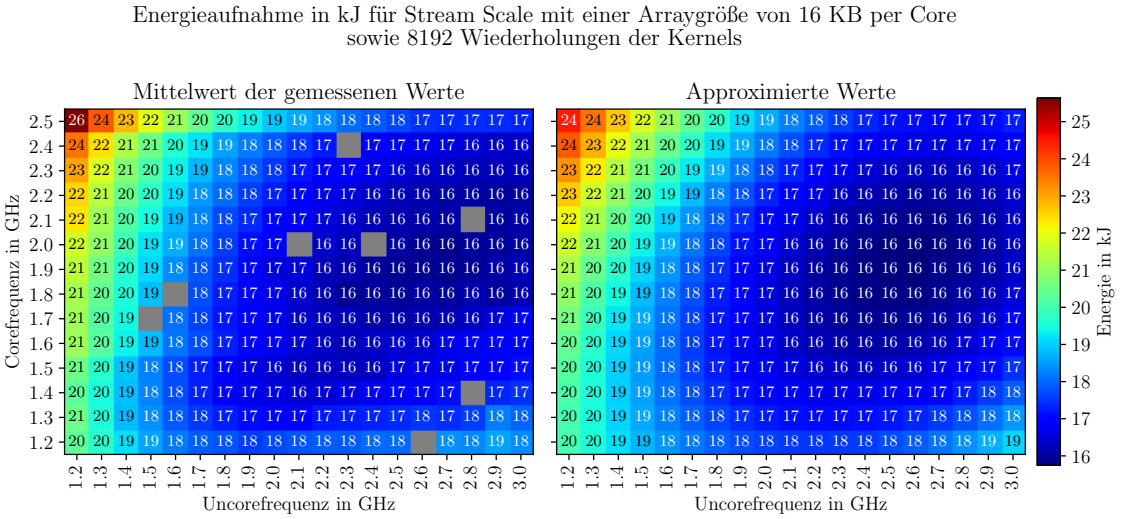
Durch dieses Vorgehen kann sowohl der Energieverbrauch einer nicht gemessenen Einstellung abgeschätzt werden als auch der Messfehler minimiert werden. Abbildung 4.8 stellt exemplarisch zwei Regionen des Stream-Scale-Kernels sowohl mit gemessenen Daten als auch mithilfe des regionsspezifischen Energiemodells dar. Die Daten der in Abbildung 4.8a gezeigten Version passen in den L1-Cache, während die Daten der Abbildung 4.8b nicht vollständig gecacht werden können, sodass regelmäßig Daten aus dem DRAM abgerufen werden müssen. Bei den gemessenen Daten stellen graue Felder einen fehlenden Messwert für diese Konfiguration dar. Wenn mehr als ein Messwert vorlag, wurden diese Messwerte zur Erstellung der Abbildung gemittelt. Der mittlere Fehler, der sich zwischen gemittelten Messwerten und dem Modell ergibt, liegt in beiden Beispielen bei 1 % und der maximale Fehler bei 5 % respektive bei 4 %. Um die optimale Frequenz einer Region zu bestimmen, wird der L-BFGS-B-Algorithmus (Morales und Nocedal, 2011; Zhu et al., 1997), ebenfalls aus SciPy (Zhu et al., 2020), verwendet.

Durch dieses Vorgehen wird sowohl der Energieverbrauch einer nicht gemessenen Einstellung abgeschätzt als auch der Messfehler minimiert. Für jede Region kann mit diesem Modell der Energieverbrauch für jede Frequenz zwischen der minimalen und der maximalen Frequenz ermittelt werden.

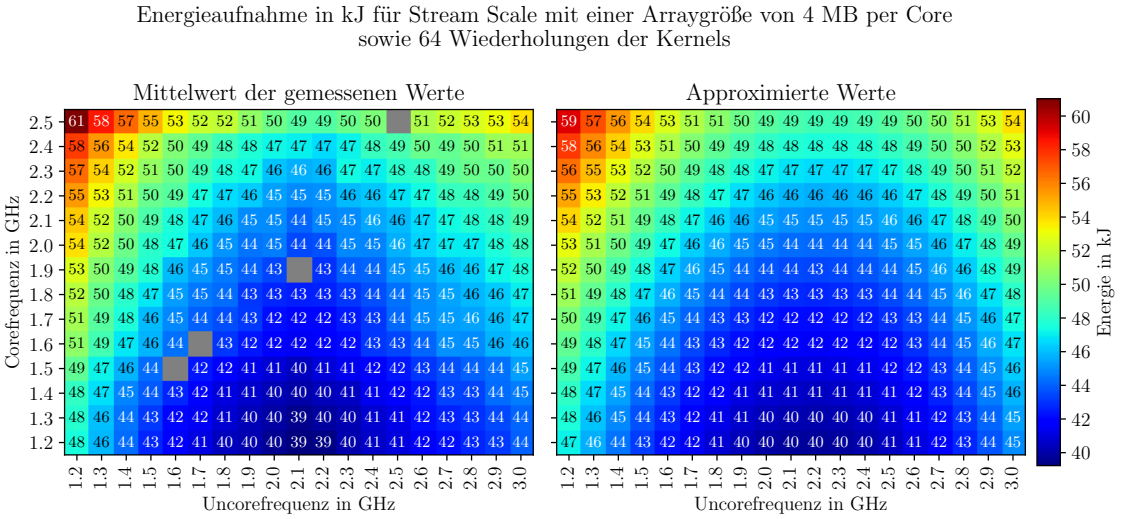
Um die Regionen miteinander vergleichen zu können, müssen sie normiert werden. Eine Normierung auf die Laufzeit scheidet aus, da dadurch der Einfluss unterschiedlicher Frequenzen auf die Laufzeit verloren geht. Chadha und Gerndt (2019) schlagen eine Normierung auf den Energieverbrauch bei einer spezifischen Frequenz von 2,0 GHz und 1,5 GHz Core- respektive Uncorefrequenz vor, welche übernommen werden kann.

Basierend auf den Energiemodellen in Abbildung 4.8 zeigt Abbildung 4.9 das normierte Energiemodell. Da Gleichung 4.2 eine stetige Funktion ist, können nun für die Merkmalsauswahl und das Training Energiewerte für beliebige Frequenzen $f_c \in [1{,}2\,\text{GHz}, 2{,}5\,\text{GHz}]$ und $f_u \in [1{,}2\,\text{GHz}, 3{,}0\,\text{GHz}]$ erzeugt werden.

Turbofrequenzen können bei diesem Vorgehen nicht berücksichtigt werden. Wie in Kapitel 4.2 beschrieben, werden verschiedene Turbofrequenzen durch ACPI in der Frequenz 2,501 GHz zusammengefasst. Damit lassen sie sich nicht gezielt setzen, weswegen sie hier nicht betrachtet werden.



(a) Der durchschnittliche Fehler des approximierten Modells liegt bei 1 %, der maximale Fehler bei 5 %.



(b) Der durchschnittliche Fehler des approximierten Modells liegt bei 1 %, der maximale Fehler bei 4 %.

Abbildung 4.8: Vergleich der gemessenen Daten und des regionsspezifischen Energiemodells. Graue Felder implizieren, dass an dieser Stelle kein Messwert vorliegt. Wenn mehrere Messwerte für eine Frequenzeinstellung vorliegen, wurden diese gemittelt. Wenn kein Messwert für eine Frequenzeinstellung vorliegt, wurde der Fehler nicht ermittelt und bei der Berechnung des durchschnittlichen und maximalen Fehlers nicht berücksichtigt.

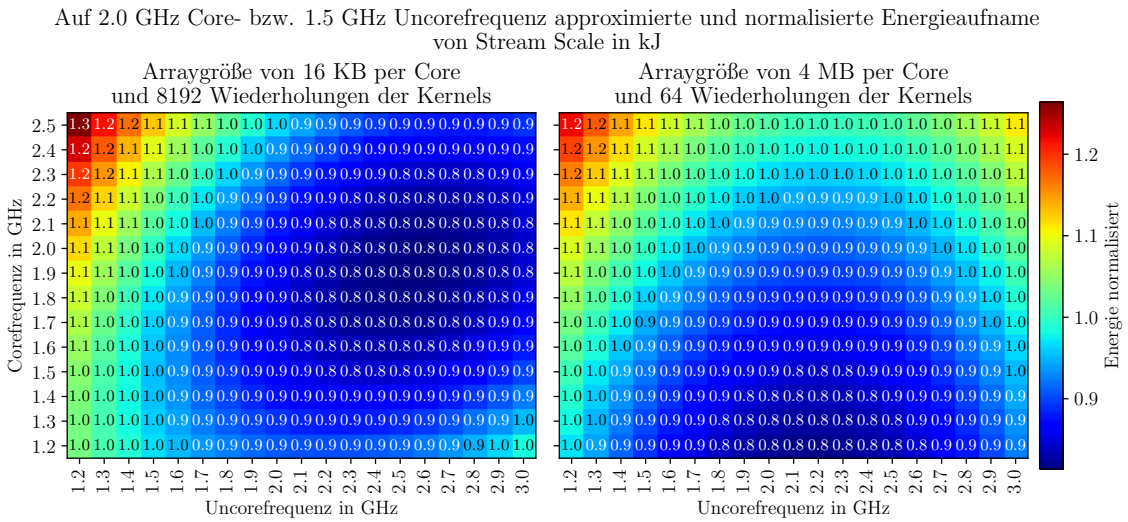


Abbildung 4.9: Vergleich zweier auf 2,0 GHz und 1,5 GHz Core- respektive Uncorefrequenz normierter Regionen

Performance-Counter

Die Messungen der Performance-Events werden bei einer Corefrequenz von 2,5 GHz und einer Uncorefrequenz von 3,0 GHz durchgeführt. Basierend auf Annahme 1, S. 10, und Annahme 2, S. 10, können die Events per Speicherkanal und per Prozessor akkumuliert werden (Intel, 2015, S. 48). Anschließend werden die akkumulierten Werte auf die Laufzeit normiert, sodass sich für jedes Event die entsprechenden Eventraten ergeben.

Wie schon in Kapitel 4.1.3 beschrieben werden die Events, die gemessen werden sollen, zufällig am Anfang einer Region gewählt. Dadurch kann es vorkommen, dass nach einem Programmdurchlauf nicht für alle Events Ergebnisse vorliegen. Die Wahrscheinlichkeit dafür hängt vor allem von der Anzahl der Wiederholungen einer Region ab. Um eine ausreichende Datengrundlage sicherzustellen, wird ein Programmlauf so lange wiederholt, bis insgesamt ungefähr 20 Eventraten pro Event und Region vorliegen. Abschließend werden von den vorliegenden Daten pro Region exakt 20 Werte zufällig gezogen, um verbleibende Unterschiede auszugleichen. Liegen weniger als 20 Werte vor, wird auf Ziehen mit Zurücklegen ausgewichen.

Damit ergeben sich für den Datensatz 1 Stream, Kapitel 4.3.1, bei 104 Regionen 2 080 Datenreihen. Für den Datensatz 2 verschiedene Benchmarks, Kapitel 4.3.2, ergeben sich bei 54 Regionen 1 080 Datenreihen.

4.4.2 Merkmalsauswahl Algorithmus

Für die Merkmalsauswahl werden die in Kapitel 3.2, S. 38 ff., und Kapitel 2.2.2, S. 29, beschriebenen Verfahren

1. $J_{HJMI,CD}$, Historical Joint Mutual Information mit Copula-Dichte,
2. $J_{HJMI,CD,\text{NormMax}}$, Historical Joint Mutual Information mit Copula-Dichte `NormMax`,
3. $J_{MRMR,CD}$, Minimum-Redundancy Maximum-Relevance mit Copula-Dichte,
4. $J_{MRMR,CD,\text{NormGauss}}$, Minimum-Redundancy Maximum-Relevance mit Copula-Dichte `NormGauss`,
5. $J_{MRMR,\gamma}$, Minimum-Redundancy Maximum-Relevance mit Variable-Selection,

6. und Merkmalsauswahl mit Random-Forest-Trees

verwendet. Aufgrund des schlechten Abschneidens

1. der J_{HJMI}, der klassischen Historical Joint Mutual Information,
2. und der $J_{HJMI,CD,\text{NormGauss}}$, der Historical Joint Mutual Information mit Copula-Dichte `NormGauss`,

in Kapitel 3.5, S. 45 ff., wird auf diese hier verzichtet.

Als Merkmale werden die Eventraten und die zufällig mithilfe einer Gleichverteilung gewählte Core- und Uncorefrequenz genutzt. Als Zielgröße wird der normierte Energieverbrauch bei den gewählten Frequenzen verwendet.

Da einige Events in manchen Situationen den Wert 0 zurückgeben, kann es zu den in Kapitel 3.1 beschriebenen Häufungen kommen. Um das zu vermeiden, wird, wie bereits erwähnt, ein leichtes Rauschen über die Daten gelegt. Um die Daten möglichst wenig zu beeinflussen, wird der kleinste Wert c_{min}, der nicht 0 ist, ermittelt. Daraus wird ein gleichverteiltes Rauschen zwischen $c_{min} \cdot 10^{-11}$ und $c_{min} \cdot 10^{-12}$ erzeugt. Dieses Rauschen wird über alle Merkmale inklusive der Frequenzen gelegt.

Das Abbruchkriterium der Merkmalsauswahl wird wie in Kapitel 3.4.2, S. 44, vorgeschlagen generiert. Darüber hinaus werden die wählbaren Merkmale bzw. Events auf die Möglichkeiten der Counter eingeschränkt. Wenn die Counter der in Kapitel 4.2.1 beschriebenen PMON-Blöcke keine weiteren Events mehr aufnehmen können, wird die Berechnung des Zusammenhangs für alle weiteren Events dieses Blocks übersprungen.

Implementierung, Optimierung und Parallelisierung der Merkmalsauswahl

Zur numerischen Bestimmung von $I_{CD}(\mathbf{x}, \mathbf{y})$ mit den in Kapitel 2.2.2, S. 30, und Kapitel 3.2.2, S 40 beschrieben Ansätzen, ergibt sich der in Quelltext 4.4 abgebildete Quelltext. Die Komplexität hängt von der Anzahl der Elemente n, $n = |\mathbf{x}| = |\mathbf{y}|$ sowie der Anzahl der Stützstellen s, $n_u = n_v = s$ ab, vergl. Gleichung 3.24, S. 41, und Gleichung 3.25, S. 42.

Für die Komplexität ergibt sich:

$$O(s^2 \cdot n^2). \tag{4.3}$$

Da die Randdichteverteilungen $F_{\mathbf{x}}$ und $F_{\mathbf{y}}$ unabhängig voneinander sind, lässt sich ihre Berechnung vorher durchführen. Daraus ergibt sich ein Aufwand von:

$$O(n^2) + O(s^2 \cdot n). \tag{4.4}$$

Die Berechnungen der e Funktionen, z. B. `ex` und `ey`, ist ebenfalls unabhängig möglich. Gleichzeitig ist ihre Berechnung aufwendig. Deshalb ist es sinnvoll, diese ebenfalls auszulagern. Es ergibt sich der in Quelltext 4.5 dargestellte Algorithmus mit einer Komplexität von:

$$O(n^2) + O(s \cdot n) + O(s^2 \cdot n). \tag{4.5}$$

Bei einer konstanten Anzahl an Stützstellen s bleibt der Aufwand in beiden Fällen bei $O(n^2)$, steigt also weiter quadratisch mit der Anzahl der Elemente n. Allerdings ist der Anstieg deutlich flacher.

Die Komplexität der Berechnung der dreidimensionalen Copula-Dichte ergibt sich ähnlich: Es muss der Zusammenhang von drei Vektoren verarbeitet werden, es ergibt sich $O(s^3)$. Gleichzeitig bleibt der Aufwand für die Bestimmung der Randdichteverteilungen bei $O(n^2)$. Es ergibt sich:

$$O(s^3 \cdot n^2). \tag{4.6}$$

```
#include <math.h>
double I_cd(double* X, double* Y, int n, int s, double offset){
    double du = 1.0 / s;
    double dv = 1.0 / s;
    double res = 0;
    for (int i = 0; i < s; i++){
        for (int j = 0; j < s; j++){
            double c_uv = 0;
            double u = du * i;
            double v = dv * i;
            for (int k = 0; k < n; k++){
                double F_x = 0;
                double F_y = 0;
                for (int l = 0; l < n; l++){
                    F_x += (X[l] <= X[k]);
                    F_y += (Y[l] <= Y[k]);
                }
                F_x /= n;
                F_y /= n;
                double ex = exp((F_x - u) * offset);
                double ey = exp((F_y - v) * offset);
                c_uv += offset * offset * ex * ey /
                    ((ex + 1) * (ex + 1) * (ey + 1) * (ey + 1));}
            double cd = c_uv / n;
            res += cd * log(cd) * du * dv;
        }
    }
    return res;
}
```

Quelltext 4.4: Algorithmus zur Berechnung der zweidimensionalen Copula-Dichte.

```
#include <math.h>
double I_cd(double* X, double* Y, int n, int s, double offset)
{
    double du = 1.0 / s;
    double dv = 1.0 / s;
    double res = 0;

    double* F_x = (double*)malloc(n*sizeof(double));
    double* F_y = (double*)malloc(n*sizeof(double));
    double* ex = (double*)malloc(n*s*sizeof(double));
    double* ey = (double*)malloc(n*s*sizeof(double));

    for (int k = 0; k < n; k++){
        for (int l = 0; l < n; l++){
            F_x[k] += (X[l] <= X[k]);
            F_y[k] += (Y[l] <= Y[k]);
        }
        F_x[k] /= n;
        F_y[k] /= n;
    }
    for (int i = 0; i < s; i++){
        for (int k = 0; k < n; k++){
            double u = du * i;
            double v = dv * i;
            ex[n * i + k] = exp((F_x[k] - u) * offset);
            ey[n * i + k] = exp((F_y[k] - v) * offset);
        }
    }
    for (int i = 0; i < s; i++){
        for (int j = 0; j < s; j++){
            double c_uv = 0;
            for (int k = 0; k < n; k++){
                double ex_curr = ex[n * i + k];
                double ey_curr = ey[n * j + k];
                double ex_1 = ex_curr + 1;
                double ey_1 = ey_curr + 1;
                c_uv += offset * offset * ex_curr * ey_curr /
                    (ex_1 * ex_1 * ey_1 * ey_1);
            }
            double cd = c_uv / n;
            res += cd * log(cd) * du * dv;
        }
    }
    return res;
}
```

Quelltext 4.5: Optimierter Algorithmus zur Berechnung der zweidimensionalen Copula-Dichte.

Durch die gleichen Veränderungen wie oben lässt sich

$$O(n^2) + O(s \cdot n) + O(s^3 \cdot n) \tag{4.7}$$

erreichen. Auch hier gilt, dass bei konstanten s der Aufwand bei $O(n^2)$ bleibt, der Anstieg aber deutlich flacher ist.

Zur Berechnung von $I_\gamma(\mathbf{x}, \mathbf{y})$ lassen sich ähnliche Optimierungen vornehmen, sodass sich

$$O(s^2 \cdot n^2) \tag{4.8}$$

bzw.

$$O(s^2 \cdot n) + O(n^2) \tag{4.9}$$

ergibt.

Die unterschiedlichen Schleifen lassen sich mittels OpenMP parallelisieren. Eine Parallelisierung mit MPI empfiehlt sich eine Ebene höher: Üblicherweise werden bei der Merkmalsauswahl mehrere Merkmale betrachtet. Die Berechnung der Transinformation oder der verbundenen Transinformation für unterschiedliche Merkmalspaare ist unabhängig. So die Informationen über die Merkmale bei allen MPI-Prozessen vorliegen, ist erst für die Auswahl eines Merkmals eine Synchronisierung erforderlich.

4.4.3 Performance-Events anderer Arbeiten zum Vergleich

Wie bereits in Kapitel 4.2.1 beschrieben entspricht das Testsystem Bakha einem Compute-Knoten des HPC-Systems Taurus (Technische Universität Dresden, 2020). Taurus wird aktiv für unterschiedliche Forschungen genutzt. Deshalb gibt es einige Arbeiten, die für diverse Ziele ebenfalls Performance-Events analysiert und ausgewählt haben. Diese werden in dieser Arbeit ebenfalls auf ihre Tauglichkeit zur Optimierung der Energieeffizienz mit dem vorgeschlagenen Ansatz untersucht.

Zuerst ist die bereits mehrfach zitierte Arbeit von Chadha und Gerndt (2019) zu nennen, da beide ebenfalls versuchen, den Energieverbrauch eines Programmes mithilfe von Performance-Events zu optimieren. Der wesentliche Unterschied zu dieser Arbeit besteht mit Blick auf das Machine-Learning in der verwendeten Methode für die Merkmalsauswahl. Wie bereits in Kapitel 2.1.4, S. 21, beschrieben werden mehrere multiple lineare Regressionsmodelle und deren jeweiliges Bestimmtheitsmaß verwendet, um die relevanten Events zu wählen. Darüber hinaus werden nur die Core-Events betrachtet, deren Anzahl nicht wie in dieser Arbeit auf vier limitiert ist (vergl. Kapitel 4.2.1, S. 58). Die Messungen sind auf der Haswell-Partition von Taurus ausgeführt worden. Tabelle 4.6 zeigt die sieben Performance-Events, die als relevant ermittelt wurden.

PMON-Block (Box)	Event	Umask
`hsw_ep::`	`BR_INST_RETIRED:`	`NOT_TAKEN`
	`MEM_UOPS_RETIRED:`	`ALL_LOADS`
	`L2_RQSTS:`	`ALL_CODE_RD`
		`ALL_DEMAND_DATA_RD`
	`BR_MISP_RETIRED:`	`CONDITIONAL`
	`RESOURCE_STALLS:`	`ANY`
	`MEM_UOPS_RETIRED:`	`ALL_STORES`

Tabelle 4.6: Als relevant für die Optimierung des Energieverbrauchs ermittelte Performance-Events von Chadha und Gerndt (2019).

Eine andere Quelle sind die bereits zitierten Arbeiten Molka (2016) und Molka et al. (2017), in denen unterschiedliche Flaschenhälse mithilfe von Performance-Events identifiziert

werden. Da ein Flaschenhals darauf hindeutet, dass ein bestimmter Teil des Prozessors nicht schnell genug arbeiten kann, liegt die Vermutung nahe, dass die Frequenz eines anderen Teils reduziert werden kann. Damit sollte sich ebenfalls Energie sparen lassen. Im Gegensatz zu dieser Arbeit haben Molka et al. (2017) die Performance-Events manuell bestimmt und sich ebenfalls auf die Core-Events beschränkt. Wie Tabelle 4.7 zeigt, wurden wieder mehr als vier Events gewählt.

PMON-Block (Box)	Event	Umask
`hsw_ep::`	`CPU_CLK_UNHALTED`	
	`CYCLE_ACTIVITY:`	`CYCLES_NO_EXECUTE`
		`STALLS_L1D_PENDING`
	`RESOURCE_STALLS:`	`SB`
	`L1D_PEND_MISS:`	`FB_FULL`
	`OFFCORE_REQUESTS_BUFFER:`	`SQ_FULL`

Tabelle 4.7: Performance-Events zur Bestimmung verschiedener Flaschenhälse nach Molka et al. (2017).

In dem Projekt „Process-Oriented Performance Engineering Service Infrastructure for Scientific Software at German HPC Centers“ (ProPE) (an Mey et al., 2019) wird versucht, Anwendungen zu erkennen, die die Kapazitäten eines Hochleistungsrechners nicht ausnutzen. Dazu werden unter anderem verschiedene Performance-Events gesammelt. Im Gegensatz zu den Arbeiten von Molka et al. (2017) oder Chadha und Gerndt (2019) werden die Daten nicht für eine Region gesammelt, sondern in regelmäßigen Abständen über einen bestimmten Zeitraum gesamplet. Als Performance-Events werden Events verwendet, die einen Eindruck von Zyklen pro Instruktion (CPI) sowie der Speicherbandbreite (likwid, 2020) und den Fließkommaoperationen pro Sekunde (Flops) (likwid, 2020) geben. Wie in Tabelle 4.8 zu sehen werden hier drei Core- und zwei IMC-Events verwendet, was den angelegten Limitierungen dieser Arbeit entspricht.

PMON-Block (Box)	Event	Umask
`hsw_ep::`	`INSTR_RETIRED_ANY`	
	`UNHALTED_CORE_CYCLES`	
	`AVX:`	`ALL`
`hswep_unc_imc::`	`CAS_COUNT:`	`RD`
		`WR`

Tabelle 4.8: Performance-Events zur Erkennung ineffizienter Anwendung nach an Mey et al. (2019).

4.4.4 Erzeugen und Validieren eines Modells mithilfe von TensorFlow und Keras

Zum Aufbau des neuronalen Netzwerkes werden TensorFlow (Martín Abadi et al., 2015, Version 1.14.0) und Keras (Chollet et al., 2015) als Teil von TensorFlow verwendet. Als Eingabewerte werden die per Merkmalsauswahl ermittelten Performance-Events sowie aus unabhängigen Gleichverteilungen gezogene Core- und Uncorefrequenzen verwendet. Als Ausgabewert wird der normierte Energieverbrauch bei diesen Frequenzen genutzt. Die Eingabedaten, mit Ausnahme der Frequenzen, werden auf den Mittelwert und die Standardabweichung der Trainingsdaten normiert. Die Frequenzen werden um den Mittelwert und die Standardabweichung der möglichen wählbaren Frequenzen normiert.

Zur Validierung, zum Testen und während des späteren produktiven Einsatzes des Netzwerkes werden die Eingabedaten um den gleichen Mittelwert und die gleiche Standardabwei-

chung wie die Trainingsdaten bereinigt. Für die Validierung werden von jeder Region zwei Datenpunkte beiseite genommen. Damit kann während des Trainings der Fehler des Modells abgeschätzt werden und das Modell optimiert werden.

Für das Testen wird ein Kreuzvalidierungsverfahren verwendet, um nach dem Training eine Aussage über die Qualität und Generalisierbarkeit des trainierten Modells und der zum Training verwendeten Merkmale treffen zu können: Wird ein Benchmark ausgelassen und wird das trainierte Modell im Test drastisch schlechter, deutet das darauf hin, dass eines oder mehrere verwendete Merkmale nicht generalisierbar genug waren.

Für das Kreuzvalidierungsverfahren wird je ein Benchmark aus dem Training ausgelassen und das Netzwerk mit den verbleibenden Benchmarks trainiert.[2] Danach werden die 20 Datensätze pro Region mit den $14 \cdot 19 = 266$ Kombinationen aus Core- und Uncorefrequenz kombiniert, indem jeder Frequenz ein zufällig gezogener Datensatz zugeordnet wird. Für jede Frequenz wird mithilfe des trainierten Modells ein normierter Energiewert bestimmt. Die Frequenz mit dem niedrigsten Wert repräsentiert das Optimum, dem mithilfe des regionsspezifischen Energiemodells ein erwarteter normierter Energieverbrauch zugeordnet werden kann. Im Vergleich zum Optimum aus den Testdaten ergibt sich ein Fehler in Form eines abweichenden Energieverbrauchs. Anders ausgedrückt zeigt sich, wie viel Optimierungspotenzial nicht genutzt wurde oder ob im Gegenteil sogar mehr Energie verbraucht wurde.

Zusätzlich wird jedes Trainieren und Testen 10-mal wiederholt, um die Unsicherheit, die durch die Zufälligkeit der Initialisierung oder beim Ziehen der Trainingsdaten entsteht, abschätzen zu können.

Verwendete Netzwerke

Für das Training werden drei unterschiedliche neuronale Netzwerke verwendet.

Als Erstes wird das Netzwerk von Chadha und Gerndt (2019) verwendet. Es wird im Weiteren mit *Modell-Chadha* referenziert. Tabelle 4.9 fasst das Netz zusammen.

Ein größeres Netzwerk, das versucht, unterschiedliche Nichtlinearitäten zu berücksichtigen, kommt als Zweites zum Einsatz. Es wird im Weiteren mit *Modell-E* referenziert. Tabelle 4.10 fasst den Aufbau zusammen.

Das dritte Netzwerk basiert auf dem Energiemodell aus Gleichung 4.2, S. 70:

$$\begin{aligned} E(f_c, f_u) = a f_c^2 f_u^2 & \\ + b f_c^2 f_u + c f_c^2 & \\ + d f_c f_u^2 + e f_u^2 & \\ + g f_c f_u + h f_c + i f_u + j & \end{aligned}$$

Das Modell wird im Weiteren mit *Modell-E-Funktion* referenziert. Tabelle 4.11 gibt einen Überblick über das neuronale Netz.

Wenn nicht anders angegeben, nutzen die normalverteilten Gewichte eine Normalverteilung mit dem Mittelwert $0,0$ und der Standardabweichung $0,05$ zur Initialisierung, während die gleichverteilten Gewichte mit einer Gleichverteilung mit dem Minimum von $-0,05$ und einem Maximum von $0,05$ initialisiert werden.

Als Optimierer wird *Adam* eingesetzt. Außer in Modell-Chadha, wo die Learning-Rate auf e^{-3} gesetzt wird, wird die von Keras vorgegebene Learning-Rate von $0,001$ verwendet. Als Fehlerfunktion für das Lernen wird der mittlere quadratische Fehler zu dem normierten Energieverbrauch des Datenpunktes verwendet. Die Größe eines Batches beträgt 32 Datenpunkte. Modell-E sowie Modell-E-Funktion werden mit 50 Epochs trainiert, während Modell-Chadha, wie in Chadha und Gerndt (2019) angegeben, mit 5 Epochs trainiert wird.

[2] Da es sich bei allen Benchmarks um Programme handelt, sind diese prinzipiell miteinander vergleichbar. So das wider Erwarten nicht der Fall ist, sollte sich das durch einen entsprechenden Fehler bei der Kreuzvalidierung bemerkbar machen.

Schicht	Anzahl d. Knoten	Aktivierungsfunktion	Initalisierung der Gewichte
Eingabe			
Nr. 1	5	ReLu	Normalverteilung, $\sigma = \sqrt{(\frac{2}{n})}, n = 5$
Nr. 2	5	ReLu	Normalverteilung, $\sigma = \sqrt{(\frac{2}{n})}, n = 5$
Nr. 3	1	ReLu	Normalverteilung, $\sigma = \sqrt{(\frac{2}{n})}, n = 1$
Ausgabe			

Tabelle 4.9: Modell-Chadha: Aufbau des neuronales Netzes nach Chadha und Gerndt (2019).

Schicht	Anzahl d. Knoten	Aktivierungsfunktion	Initalisierung der Gewichte
Eingabe			
Nr. 1	128	ReLu	Normalverteilung
Nr. 2	128	ReLu	Normalverteilung
Nr. 3	64	ReLu	Normalverteilung
Nr. 4	64	tanh	Normalverteilung
Nr. 5	64	ReLu	Normalverteilung
Nr. 6	64	Exponential	Normalverteilung
Nr. 7	64	ReLu	Normalverteilung
Nr. 8	64	Exponential	Normalverteilung
Nr. 9	64	ReLu	Normalverteilung
Nr. 10	1	Linear	Gleichverteilung
Ausgabe			

Tabelle 4.10: Modell-E: Aufbau des neuronales Netzes mit verschiedenen Nichtlinearitäten.

Schicht	Anzahl d. Knoten	Aktivierungsfunktion	Initalisierung der Gewichte
Eingabe			
Nr. 1	128	ReLu	Normalverteilung
Nr. 2	128	ReLu	Normalverteilung
Nr. 3	64	ReLu	Normalverteilung
Nr. 4	64	tanh	Normalverteilung
Nr. 5	64	ReLu	Normalverteilung
Nr. 6	64	Exponential	Normalverteilung
Nr. 7	64	ReLu	Normalverteilung
Nr. 8 (a - j)	64	ReLu	Normalverteilung
Nr. 9 (a - j)	64	tanh	Gleichverteilung
Nr. 10 (a - j)	64	ReLu	Gleichverteilung
Nr. 11 (a - j)	1	Linear	Gleichverteilung
Nr. 12	-	Gleichung 4.2: $E(f_c, f_u)$	-
Ausgabe			

Tabelle 4.11: Modell-E-Funktion: Aufbau des neuronalen Netzes unter Verwendung von Gleichung 4.2, S. 70. Schichten, die mit a - j gekennzeichnet sind, werden parallel für jeden Parameter a - j der Gleichung 4.2 erzeugt und als Eingabe für den Parameter verwendet.

4.5 Zusammenfassung

In diesem Kapitel wurden die verschiedenen praktischen Details des Frameworks erläutert. Eine wesentliche Grundlage des Frameworks ist dabei die READEX Runtime Library. Diese musste angepasst werden, um das Sammeln der Daten für das Machine-Learning zu ermöglichen. Anschließend wurden sowohl das System, auf dem die Daten gesammelt werden sollen, als auch die Benchmarks, mit denen die Daten generiert werden sollen, vorgestellt. Dabei wurden zwei Datensätze erstellt: Der erste Datensatz ist ein spezieller, der auf dem Stream-Benchmark basiert, nicht generalisierbar ist, aber eine Analyse der ausgewählten Merkmale und neuronalen Netze erlaubt. Der zweite Datensatz ist generischer, da verschiedene Benchmarks verwendet werden. Diese Benchmarks haben sehr verschiedene optimale Frequenzen und sollten das Training eines generischen neuronalen Netzes für die spätere Anwendung mit realen Programmen erlauben.

Schließlich folgen noch die Details für den Machine-Learning-Teil im Framework. Neben den Details zur Aufbereitung der gesammelten Daten wird auch ein Vorschlag für eine effiziente Implementierung der Merkmalsauswahl sowie den Aufbau der für das Training verwendeten neuronalen Netze geäußert. Im folgenden Kapitel wird das hier vorgestellte Framework evaluiert.

5 Evaluierung des Ansatzes

Nachdem im vorherigen Kapitel das Framework im Detail erläutert worden ist, wird in diesem Kapitel das Framework evaluiert. Dazu werden zuerst die im vorherigen Kapitel vorgestellten Datensätze analysiert.

Der erste betrachtete Datensatz basiert, wie bereits in Kapitel 4.3.1, S. 62 diskutiert, auf dem Stream-Benchmark. Konfiguriert wird die durch Stream bearbeitete Datenmenge. Die dadurch entstehenden unterschiedlichen Regionen lassen sich nach verwendeter Datenmenge ordnen und erlauben einen Vergleich der gewählten Merkmale mit der zugehörigen Bandbreite. Das gestattet eine kurze Diskussion der ausgewählten Events. Anschließend werden mithilfe des Kreuzvalidierungsverfahrens die Ergebnisse der unterschiedlichen Merkmale und neuronalen Netze analysiert.

Der zweite betrachtete Datensatz basiert auf den in Kapitel 4.3.2, S. 65, diskutierten verschiedenen Benchmarks. Da diese sich nicht wie im ersten Fall ordnen lassen, werden die gewählten Merkmale lediglich kurz zusammengefasst. Allerdings ist der Datensatz allgemeiner, da die Vielfalt der verwendeten Benchmarks größer ist. Das sollte das Training eines generischen neuronalen Netzes erlauben, mit dem der Energieverbrauch eines beliebigen Programmes vorhergesagt werden kann. Es folgt eine Kreuzvalidierung.

Abschließend wird das gesamte Framework verwendet, um den Energieverbrauch von Kripke zu optimieren. Kripke ist eine Mini-App, also ein kleines Programm, das zur Analyse verschiedener Implementierungen von Lösern der Boltzmann-Gleichung entwickelt wurde (Kunen et al., 2015). Bei der Optimierung kommen die Merkmale und das neuronale Netz zum Einsatz, die für den zweiten, also den allgemeineren Datensatz das beste Ergebnis geliefert haben. Der optimierte Energieverbrauch wird mit den Ergebnissen eines ebenfalls von mir entwickelten Reinforcement-Ansatzes verglichen.

Zusammenfassung der Datengrundlage

An dieser Stelle ist es sinnvoll, noch einmal die Merkmale zusammenzufassen, aus denen gewählt werden soll und die zum Training verwendet werden sollen. Ziel ist, den normalisierten Energieverbrauch E_n einer Region r bei verschiedenen Frequenzen f_c, f_u vorherzusagen, um dann damit die Frequenzen f_c und f_u zu finden, für die der Energieverbrauch minimal wird. Dazu sollen neben den Frequenzen die Eventraten von Performance-Events R_{p_i} verwendet werden. Zusammengefasst ergibt sich (vergl. Gleichung 3.5):

$$E_n(f_c, f_u, r) = g(R_{p_1}(r), R_{p_2}(r), ..., R_{p_n}(r), f_c, f_u)$$

Dabei soll die Funktion g durch ein neuronales Netz approximiert werden.

Die Eventraten wurden bei einer Referenzfrequenz gemessen und sind von f_c und f_u unabhängig. Das E_n von f_c und f_u ist bekannt. Die Frage für die Merkmalsauswahl ist: Welche Merkmale erklären am besten das Verhalten von E_n über f_c und f_u hinaus. Auf die Hardware bezogen geben die verschiedenen Eventraten einen Eindruck von unterschiedlichen Flaschenhälsen. Je nachdem welcher Teil des Prozessors für einen Flaschenhals verantwortlich ist, hat die Änderung von f_c und f_u mehr oder weniger Einfluss auf den Energieverbrauch.

Für die Merkmalsauswahl könnten f_c und f_u unberücksichtigt bleiben, da diese als Merkmal für das spätere Machine-Learning bereits gesetzt sind. Beide Merkmale bieten aber die Möglichkeit einer einfachen Evaluation der Merkmalsauswahlverfahren: Jedes Verfahren soll-

te den Zusammenhang zwischen den Frequenzen und dem normalisierten Energieverbrauch erkennen.

Wie schon beschrieben wird pro Region die Eventrate 20-mal bei einer festen Frequenz gemessen. Jede dieser Eventraten wird mit dem Energieverbrauch der Region bei zufälligen ausgewählten f_c und f_u assoziiert. f_c und f_u werden selbst zu den Merkmalen hinzugefügt.

Für das Lernen werden aus den Merkmalen mithilfe von Merkmalsauswahlverfahren die relevanten Eventraten ausgewählt und zusammen mit f_c und f_u in ein neuronales Netzwerk gegeben. Zielgröße des Trainings ist der zu f_c, f_u und der Region assoziierte Energieverbrauch $E_n(f_c, f_u, r)$. Bewertet werden die Merkmale und neuronalen Netze am Ende nach der Abweichung des Energieverbrauchs bei der so vorhergesagten minimalen Frequenz verglichen mit dem bekannten Optimum.

Training und Evaluierung

Für das Training und die Evaluierung der neuronalen Netze und gewählten Merkmale werden die Daten, wie in Kapitel 4.4.4, S. 77, beschrieben, in drei Datensätze aufgespalten. Zuerst werden die Daten von jeweils einem Benchmark als Testdatensatz beiseite genommen. Von den 20 Datenpunkten jeder Region werden zwei zur Validierung des Trainings beiseite genommen. Die restlichen 18 Datenpunkte werden zum Training verwendet. Wenn das Training abgeschlossen ist, wird das entstandene neuronale Netzwerk mit dem Testdatensatz getestet. Dies wird für jeden Benchmark als Testdatensatz wiederholt.

5.1 Der Stream-Benchmark

Zunächst wird, wie in Kapitel 4.3.1, S. 62, beschrieben, der Stream-Benchmark analysiert. Dabei dient dieser Benchmark als ein spezieller, nicht generalisierbarer Fall. Gleichzeitig lässt er aber eine Analyse der ausgewählten Merkmale und neuronalen Netze zu, wie im Folgenden zu sehen.

5.1.1 Analyse der gewählten Merkmale

Der Stream-Benchmark erlaubt eine Einordnung der gewählten Merkmale im Bezug zur verwendeten Datenmenge, zu der damit erreichbaren Bandbreite und damit zur Ebene der Speicherhierarchie, die verwendet wird. Die Speicherebene beeinflusst wiederum die Frequenz, bei der der Energieverbrauch optimal wird, wie in Kapitel 4.3.1, S. 62, beschrieben.

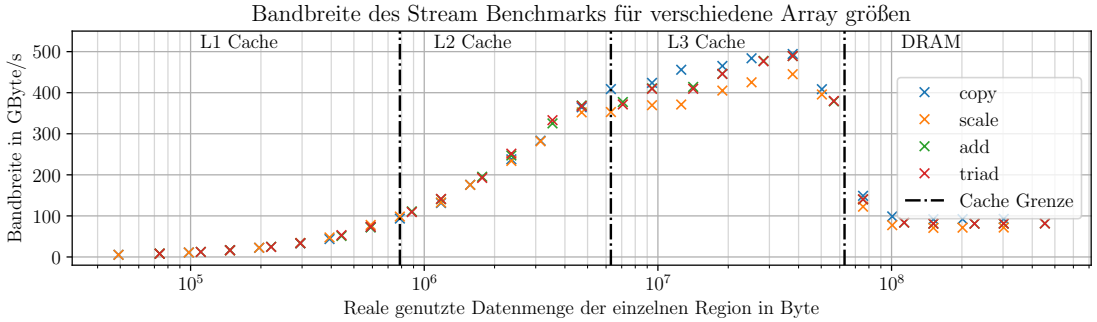


Abbildung 5.1: Gemessene Speicherbandbreite der vier Stream-Kernel `copy`, `scale`, `add` und `triad`, die mittels Score-P User-Instrumentierung und der RRL vermessen wurden. Die theoretische Grenze der Daten, die in einen Cache passen, ist dargestellt (vergl. Kapitel 4.3.1, S. 63).

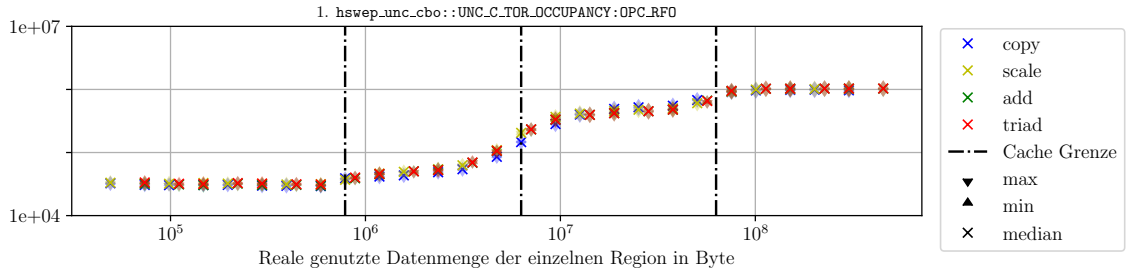


Abbildung 5.2: Beispieleventrate von `hswep_unc_cbo::UNC_C_TOR_OCCUPANCY:OPC_RFO` über die Datenmenge und die unterschiedlichen Kernel.

Zur Einordnung zeigt Abbildung 5.1 noch einmal den Verlauf der Speicherbandbreite der verschiedenen Kernel. Abbildung 5.2 zeigt zum Vergleich den Verlauf einer der gewählten Eventraten. Dabei ist die Bandbreite bzw. die Eventrate für die verschiedenen Kernel `copy`, `scale`, `add` und `triad` durch die Farbe separiert und über die Datenmenge geordnet aufgetragen. In Abbildung 5.2 sind neben dem Median aus 10 Messungen noch das Minimum und Maximum aufgetragen.

Ein gutes Event erlaubt, wie das in Abbildung 5.2 zu sehende Event, eine Separierung der Speicherebenen. Gleichzeitig sollte das Event, wie das gezeigte, möglichst wenig streuen. Dabei ist es nicht notwendig, dass der gleiche Verlauf wie in Abbildung 5.1 zu sehen ist. Wenn sich allerdings Ähnlichkeiten erkennen lassen, ist das ein weiteres Merkmal eines guten Events. Abbildung 5.3 bis Abbildung 5.8 sind ähnlich zu Abbildung 5.2 aufgebaut und geben einen Überblick über die Events, die durch die verschiedenen Merkmalsauswahlverfahren ausgewählt wurden.

Um das Kapitel nicht unnötig in die Länge zu ziehen, wird auf eine ausführliche Beschreibung aller gewählten Events verzichtet. Diese lässt sich in den entsprechenden Handbüchern nachlesen: Events mit dem Präfix `hsw_ep` können in Intel (2019c, S. 3605, Vol. 3B 19-1 ff.) gefunden werden, alle anderen lassen sich in Intel (2015) finden.

Merkmale gewählt mit $J_{HJMI,CD}$

Der Algorithmus wählt zunächst die Uncore- und danach die Corefrequenz als relevante Merkmale zur Bestimmung eines normierten Energiemodells der verschiedenen Regionen. Darauf folgen die in Abbildung 5.3 dargestellten Eventraten in der angegebenen Reihenfolge.

Zunächst zeigt sich, dass der Algorithmus 20 Events wählt, bevor das Abbruchkriterium greift. Da die Events aus unterschiedlichen PMON-Blöcken kommen, werden die in Tabelle 4.2, S. 61, aufgeführten Limitierungen pro Block dennoch eingehalten.

Besonders bei der ersten ausgewählten Eventrate

hswep_unc_cbo::UNC_C_TOR_OCCUPANCY:OPC_RFO

zeigt sich die Auswirkung der Überschreitung einer Cachegrenze. Das Event kommt aus den Caching-Agents des Prozessors (Intel, 2015, S. 66: „TOR_OCCUPANCY“). OPC_RFO steht für „OPCode Read For Ownership“ und verweist auf das Cache-Koheränz-Protokoll (Intel, 2015, S. 47). Die Eventrate gibt demnach an, wie oft ein Kern versucht sich einen Teil des Speichers für das exklusive Arbeiten zu sichern.

Technisch gesehen lässt sich allein mit diesem Event für den Stream-Benchmark ein Modell zur Abschätzung des optimalen Arbeitspunktes der Core- und Uncorefrequenz erstellen: Wenn die Eventrate bei ungefähr $1 \cdot 10^6 \frac{\#}{s}$ liegt, nutzt der Stream-Benchmark offensichtlich den DRAM, und eine Core- und Uncorefrequenz von $f_c = 1{,}2\,\text{GHz}$ und $f_u = 2{,}1\,\text{GHz}$ ist optimal, wie in Abbildung 4.6, S. 64, zu sehen. Wenn die Eventrate darunter liegt, sind $f_c = 2{,}2\,\text{GHz}$ und $f_u = 2{,}5\,\text{GHz}$ optimal. Auch ein Modell des normierten Energieverbrauchs

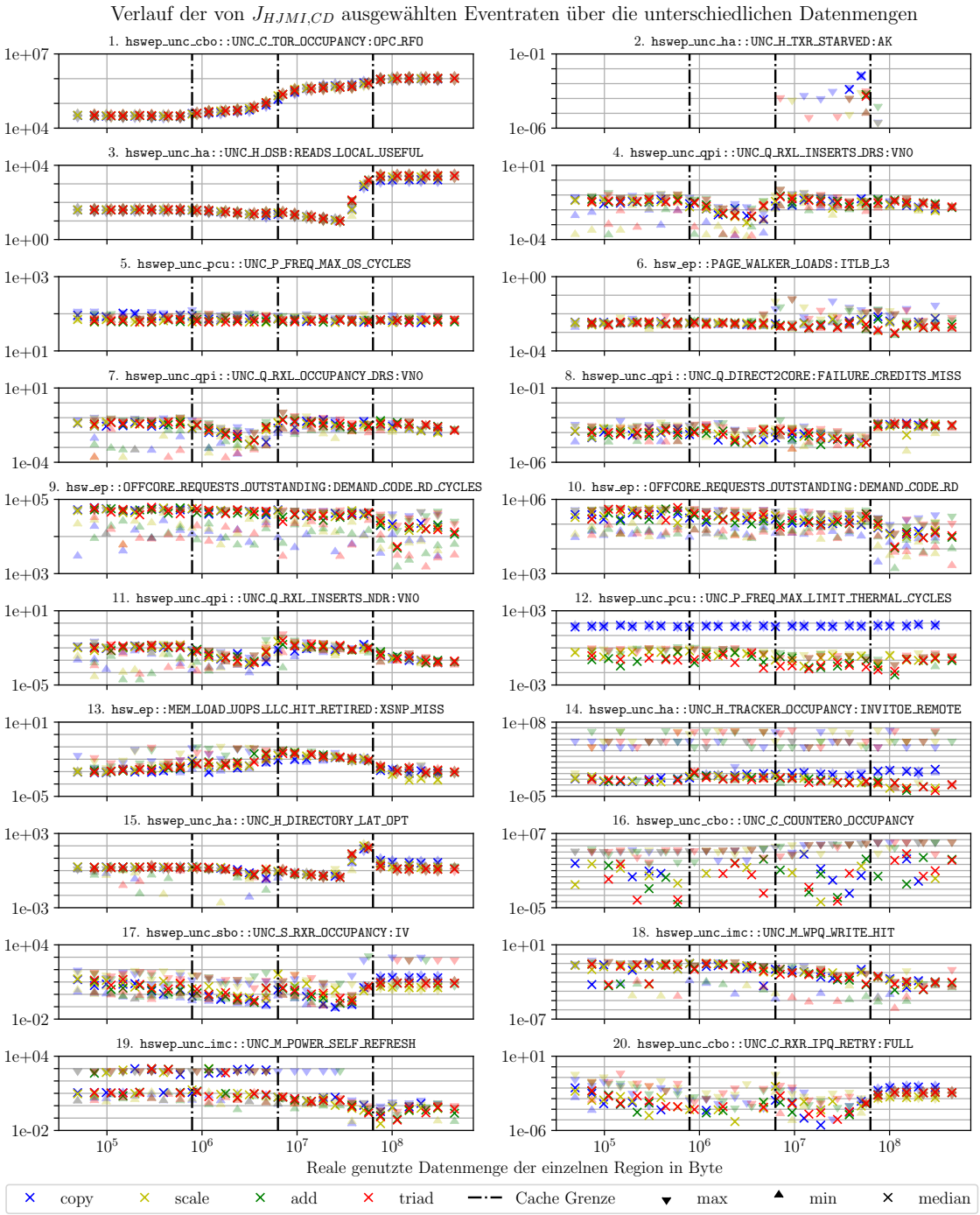


Abbildung 5.3: Verlauf der durch $J_{HJMI,CD}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab. Wenn, wie bei `hswep_unc_ha::UNC_H_TXR_STARVED:AK`, bestimmte Werte nicht angegeben sind, sind sie 0 und können nicht dargestellt werden.

bei verschiedenen Frequenzen ließe sich damit erstellen. Interessant ist, dass die Eventrate immer weiter steigt, während die Speicherbandbreite ab dem Übergang zum DRAM sinkt.

Allerdings wählt der Algorithmus noch eine Menge weiterer Merkmale, bevor er abbricht. Dabei gibt es aber wenig Ähnlichkeiten zwischen den ausgewählten Merkmalen.

Merkmale gewählt mit $J_{HJMI,CD,\textsf{NormMax}}$

Der Algorithmus wählt wieder zunächst die Uncore- und danach die Corefrequenz als relevante Merkmale. Darauf folgt die in Abbildung 5.4 abgebildete Eventrate. Dabei handelt es sich um das gerade beschriebene

hswep_unc_cbo::UNC_C_TOR_OCCUPANCY:OPC_RFO,

das durch $J_{HJMI,CD}$ als Erstes gewählt wurde. Offensichtlich führt die Normierung auch dazu, dass die Abbruchbedingung sehr viel eher greift.

Merkmale gewählt mit $J_{MRMR,CD}$

Der Algorithmus wählt zunächst die Uncorefrequenz, danach die erste in Abbildung 5.5 angegebene Eventrate und erst danach die Corefrequenz als relevante Merkmale. Darauf folgen die restlichen in Abbildung 5.5 gezeigten Eventraten in der angegebenen Reihenfolge.

Die erste gewählte Eventrate

hswep_unc_sbo::UNC_S_RING_AD_USED:UP

ist hier nicht so prägnant wie die durch das vorherige Auswahlverfahren gewählten ersten Eventraten. Zwar sinkt sie beim Übergang von L3-Cache zum DRAM leicht ab, aber der Unterschied ist nicht eindeutig. In Kombination mit der vierten gewählten Eventrate,

hswep_unc_qpi::UNC_Q_DIRECT2CORE:FAILURE_CREDITS_MISS,

lässt sich das ausgleichen. Allerdings streut dieses Event mehr.

Merkmale gewählt mit $J_{MRMR,CD,\textsf{NormGauss}}$

Der Algorithmus wählt zunächst die Uncorefrequenz, danach die erste in Abbildung 5.6 angegebene Eventrate und erst danach die Corefrequenz als relevante Merkmale. Darauf folgen die restlichen in Abbildung 5.6 gezeigten Eventraten in der angegebenen Reihenfolge.

Es werden weniger Eventraten gewählt als im vorherigen Fall, der ohne Normierung von I_{CD} arbeitet. Interessant ist das erste gewählte Event

hswep::RESOURCE_STALLS:ALL.

Bevor eine Operation in den Scheduler gegeben werden kann, müssen die entsprechenden Ressourcen allokiert werden. Wenn eine der Komponenten des Prozessors einen Flaschenhals darstellt, kommt es hier zu einem Mangel, womit es zu sogenannten Verzögerungen (engl. stalls) kommt (Intel, 2019b, S. B-46/S. 770). Diese Stalls steigen immer weiter an, erlauben aber wieder eine Trennung der Speicherebenen. Das dritte und vierte gewählte Event bieten über das erste hinaus keine Mehrinformation. Da sie aber auch einen eindeutigen Verlauf haben, können sie die Qualität des später trainierten Modells verbessern.

Merkmale gewählt mit $J_{MRMR,\gamma}$

Auch dieser Algorithmus wählt die Uncorefrequenz, danach die erste in Abbildung 5.7 angegebene Eventrate und erst danach die Corefrequenz als relevante Merkmale. Darauf folgen die übrigen angegebenen Eventraten. Besonders interessant ist in diesem Fall die Eventrate von

hsw_ep::L1D_PEND_MISS:OCCURRENCES.

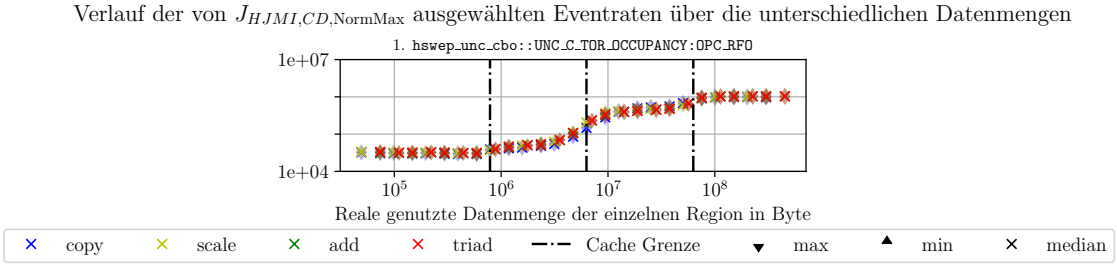


Abbildung 5.4: Verlauf der durch $J_{HJMI,CD,\text{NormMax}}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab.

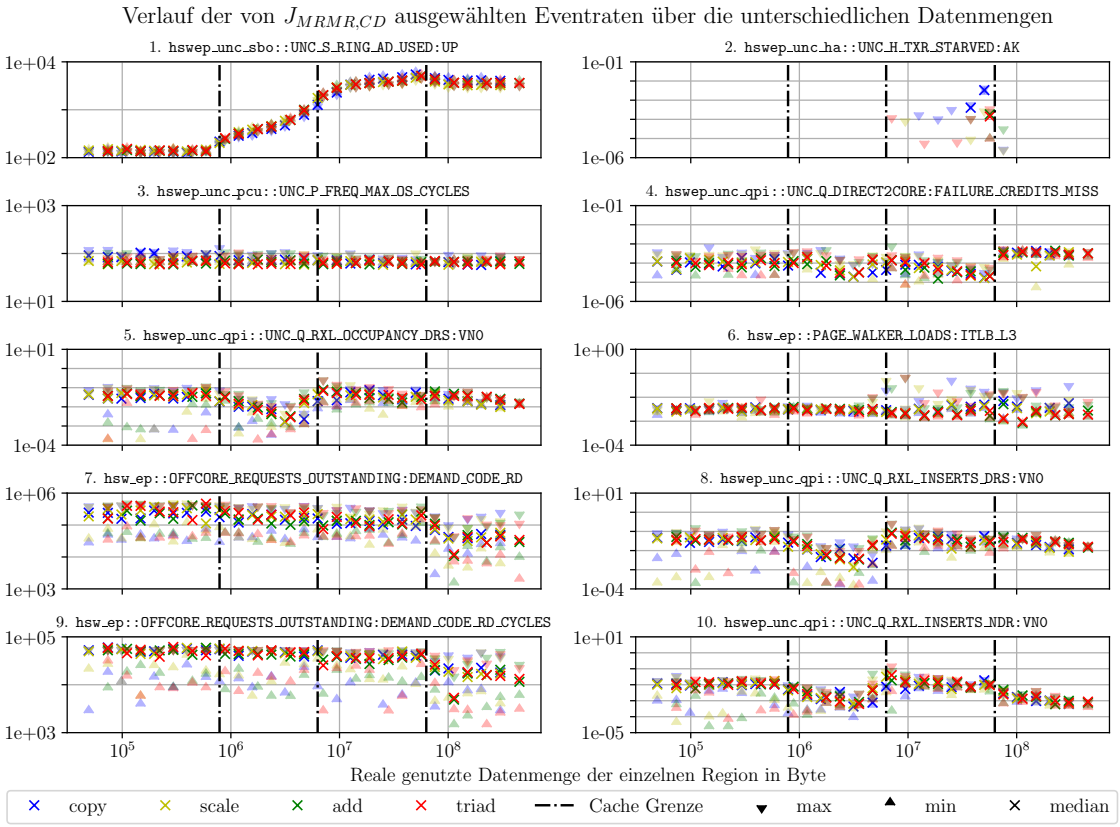


Abbildung 5.5: Verlauf der durch $J_{MRMR,CD}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab. Wenn, wie bei `hswep_unc_ha::UNC_H_TXR_STARVED:AK`, bestimmte Werte nicht angegeben sind, sind sie 0 und können nicht dargestellt werden.

Im Unterschied zu allen vorher gewählten Raten gibt es hier klare Unterschiede zwischen den vier Kerneln des Stream-Benchmarks: `copy` und `scale` sowie `add` und `triad`. Vermutlich hängt das mit dem ansteigenden Rechenaufwand der Kernel zusammen. Damit können neben den Cache-Ebenen, die sich mithilfe der anderen gewählten Eventraten voneinander abgrenzen lassen, auch die unterschiedlichen Stream-Kernel unterschieden werden. Dadurch sollte sich für Stream ein genaues normiertes Energiemodell erstellen lassen.

Merkmale gewählt mit Random Forest

Vor den in Abbildung 5.8 gezeigten Eventraten werden mithilfe von Random Forest wieder die Uncore- und die Corefrequenz gewählt. Mit 24 Eventraten werden mit diesem Verfahren die meisten Events gewählt. Dabei zeigt sich, dass zwischen vielen Eventraten große Ähnlichkeiten bestehen. Als Beispiel sei auf 3.

```
hswep_unc_sbo::UNC_S_RING_AD_USED:UP
```

bzw. 6.

```
hswep_unc_sbo::UNC_S_RING_AD_USED:UP_EVEN
```

und 8.

```
hswep_unc_sbo::UNC_S_RING_AD_USED:UP_ODD
```

verwiesen. `:UP_ODD` und `:UP_EVEN` zählen ungefähr das Gleiche und ergeben zusammen `:UP` (Intel, 2015, S. 76). Wie bereits in Kapitel 3.5.3, S. 51, festgestellt ist es bei einer Merkmalsauswahl mithilfe von Random Forest nicht möglich, Merkmale, die sich wiederholen, herauszufiltern.

Zusammenfassung

Die unterschiedlichen Merkmalsauswahlverfahren lenken die Aufmerksamkeit auf unterschiedliche Events und deren Eventraten. Bei einigen Events ist sofort ersichtlich, dass sich damit ein normiertes Energiemodell für Stream erstellen lässt. Alle Verfahren erkennen den Zusammenhang mit der Core- und Uncorefrequenz.

Einen großen Unterschied gibt es bei der Menge der gewählten Merkmale. So brechen $J_{HJMI,CD,\text{NormMax}}$ und $J_{MRMR,CD,\text{NormGauss}}$ nach einem bzw. vier ausgewählten Eventraten ab und haben damit die wenigsten gewählten Merkmale. $J_{MRMR,\gamma}$ und $J_{MRMR,CD}$ brechen nach 7 bzw. 10 Eventraten ab und liegen damit im Mittelfeld. Am meisten Eventraten werden bei $J_{HJMI,CD}$ mit 20 und bei Random Forest mit 24 gewählt.

Wie gut mit den ausgewählten Eventraten ein neuronales Netz trainiert werden kann, wird im Folgenden analysiert.

5.1.2 Ergebnisse des Trainings

Abbildung 5.9 zeigt die Ergebnisse über alle Iterationen des in Kapitel 4.4.4, S. 77, beschriebenen Kreuzvalidierungsverfahrens. Um Ergebnisse der unterschiedlichen Modelle und Merkmalskombinationen einordnen zu können, wurden alle Modelle einmal nur mit der Core- und Uncorefrequenz trainiert. Mit diesen Modellen lässt sich jeweils eine mittlere, für alle Konfigurationen optimale Frequenz ermitteln, mit der bereits Energie eingespart werden kann. Das ist vergleichbar mit der Anpassung der Standardfrequenz eines Systems, wie bereits in der Einleitung Kapitel 1, S. 7, diskutiert (Auweter et al., 2014). Jede Optimierung sollte besser sein als diese Referenz.

Weiter wird darauf verzichtet, den mittleren quadratischen Fehler zum normierten Energieverbrauch zu berichten. Dieser wurde zwar zum Training verwendet, ist aber für die Einordnung der Ergebnisse nicht hilfreich. Ziel der Arbeit ist es, eine optimale Frequenz zu finden. Als Evaluierungskriterium wird deshalb der Mehrverbrauch an Energie durch die

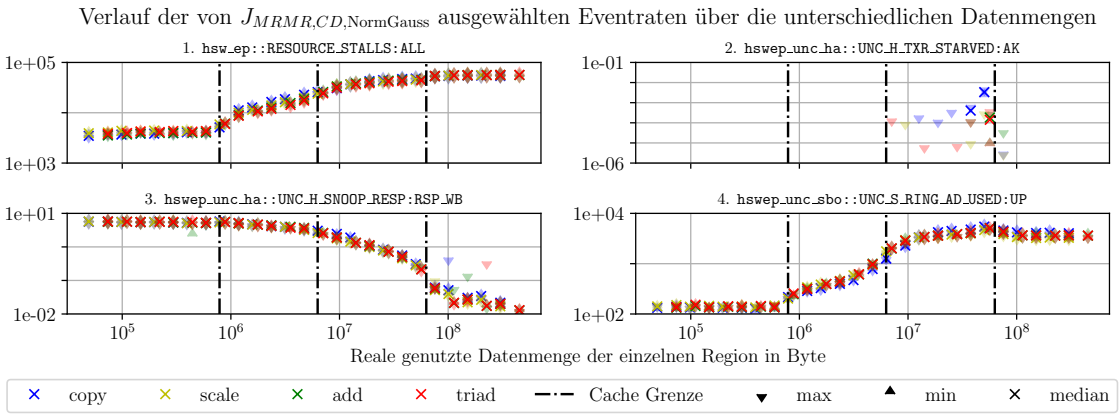


Abbildung 5.6: Verlauf der durch $J_{MRMR,CD,\text{NormGauss}}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab. Wenn, wie bei `hswep_unc_ha::UNC_H_TXR_STARVED:AK`, bestimmte Werte nicht angegeben sind, sind sie 0 und können nicht dargestellt werden.

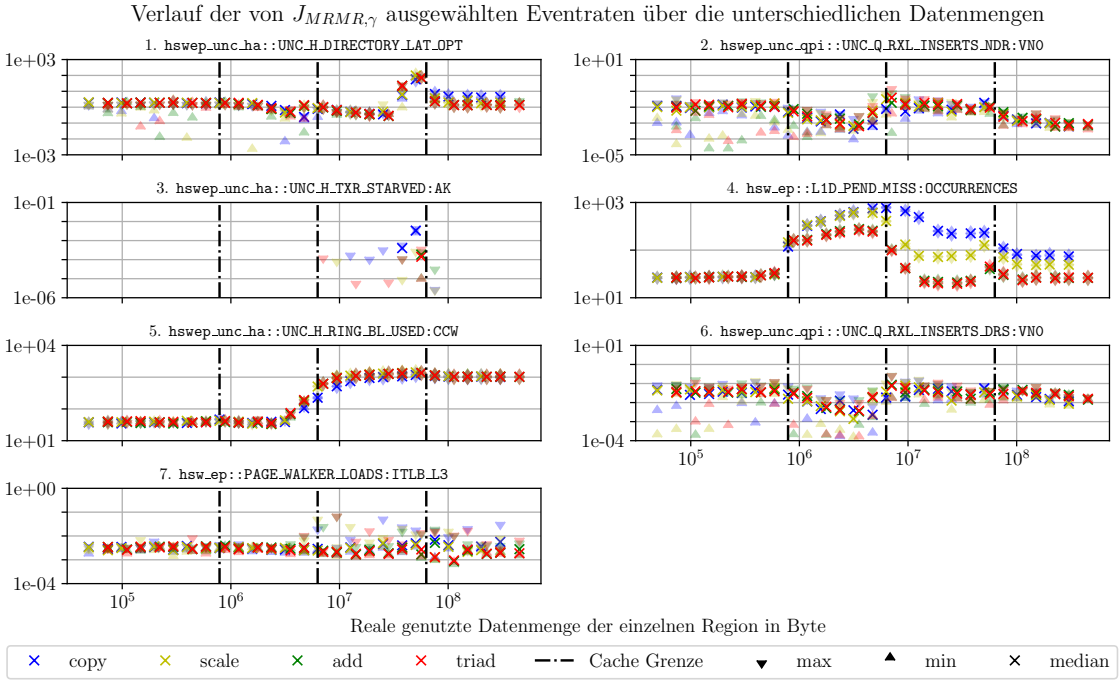


Abbildung 5.7: Verlauf der durch $J_{MRMR,\gamma}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab. Wenn, wie bei `hswep_unc_ha::UNC_H_TXR_STARVED:AK`, bestimmte Werte nicht angegeben sind, sind sie 0 und können nicht dargestellt werden.

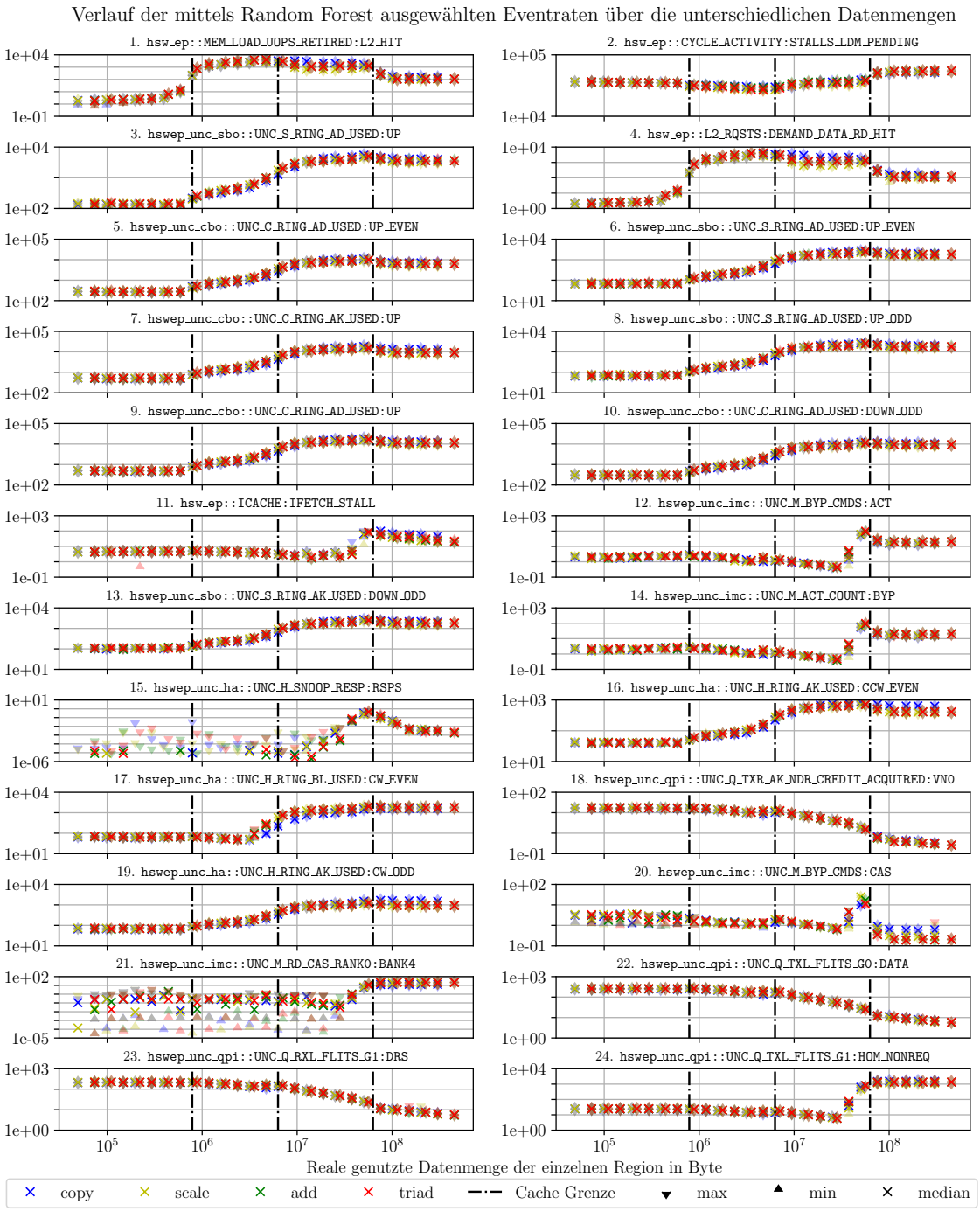


Abbildung 5.8: Verlauf der mithilfe von Random Forest ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. Es ist jeweils der Median, das Maximum und das Minimum der gemessenen Eventrate angegeben. Die Grenzen der Caches des verwendeten Prozessors sind angegeben und grenzen von links nach rechts den L1-, L2- und L3-Cache sowie den DRAM voneinander ab.

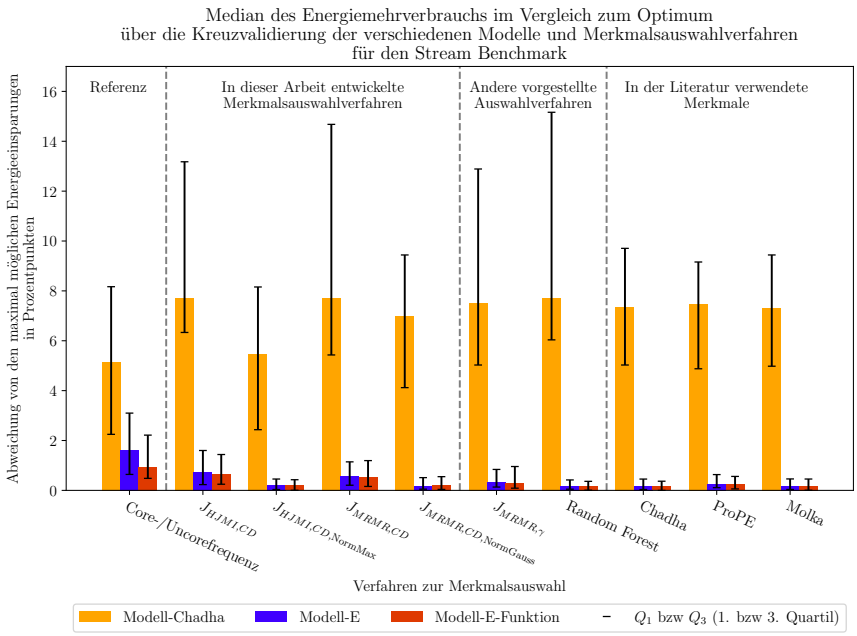


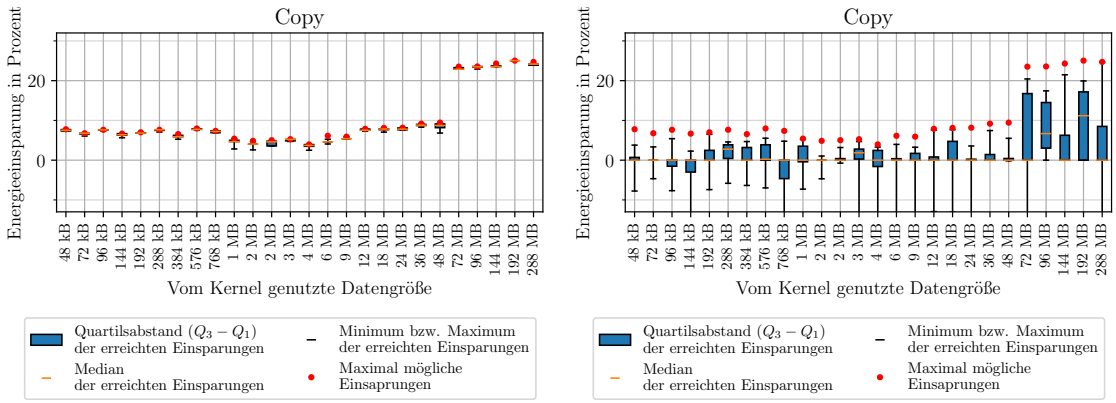
Abbildung 5.9: Median des Energiemehrverbrauchs bezogen auf die maximal möglichen Energieeinsparungen für Stream. Je geringer der Median, desto besser ist das neuronale Netz zusammen mit den Merkmalsauswahlverfahren. Der Median sowie die Quartile wurden über alle Kreuzvalidierungsversuche der jeweiligen Modelle gebildet. Bei dem Merkmalsauswahlverfahren Core-/Uncorefrequenz wurden zum Training des Modells neben der Core- und Uncorefrequenz keine weiteren Merkmale verwendet. Jedes Training mit Merkmalen sollte besser sein als diese Referenz. Die Verfahren Chadha, ProPE und Molka beziehen sich auf die in Arbeiten von Chadha und Gerndt (2019), an Mey et al. (2019) und Molka et al. (2017) ausgewählten Merkmale.

vom Modell gewählte optimale Frequenz im Vergleich zur bekannten optimalen Frequenz bewertet.

Bei den detaillierten Analysen der unterschiedlichen Modelle, in Abbildung 5.11 bis Abbildung 5.16, wird die Energieeinsparung im Vergleich zur höchsten Core- und Uncorefrequenz angegeben. Die maximal möglichen Einsparungen sind ebenfalls angegeben. Bei näherer Betrachtung der maximal möglichen Einsparungen zeigt sich ein Sprung beim Übergang der von Stream verwendeten Datengröße von 36 MB zu 72 MB. An dieser Stelle muss neben dem L3-Cache von 30 MB auch der DRAM verwendet werden, wodurch sich auch die optimale Frequenz verschiebt.

Abbildung 5.10 zeigt zur Veranschaulichung der folgenden Darstellung und der anschließenden detaillierten Analysen ein gutes und ein schlechtes Ergebnis. Bei einem guten Ergebnis sind die Quartile sowie das Minimum und das Maximum möglichst klein. Gleichzeitig liegt der Median möglichst nah am Optimum, dem roten Punkt. Der Fehler im Vergleich zum Optimum ist damit gering. In Abbildung 5.10a ist das der Fall, in Abbildung 5.10b nicht.

Bei der Betrachtung des Ergebnisses von Modell-Chadha zeigt sich, dass das neuronale Netz in den 10 Iterationen jedes Schrittes des Kreuzvalidierungsverfahrens sehr unterschiedliche Modelle generiert. Das führt zu einer hohen Schwankung des in Abbildung 5.9 angegebenen Fehlers. Schaut man sich das Ergebnis der Kreuzvalidierung in Abbildung 5.11 und Abbildung 5.12 im Detail an, zeigt sich, dass mit manchen der generierten Modelle die optimale Frequenz sehr gut ermittelt werden kann. Andere liegen hingegen weit daneben. Das Hinzufügen von Merkmalen zur Core- und Uncorefrequenz führt meistens zu einer Verschlechterung des Ergebnisses. Das legt die Vermutung nahe, dass das Modell eine Art mittleres



(a) Gutes Ergebnis

(b) Schlechtes Ergebnis

Abbildung 5.10: Vergleich eines guten und eines schlechten Trainingsergebnisses. Bei einem guten Ergebnis sind die Quartile sowie das Minimum und das Maximum möglichst klein. Gleichzeitig liegt der Median möglichst nah am Optimum, dem roten Punkt. Der Fehler im Vergleich zum Optimum ist damit gering. In Abbildung 5.10a ist das der Fall, in Abbildung 5.10b nicht.

Energiemodell lernt, das mal näher am optimalen Fall für die Datenmengen liegt, die in den Cache passen, und mal näher an dem optimalen Fall für Daten im DRAM. Das Hinzufügen von Merkmalen, also Events, scheint die Generierung des durchschnittlichen Modells eher zu behindern. Es kann nicht ausgeschlossen werden, dass mit Modell-Chadha ein Modell generiert werden kann, das in allen Fällen eine gute Bestimmung der optimalen Frequenzen erlaubt. Allerdings kann davon auch nicht in jedem Fall ausgegangen werden.

Anders sieht das für Modell-E und Modell-E-Funktion aus. Wie die Ergebnisse in Abbildung 5.13 bis Abbildung 5.16 zeigen, lernen die beiden Modelle allein mit der Core- und Uncorefrequenz ein stabiles Modell, das mit jedem Training zu ähnlichen Ergebnissen führt. Durch das Hinzufügen von weiteren Merkmalen in Form der jeweils ausgewählten Eventraten verbessert sich das Ergebnis. Die Schwankungen, die jetzt noch bleiben, ergeben sich vor allem durch wiederholte Fehler bei einzelnen Konfigurationen des Stream-Benchmarks. Diese lassen sich durch Randfälle bei den ausgewählten Merkmalen erklären, also zum Beispiel eine Eventrate, die beim Übergang von L3-Cache zum DRAM ausschlägt, bevor sich die optimale Frequenz verschiebt. Bezogen auf die Merkmalsauswahlverfahren lässt sich sagen, dass $J_{HJMI,CE,\text{NormMax}}$, $J_{MRMR,CE,\text{NormGauss}}$ und Random Forest am besten abschneiden. Auch die von Chadha und Gerndt (2019), Molka et al. (2017) und an Mey et al. (2019) verwendeten Merkmale lassen sich verwenden, um ein zuverlässiges Modell zu generieren. Wie bereits beschrieben erfüllt aber Chadha und Gerndt (2019) nicht die Anforderung von nur vier Core-Performance-Events, während Molka et al. (2017) und an Mey et al. (2019) Expertenwissen voraussetzen, um die entsprechenden Merkmale zu wählen. Anders formuliert ist es für ein wohldefiniertes Beispiel wie den Stream-Benchmark mit den vorgestellten Merkmalsauswahlverfahren möglich, Merkmale zu wählen, die genauso gut sind wie die von Hand ausgewählten Merkmale und deren Zahl geringer ist als in Chadha und Gerndt (2019) angegeben.

Im Folgenden wird nun analysiert, wie sich die Verfahren verhalten, wenn nicht mehr mit einem wohldefinierten Benchmark gearbeitet wird, sondern mit einer Menge an unterschiedlichen Benchmarks.

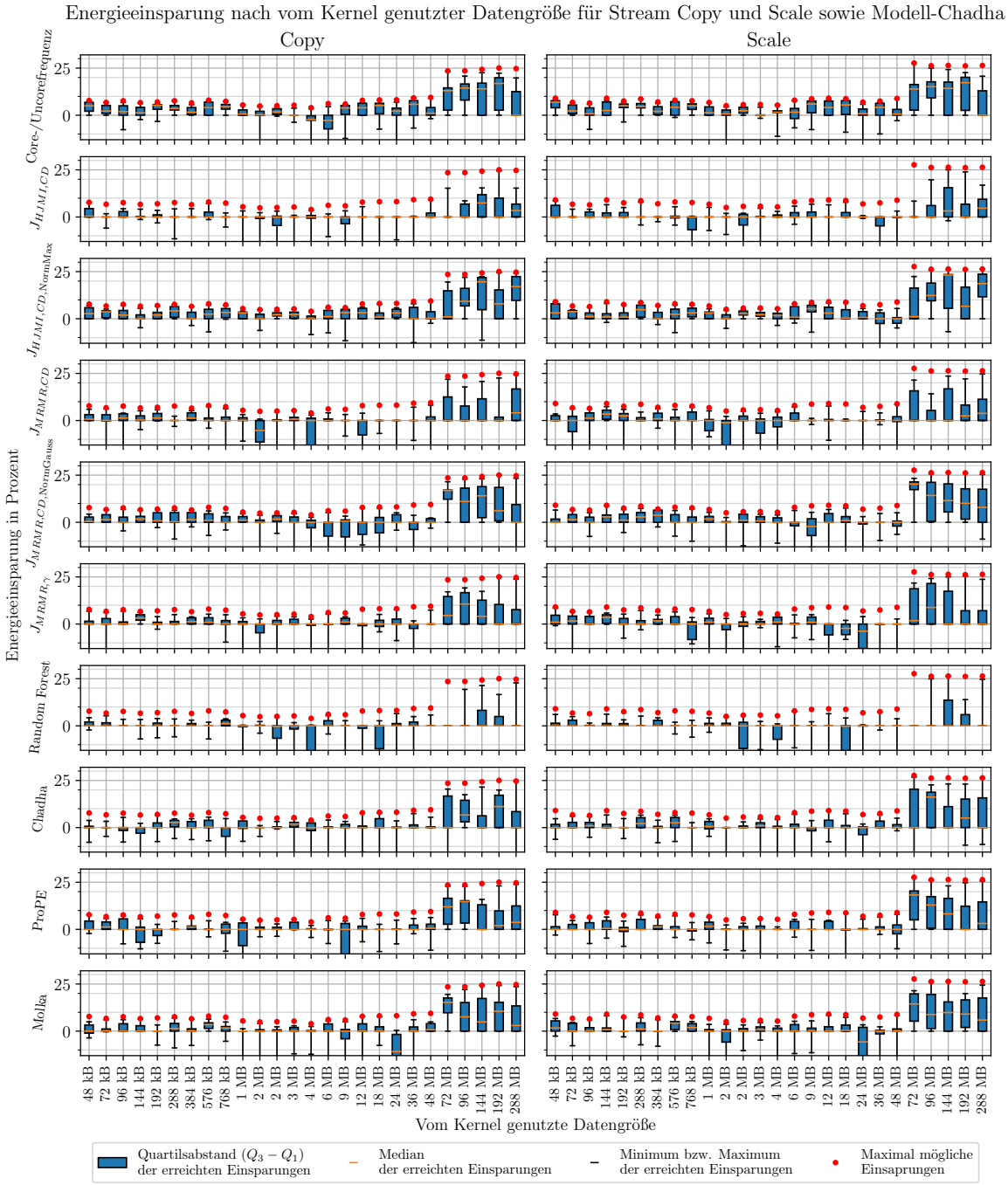


Abbildung 5.11: Ergebnisse der Kreuzvalidierung für das Modell-Chadha. Gezeigt werden Stream-Copy und -Scale als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

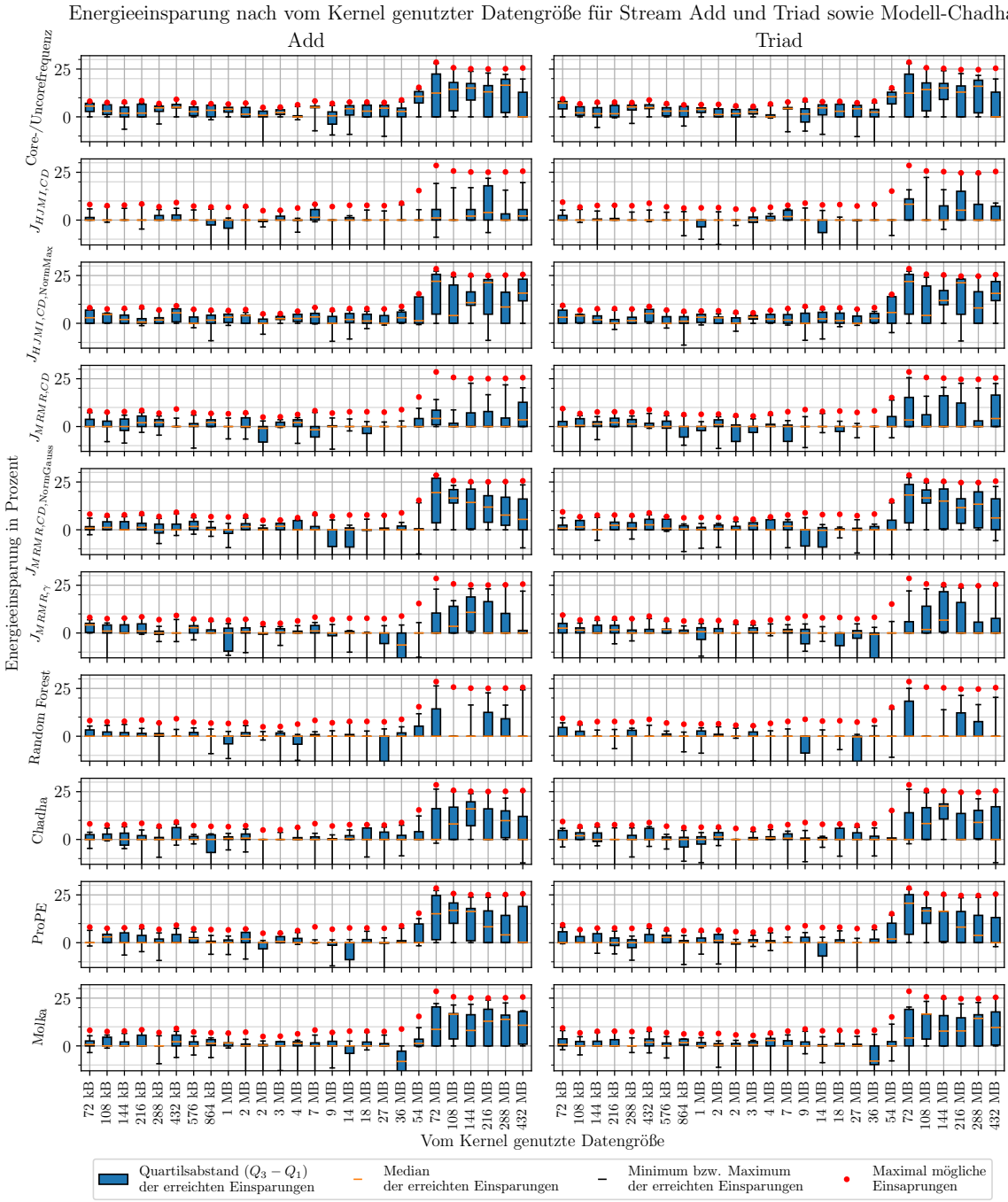


Abbildung 5.12: Ergebnisse der Kreuzvalidierung für das Modell-Chadha. Gezeigt werden Stream-Add und -Triad als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

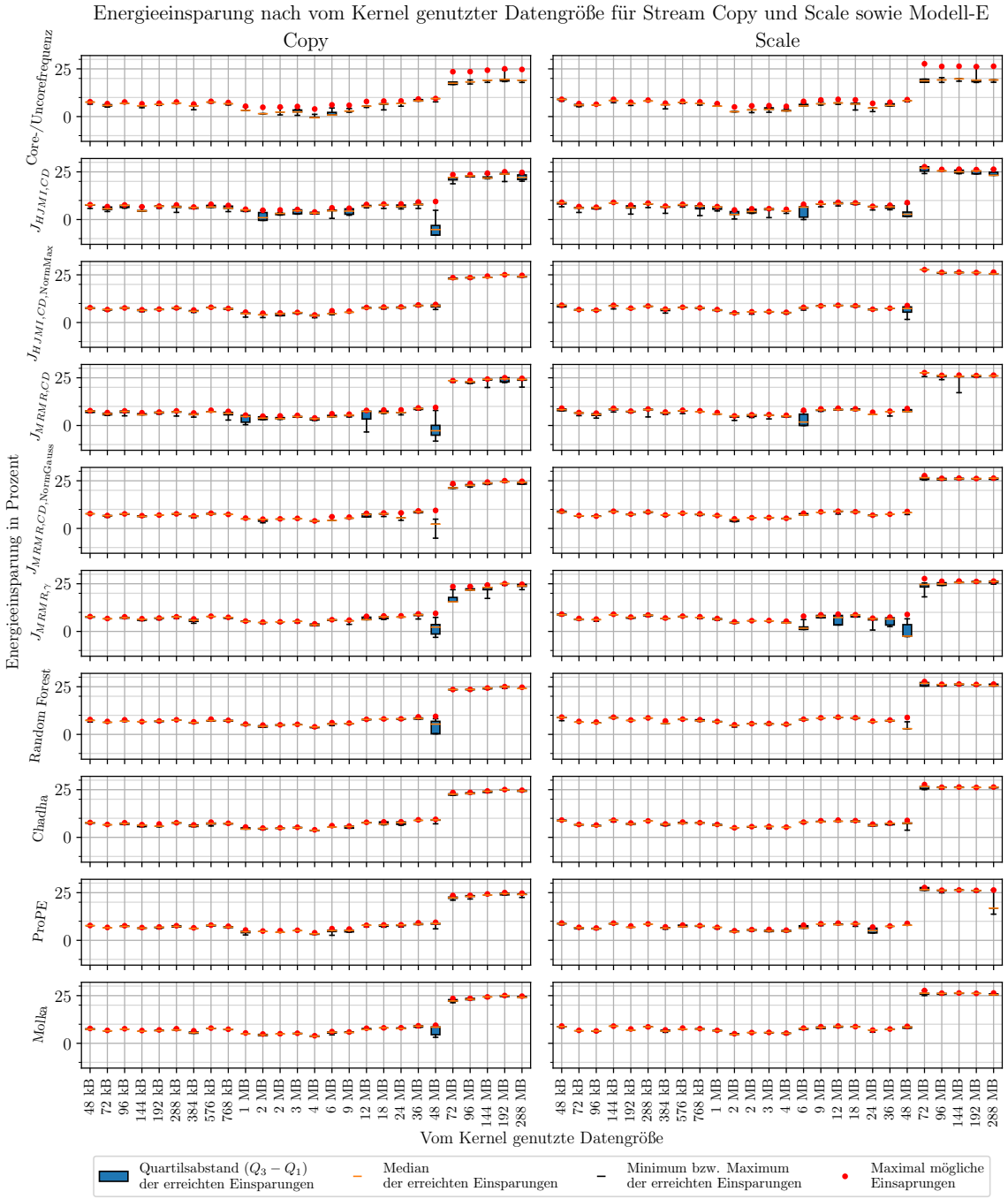


Abbildung 5.13: Ergebnisse der Kreuzvalidierung für das Modell-E. Gezeigt werden Stream-Copy und -Scale als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

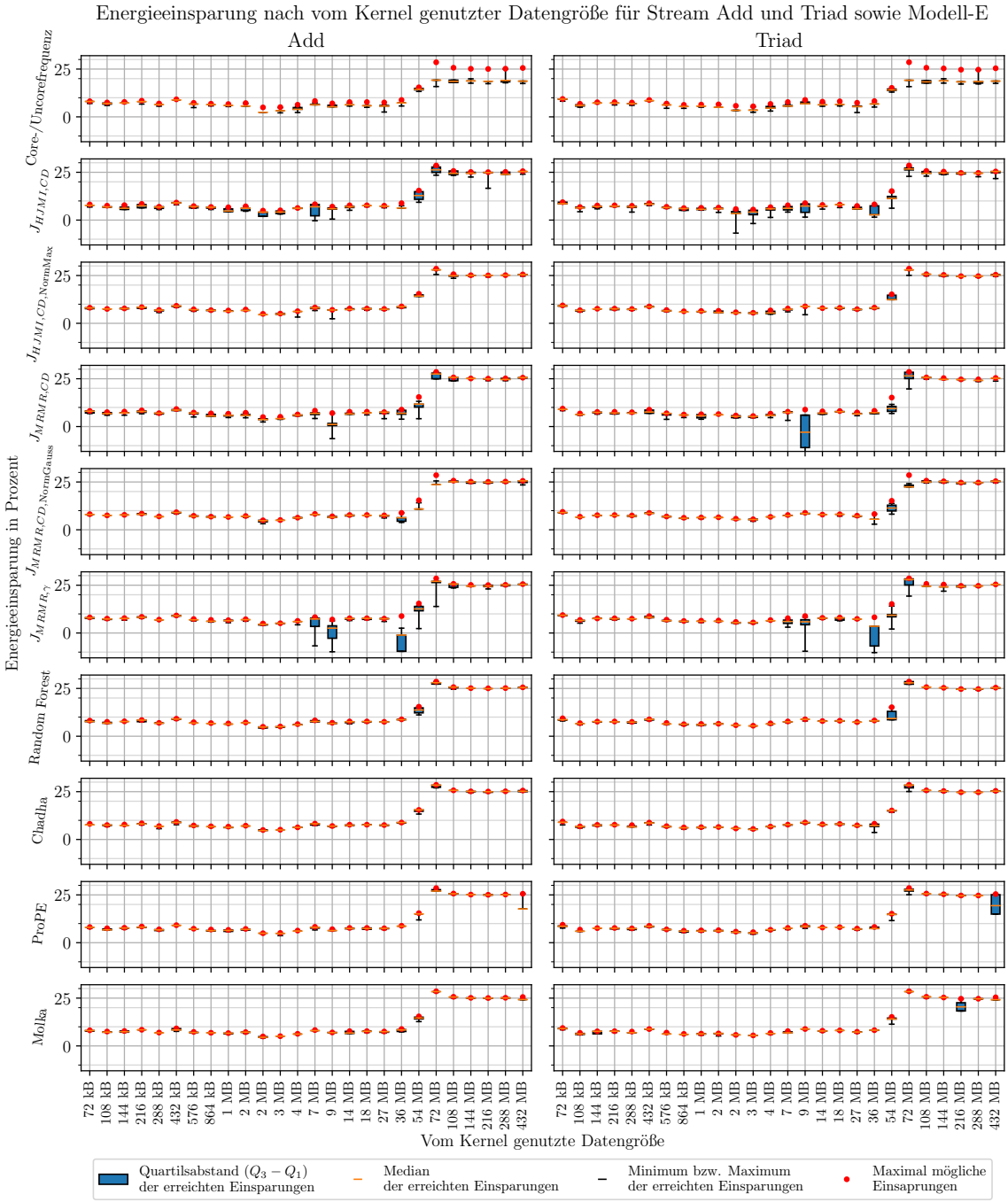


Abbildung 5.14: Ergebnisse der Kreuzvalidierung für das Modell-E. Gezeigt werden Stream-Add und -Triad als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

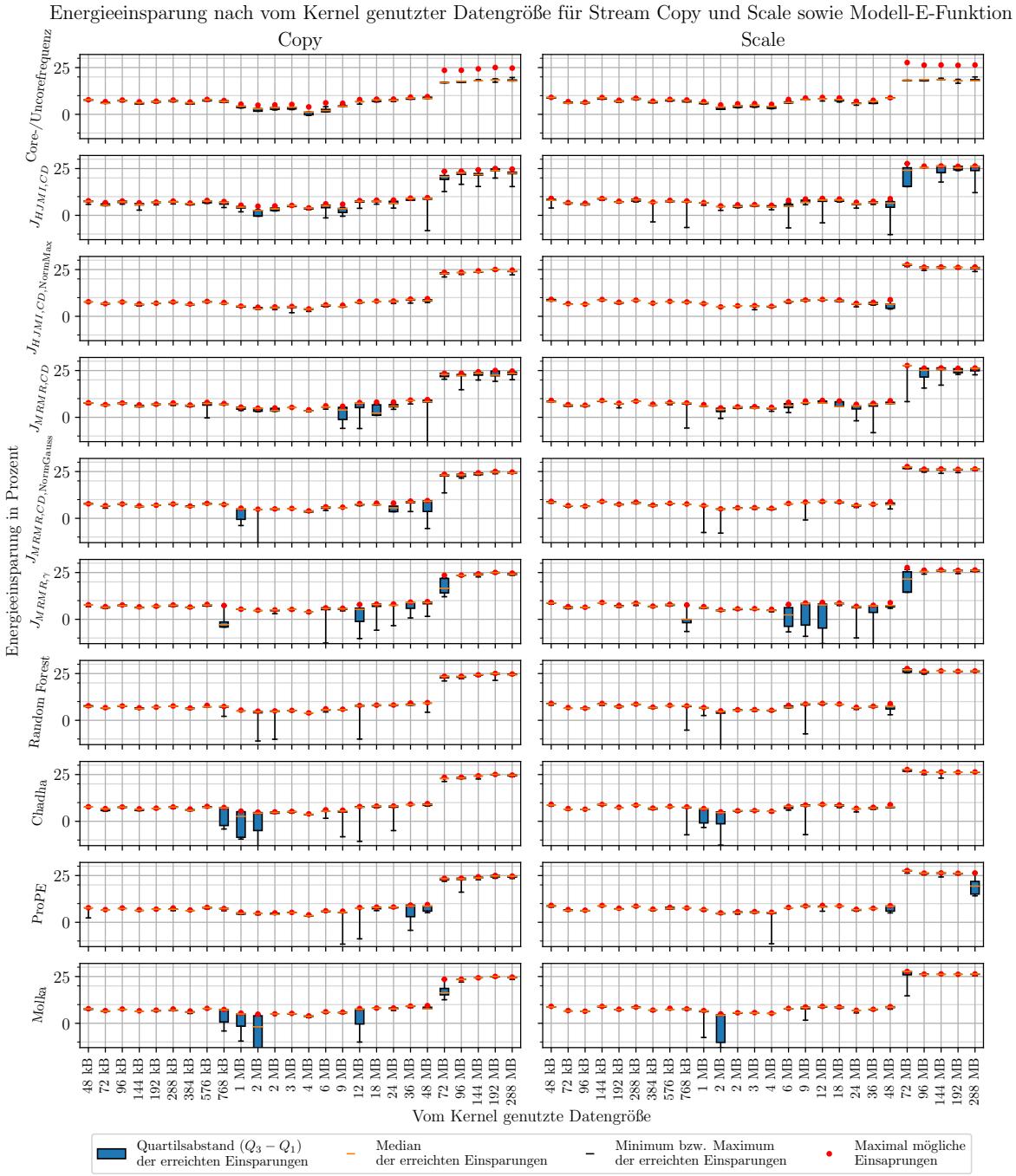


Abbildung 5.15: Ergebnisse der Kreuzvalidierung für das Modell-E-Funktion. Gezeigt werden Stream-Copy und -Scale als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

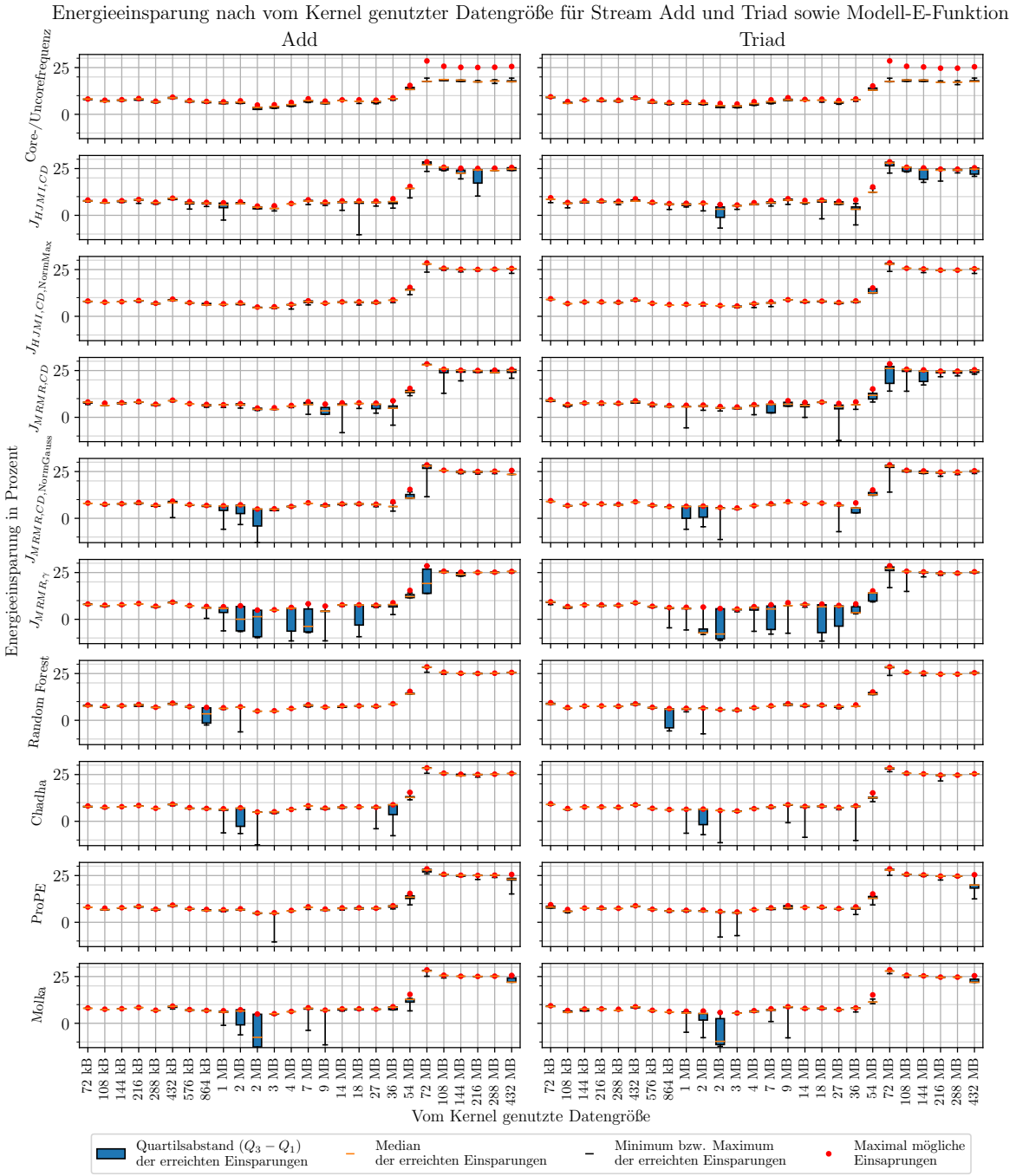


Abbildung 5.16: Ergebnisse der Kreuzvalidierung für das Modell-E-Funktion. Gezeigt werden Stream-Add und -Triad als relative Energieeinsparung in Prozentpunkten. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.9 sortiert.

5.2 Verschiedene Benchmarks

Wie in Kapitel 4.3.2, S. 65, beschrieben besteht der zweite Datensatz aus verschiedenen Benchmarks. Damit ist der Datensatz generischer. Die Benchmarks haben sehr verschiedene Regionen mit unterschiedlichen, zum Teil aber auch gleichen oder ähnlichen optimalen Frequenzen. Sie sollten das Training eines generischen neuronalen Netzes für die spätere Anwendung mit realen Programmen erlauben.

5.2.1 Ausgewählte Merkmale

Im Gegensatz zu den Benchmarks basierend auf Stream lassen sich die unterschiedlichen Benchmarks nicht ordnen. Damit ist eine Analyse wie in Kapitel 5.1.1 nicht möglich. Dennoch soll hier ein kurzer Überblick über die gewählten Events gegeben werden sowie exemplarisch jeweils das erste Event analysiert werden. Die Beschreibung der übrigen Events kann den Handbüchern entnommen werden: Events der Klasse `hsw_ep` können in Intel (2019c, S. 3605, Vol. 3B 19-1 ff.) gefunden werden, alle anderen lassen sich in Intel (2015) finden. Tabelle 5.1 bis Tabelle 5.6 geben die durch die unterschiedlichen Merkmalsauswahlverfahren gewählten Merkmale an.

Zunächst fällt auf, dass sowohl bei $J_{HJMI,CD}$ als auch bei $J_{HJMI,CD,\text{NormMax}}$, $J_{MRMR,CD}$ und $J_{MRMR,CD,\text{NormGauss}}$ das erste gewählte Merkmal das gleiche ist:

hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING.

Das Verhalten ist zu erwarten, da alle vier Merkmalsauswahlverfahren im ersten Schritt die gleiche Berechnung ausführen, welche zum Teil nur unterschiedlich normiert wird. Das Event selbst hat im Handbucheintrag (Intel, 2019c, S. 3686, 19-82 Vol 3B) für die Prozessorgeneration des Testsystems keine Beschreibung. Erst für die nachfolgende Generation ist angegeben, dass das Event Takte mit ausstehenden Speicherladeoperationen zählt (Intel, 2019c, S. 3674, 19-70 Vol 3B). Das Event deutet also auf eine durch den Speicher limitierte Region hin. Es ist naheliegend, dass das einen Einfluss auf den Energieverbrauch und die optimale Einstellung einer Region hat. Interessant ist, dass dieser Einfluss größer als der der Core- oder Uncorefrequenz zu sein scheint.

Als Nächstes werden bei allen vier Verfahren die Core- und die Uncorefrequenz als Merkmale gewählt. Da beide Frequenzen einen wesentlichen Einfluss auf den Energiemehrverbrauch eines Programmes haben, ist diese Auswahl ebenfalls naheliegend. Die vier Verfahren weichen aber bereits in der Reihenfolge, in der sie die Frequenzen wählen, voneinander ab.

$J_{MRMR,\gamma}$ wählt ebenfalls zunächst ein Event, bevor die Core- und Uncorefrequenz folgen:

hswep_unc_cbo::UNC_C_RXR_INSERTS:IRQ_REJECTED.

Laut Handbuch werden die Allokationen pro Takt in der „IRQ_REJECTED Ingress Queue“ gezählt (Intel, 2015, S. 58 f.). IRQ steht wiederum für „Ingress Request Queue on AD Ring“ und ist mit Anfragen vom Kern assoziiert (Intel, 2019c, S. 49). AD steht für „Address Ring“, der sich unter anderem mit Lese- und Schreibanfragen des Kernes beschäftigt. Es lässt sich also schlussfolgern, dass das Event die zurückgewiesenen Eintrittsanfragen von Lese- und Schreibanfragen aus den Kernen zählt. Ob diese auf den L3-Cache oder den DRAM bezogen sind, ist zunächst unklar. Der Zusammenhang mit dem Energieverbrauch und der optimalen Frequenz lässt sich aus der Beschreibung also nur schwer abschätzen.

Nur bei Random Forest sind die Core- und Uncorefrequenz die ersten gewählten Merkmale, gefolgt von:

hswep_unc_cbo::UNC_C_RXR_OCCUPANCY:IRQ.

Das Event zählt die Anzahl der Einträge in der entsprechenden Queue für jeden Takt. Naheliegend ist, dass viele Einträge in der Queue dafür sorgen, dass diese voll ist und das Hinzufügen neuer Einträge zurückgewiesen wird. Inwieweit das mit dem Energieverbrauch und der optimalen Frequenz zusammenhängt, lässt sich daraus wieder schwer abschätzen.

Nummer	I_{CD} mit E_n	Name des Merkmals
1	0.307	`hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING`
2	0.226	`Corefrequenz`
3	0.286	`Uncorefrequenz`
4	0.134	`hswep_unc_cbo::UNC_C_RXR_OCCUPANCY:PRQ`

Tabelle 5.1: Mittels $J_{HJMI,CD}$ ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Nummer	$I_{CD,\text{NormMax}}$ mit E_n	Name des Merkmals
1	0.139	`hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING`
2	0.102	`Corefrequenz`
3	0.129	`Uncorefrequenz`

Tabelle 5.2: Mittels $J_{HJMI,CD,\text{NormMax}}$ ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Nummer	I_{CD} mit E_n	Name des Merkmals
1	0.307	`hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING`
2	0.286	`Uncorefrequenz`
3	0.226	`Corefrequenz`

Tabelle 5.3: Mittels $J_{MRMR,CD}$ ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Nummer	$I_{CD,\text{NormGauss}}$ mit E_n	Name des Merkmals
1	0.677	`hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING`
2	0.660	`Uncorefrequenz`
3	0.603	`Corefrequenz`
4	0.672	`hsw_ep::INST_RETIRED:ALL`
5	0.442	`hswep_unc_ha::UNC_H_BT_OCCUPANCY`
6	0.670	`hswep_unc_ha::UNC_H_RING_AK_USED:CW_EVEN`
7	0.669	`hsw_ep::UOPS_ISSUED:STALL_CYCLES`
8	0.673	`hsw_ep::INSTRUCTION_RETIRED`
9	0.436	`hswep_unc_pcu::UNC_P_PKG_RESIDENCY_C1E_CYCLES`
10	0.667	`hswep_unc_cbo::UNC_C_RING_AD_USED:UP_EVEN`
11	0.646	`hswep_unc_cbo::UNC_C_RXR_OCCUPANCY:IRQ`

Tabelle 5.4: Mittels $J_{MRMR,CD,\text{NormGauss}}$ ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Nummer	I_γ mit E_n	Name des Merkmals
1	0.433	hswep_unc_cbo::UNC_C_RXR_INSERTS:IRQ_REJECTED
2	0.342	Uncorefrequenz
3	0.207	Corefrequenz
4	0.370	hsw_ep::MEM_LOAD_UOPS_RETIRED:L1_HIT
5	0.287	hswep_unc_qpi::UNC_Q_DIRECT2CORE:FAILURE_MISS
6	0.426	hswep_unc_cbo::UNC_C_RXR_OCCUPANCY:IRQ
7	0.174	hswep_unc_cbo::UNC_C_RXR_INSERTS:PRQ
8	0.318	hsw_ep::L1D_PEND_MISS:OCCURRENCES
9	0.396	hswep_unc_cbo::UNC_C_RXR_IPQ_RETRY:ANY
10	0.316	hsw_ep::BR_MISP_RETIRED:NEAR_TAKEN
11	0.276	hsw_ep::PAGE_WALKER_LOADS:DTLB_L1
12	0.348	hswep_unc_qpi::UNC_Q_RXL_OCCUPANCY_DRS:VN0

Tabelle 5.5: Durch Variable Selection $J_{MRMR,\gamma}$ ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Nummer	Importance	Name des Merkmals
1	0.175	Corefrequenz
2	0.112	Uncorefrequenz
3	0.063	hswep_unc_cbo::UNC_C_RXR_OCCUPANCY:IRQ
4	0.046	hswep_unc_cbo::UNC_C_RXR_INSERTS:IRQ_REJECTED
5	0.031	hswep_unc_cbo::UNC_C_RXR_IRQ_RETRY:ANY
6	0.027	hswep_unc_cbo::UNC_C_RXR_IRQ_RETRY:ADDR_CONFLICT
7	0.019	hsw_ep::CYCLE_ACTIVITY:STALLS_LDM_PENDING
8	0.011	hsw_ep::UOPS_RETIRED:ALL
9	0.007	hsw_ep::PAGE_WALKER_LOADS:DTLB_L2
10	0.007	hsw_ep::CYCLE_ACTIVITY:CYCLES_NO_EXECUTE
15	0.004	hswep_unc_qpi::UNC_Q_RXL_CYCLES_NE
19	0.004	hswep_unc_sbo::UNC_S_FAST_ASSERTED
21	0.004	hswep_unc_qpi::UNC_Q_RXL_INSERTS_DRS:VN0
22	0.004	hswep_unc_sbo::UNC_C_BOUNCE_CONTROL
26	0.003	hswep_unc_imc::UNC_M_CAS_COUNT:RD_UNDERFILL
33	0.003	hswep_unc_ha::UNC_H_SNOOP_RESP:RSPS
35	0.002	hswep_unc_ha::UNC_H_TRACKER_CYCLES_NE:LOCAL
36	0.002	hswep_unc_ha::UNC_H_CONFLICT_CYCLES
37	0.002	hswep_unc_qpi::UNC_Q_TXR_AK_NDR_CREDIT_ACQUIRED:VN1
43	0.002	hswep_unc_ha::UNC_H_RING_AK_USED:CW_EVEN
47	0.002	hswep_unc_qpi::UNC_Q_RXL0_POWER_CYCLES
60	0.001	hswep_unc_imc::UNC_M_DRAM_PRE_ALL
63	0.001	hswep_unc_sbo::UNC_S_RING_BOUNCES:AD_CACHE
74	0.001	hswep_unc_sbo::UNC_S_RXR_OCCUPANCY:BL_CRD
107	0.001	hswep_unc_imc::UNC_M_WR_CAS_RANK0:BANK4
115	0.001	hswep_unc_imc::UNC_M_RD_CAS_RANK0:BANK4

Tabelle 5.6: Durch Random Forest ausgewählte Merkmale in der Reihenfolge, in der sie gewählt wurden.

Es scheint so, dass für die ausgewählten Benchmarks die Core- und Uncorefrequenz nicht notwendigerweise das wichtigste Kriterium sind. Dennoch erkennen alle Verfahren die Core- und Uncorefrequenz als relevante Merkmale. Wie sich darüber hinaus zeigt, lässt sich der Einfluss der gewählten Merkmale auf den Energieverbrauch und die optimale Frequenz nur bedingt aus deren Beschreibung abschätzen. Eine analytische Auswertung aller gewählten Merkmale ist daher hier nicht zielführend.

$J_{HJMI,CD}$, $J_{HJMI,CD,\text{NormMax}}$ und $J_{MRMR,CD}$ wählen mit vier bzw. drei Merkmalen die wenigsten Merkmale aus. $I_{MRMR,CD,\text{NormGauss}}$ und $I_{MRMR,\gamma}$ wählen mit 11 respektive 12 Merkmalen ungefähr die gleiche Anzahl von Merkmalen aus. Interessant ist, dass die Normierung von I_{CD} sich mehr auf die Merkmalsauswahl auswirkt als das Verfahren zur Auswahl, z. B. HJMI oder MRMR. Die mit Abstand meisten Merkmale werden durch Random Forest ausgewählt.

5.2.2 Ergebnisse des Trainings

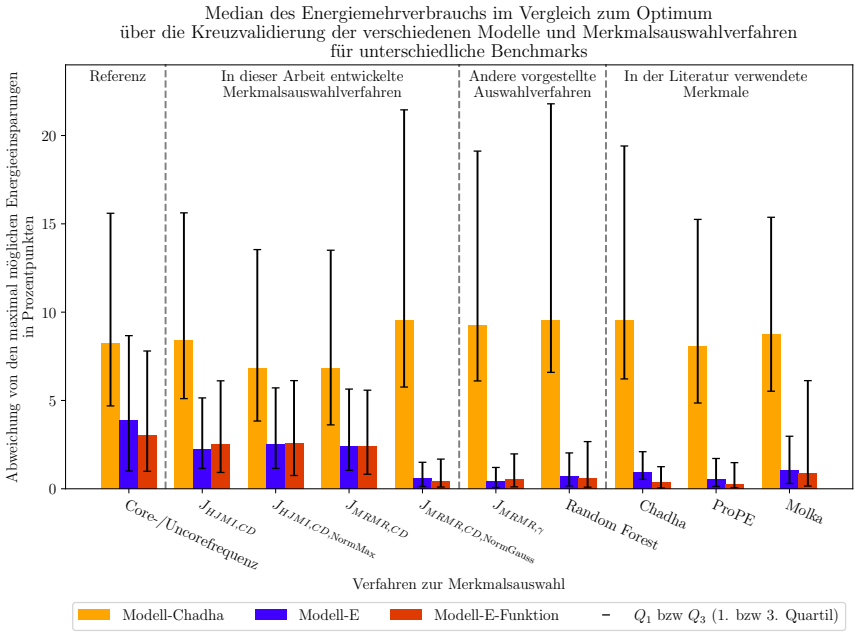


Abbildung 5.17: Median des Energiemehrverbrauchs bezogen auf die maximal möglichen Energieeinsparungen für verschiedene Benchmarks. Je geringer die Abweichung vom Optimum, also der Median, ist, desto besser ist das neuronale Netz zusammen mit den Merkmalsauswahlverfahren. Der Median sowie die Quartile wurden über alle Kreuzvalidierungsversuche der jeweiligen Modelle gebildet. Bei dem Merkmalsauswahlverfahren Core-/Uncorefrequenz wurden zum Training des Modells neben der Core- und Uncorefrequenz keine weiteren Merkmale verwendet. Jedes Training mit Merkmalen sollte besser sein als diese Referenz. Die Verfahren Chadha, ProPE und Molka verweisen auf die in Arbeiten von Chadha und Gerndt (2019), an Mey et al. (2019) und Molka et al. (2017) ausgewählten Merkmale.

Abbildung 5.17 zeigt wieder die Ergebnisse über alle Iterationen des Kreuzvalidierungsverfahrens. In diesem Fall kann das Modell-Chadha in einigen Fällen besser abschneiden als die Referenz, bei der nur mit der Core- und Uncorefrequenz trainiert wird. Die Schwankung und der Fehler, also der Median der Abweichung vom Optimum, sind aber immer noch höher als bei Modell-E und Modell-E-Funktion. Die besten Ergebnisse mit Merkmalen aus den Merkmalsauswahlverfahren werden hier mit $J_{MRMR,\gamma}$, $J_{MRMR,CE,\text{NormGauss}}$ und Random Forest erreicht, also den drei Verfahren, die die meisten Merkmale ausgewählt haben. Die

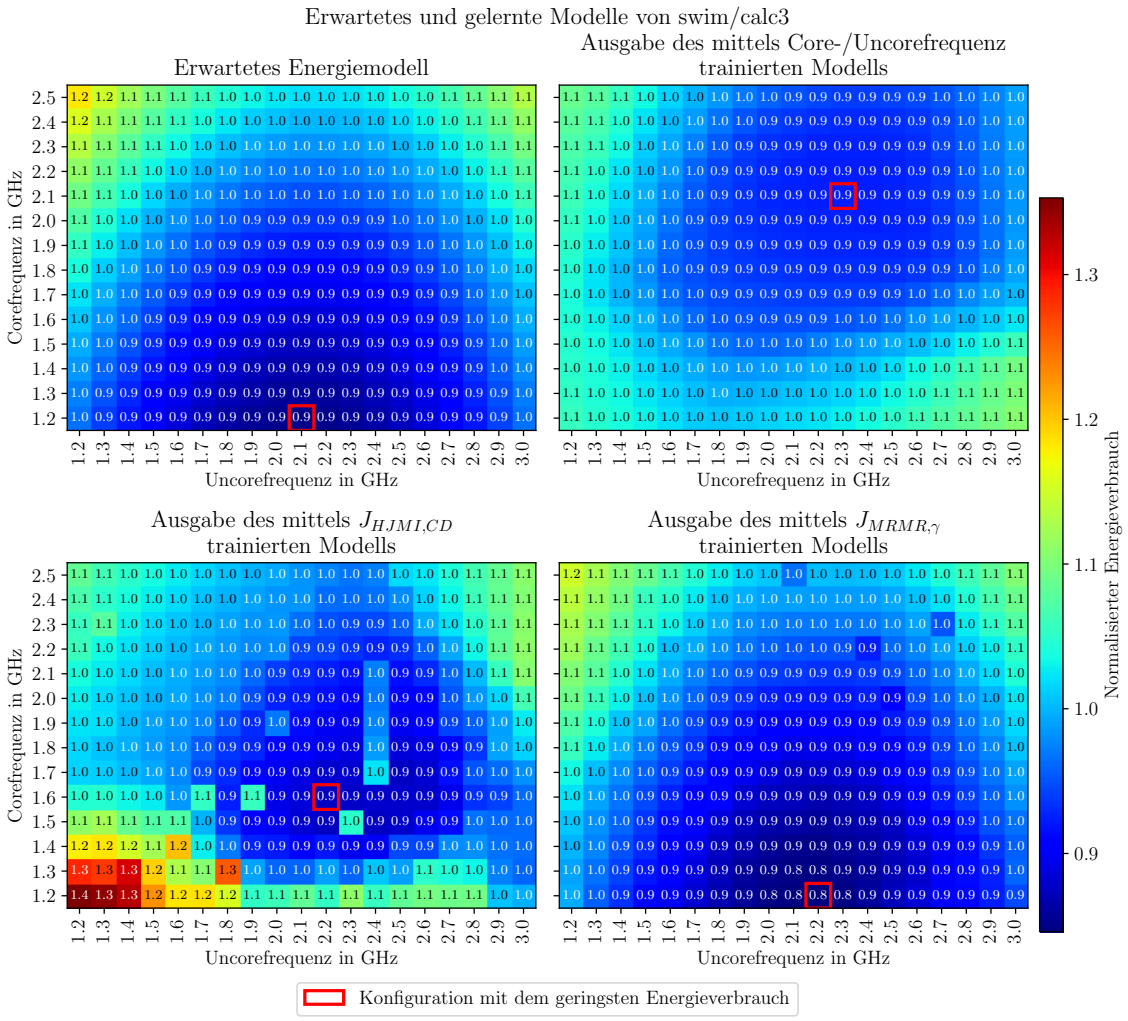


Abbildung 5.18: Auswahl der durch Modell-E generierten normierten Energiemodelle und eines zu erwartenden Energiemodells für verschiedene Merkmale, mit denen Modell-E trainiert wurde

Merkmale aus der Literatur schneiden ebenfalls gut ab: So hat Modell-E-Funktion zusammen mit den Merkmalen von ProPE (an Mey et al., 2019) den geringsten Fehler. Allerdings ist der Abstand vom 1. zum 3. Quantil etwas größer als bei Modell-E und den Merkmalen von $J_{MRMR,\gamma}$.

Beim Blick auf die detaillierten Ergebnisse in Abbildung 5.19 bis Abbildung 5.21 zeigt sich der Grund. Abbildung 5.21 zeigt für ProPE und Modell-E-Funktion einige Ausreißer, die nicht nur zu einer Abweichung vom Optimum führen. So kann es vorkommen, dass signifikant mehr Energie verbraucht wird, als wenn keine Optimierung vorgenommen wird und die Standardeinstellungen verwendet werden. Zwar kann es mit $J_{MRMR,\gamma}$ und Modell-E sowie Abbildung 5.20 ebenfalls dazu kommen, dass mehr Energie verbraucht wird, allerdings sind diese Fälle seltener und weniger drastisch. Es werden deshalb für die folgende Evaluation des gesamten Frameworks die durch $J_{MRMR,\gamma}$ gewählten Merkmale verwendet.

Abbildung 5.18 stellt noch einmal anschaulich den Unterschied dar, der durch verschiedene Merkmale beim gleichen Modell entstehen kann. Für jede dargestellte Frequenz wurde aus den für die Region vorliegenden Messungen der ausgewählten Eventraten je eine Messung zufällig gezogen. Damit lässt sich beurteilen, wie gut die Ergebnisse für die jeweilige Kombination aus Merkmalen und Modellen über verschiedene Iterationen eines Benchmarks sind.

Scheint das Bild eher stetig, gibt es entweder in den Daten wenig Rauschen oder das Modell geht gut mit diesem Rauschen um. Gibt es Sprünge im Bild, kann das Modell das Rauschen in den Daten nicht kompensieren.

Es zeigt sich, dass das nur mittels Core- und Uncorefrequenz trainierte Modell wenig mit den Erwartungen für diesen Benchmark zu tun hat. Es wurde, wie schon erwähnt, eher ein mittleres Energiemodell erstellt, das für diesen Benchmark nicht passend ist. Die mittels $J_{HJMI,CE}$ gewählten Merkmale erzeugen zusammen mit Modell-E ein eher unstetiges Bild, das zwar Gemeinsamkeiten mit dem erwarteten Bild hat, aber an den entscheidenden Stellen Fehler aufweist, sodass die gefundene optimale Frequenz in einiger Distanz zur erwarteten optimalen Frequenz liegt. Lediglich das mittels $J_{MRMR,\gamma}$ trainierte Modell-E bildet das erwartete Energiemodell ab, und die optimale Frequenz liegt nahe der erwarteten optimalen Frequenz.

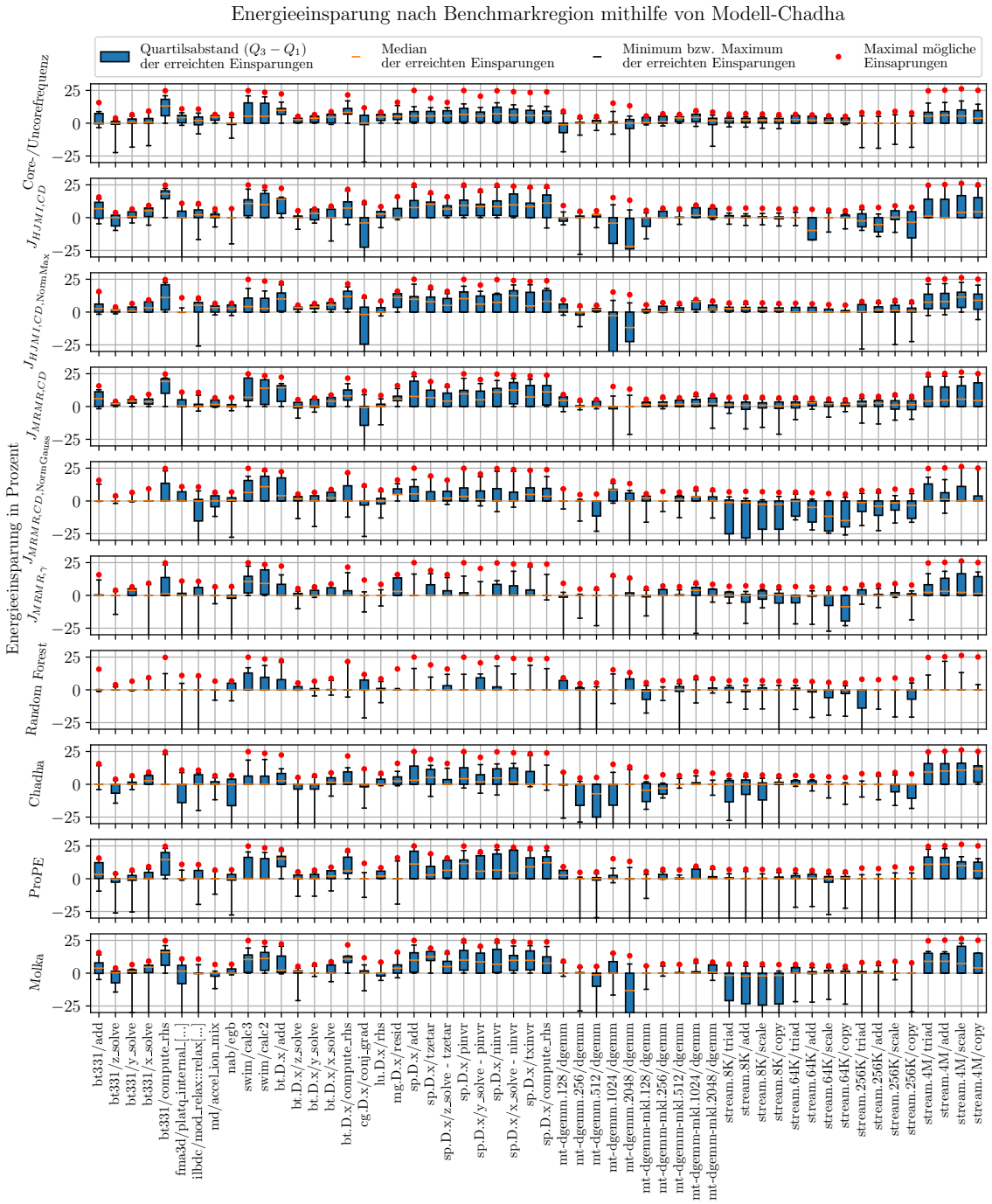


Abbildung 5.19: Ergebnisse der Kreuzvalidierung für das Modell-Chadha. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.17 sortiert.

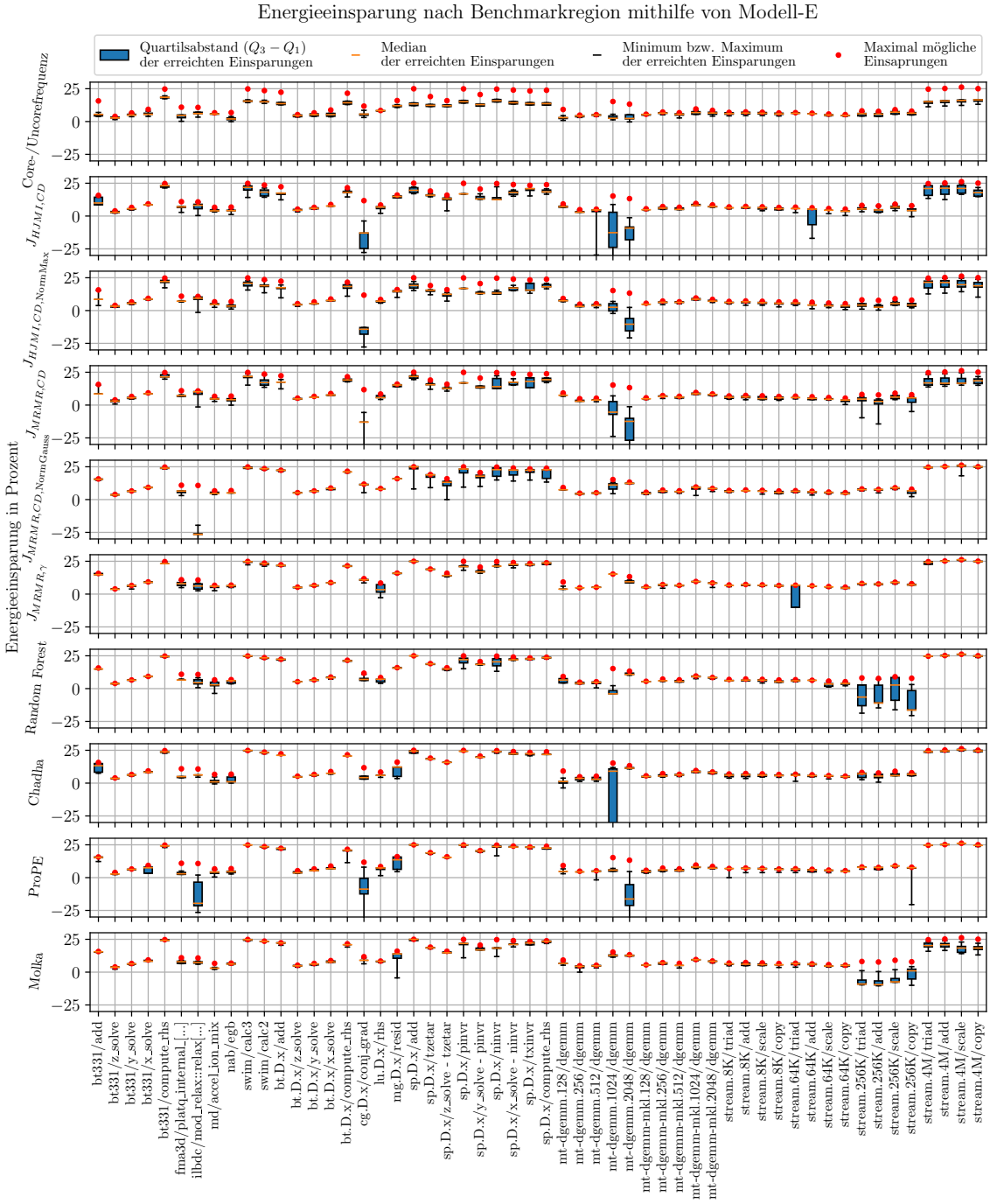


Abbildung 5.20: Ergebnisse der Kreuzvalidierung für das Modell-E. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.17 sortiert.

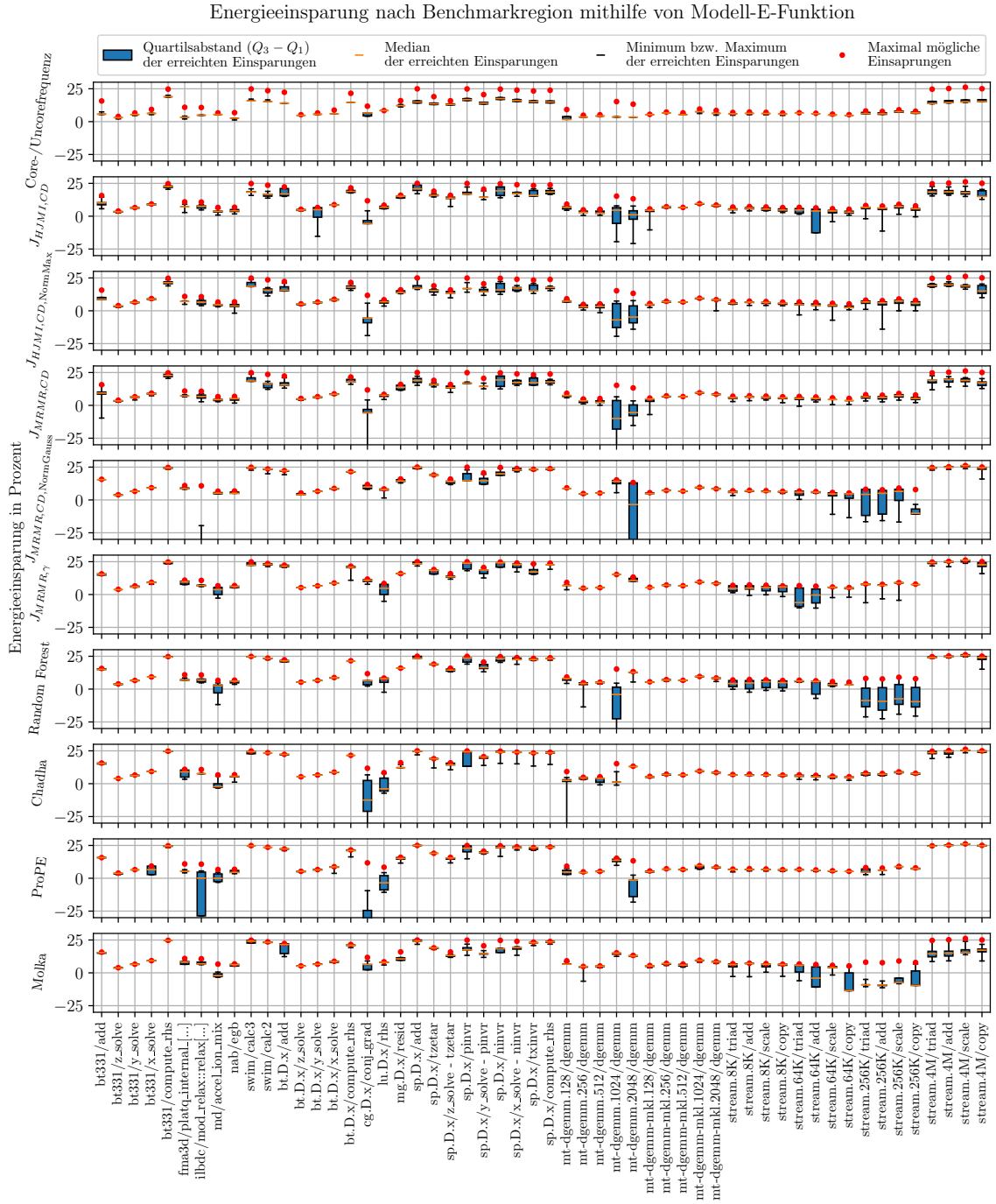


Abbildung 5.21: Ergebnisse der Kreuzvalidierung für das Modell-E-Funktion. Die Verfahren zur Merkmalsauswahl sind nach Abbildung 5.17 sortiert.

5.3 Energieoptimierung einer Anwendung

Nach der Evaluation der verschiedenen Netze und Merkmalsauswahlverfahren bleibt die Evaluation des kompletten Frameworks an einer Anwendung. Modell-E hat zusammen mit den Merkmalen, die mittels $J_{MRMR,\gamma}$ gewählt wurden, den geringsten Fehler mit der kleinsten Schwankung des Fehlers bei der Erstellung des Energiemodells gehabt. Modell-E wird nun erneut mit den ausgewählten Eventraten aller Benchmarks trainiert. Die Anwendung des trainierten Modells erfolgt mithilfe der RRL. Zum Testen wird als Vertreter für ein beliebiges Programm die Kripke Mini-App (Kunen et al., 2020) verwendet. Diese wurde bereits zur Evaluation der Optimierung von Anwendungen mithilfe eines von mir umgesetzten Reinforcement-Ansatzes verwendet (Gocht et al., 2019). Wie in Kapitel 2.1.4, S. 22, beschrieben wird für einen Zustands-Aktionsraum, bestehend aus Frequenzen und möglichen Änderungen der Frequenzen, eine erwartete Belohnung für eine Aktion gelernt. Die Belohnung wird dabei basierend auf der eingesparten Energie berechnet. Die Ergebnisse, die mit diesem Ansatz erreicht werden können, sind zum Vergleich mit angegeben.

Quelltext 5.1 zeigt die Konfiguration, mit der Kripke genutzt wurde.

Quelltext 5.1: Konfiguration der verwendeten Mini-App Kripke.

```
kripke.exe --groups 48 --gset 2 --quad 96 --dset 96 --legendre 6\
    --zones 96,48,48 --procs 1,1,1 --niter 500
```

Die RRL wurde so konfiguriert, dass für nicht relevante Regionen die initiale Konfiguration von Core- und Uncorefrequenz von 2,5 GHz respektive 3,0 GHz verwendet wird. Zur Ermittlung der Ergebnisse in Tabelle 5.7, Tabelle 5.8 und Abbildung 5.22 wurde die Mini-App 10-mal ausgeführt und der Mittelwert des Energieverbrauchs verwendet. Zum Vergleich wurden in Tabelle 5.7 2,5 GHz Core- und 3,0 GHz Uncorefrequenz gesetzt, während in Tabelle 5.8 die Wahl der Core- und Uncorefrequenz dem Prozessor überlassen wurde. Die Messung des Energieverbrauchs der kompletten Anwendung wurde mithilfe von MetricQ (Ilsche et al., 2019) durchgeführt.

Konfiguration	Energieunterschied	Laufzeitunterschied
Neuronales Netz	14 %	-5 %
Reinforcement-Ansatz	20 %	-8 %

Tabelle 5.7: Energie- und Laufzeitunterschied von Kripke für zwei Optimierungsansätze im Vergleich zu 2,5 GHz Core- und 3,0 GHz Uncorefrequenz, was die höchste verwendete Core- und Uncorefrequenz repräsentiert.

Konfiguration	Energieunterschied	Laufzeitunterschied
Neuronales Netz	24 %	-7 %
Reinforcement-Ansatz	30 %	-10 %

Tabelle 5.8: Energie- und Laufzeitunterschied von Kripke für zwei Tuningansätze, im Vergleich zu einer freien Corefrequenz (auch *Turbo* genannt) und einer freien Uncorefrequenz.

Es zeigt sich, dass mit dem Reinforcement-Ansatz im Schnitt mehr Energie gespart werden kann als mit dem auf einem neuronalen Netz basierenden Ansatz. Gleichzeitig fällt in Abbildung 5.22 die Dynamik in der Laufzeit der Mini-App auf, so nicht der Reinforcement-Ansatz oder die Standardeinstellung verwendet wird. Sowohl der Reinforcement-Ansatz als auch die Standardeinstellung können die Frequenzen über die Laufzeit der Anwendung immer neu justieren. Das deutet darauf hin, dass die Mini-App nicht das exakt gleiche Verhalten

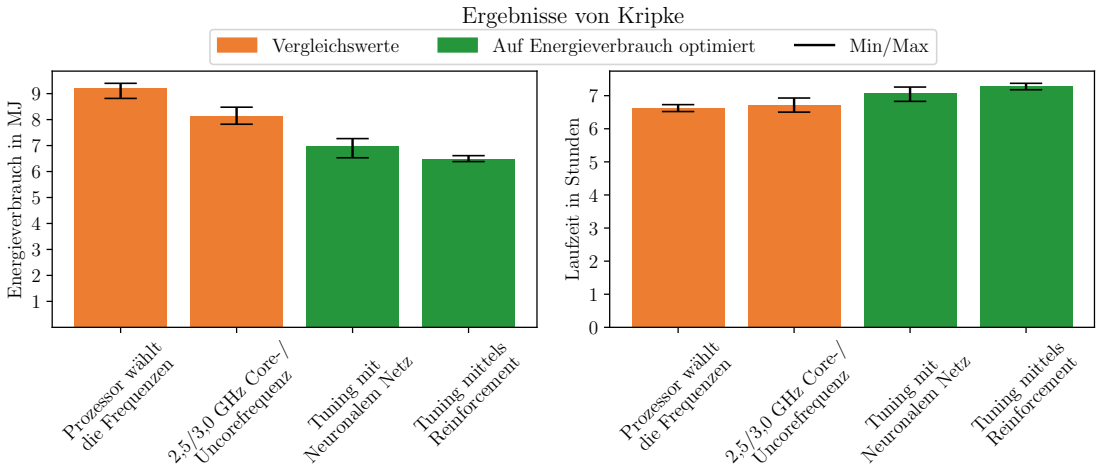


Abbildung 5.22: Mittelwert der verschiedenen Messungen für Energieverbrauch und Laufzeit von Kripke zusammen mit den minimal und maximal gemessenen Werten.

zwischen zwei Läufen aufweist. Damit ist auch wahrscheinlich, dass sich auch das Verhalten und damit das Optimum einzelner Regionen über einen Lauf von Kripke ändern. Der Reinforcement-Ansatz kann besser auf diese Änderungen reagieren, während der Ansatz basierend auf neuronalen Netzen einmalig am Anfang eine Einstellung festlegt, die für den Rest der Programmlaufzeit gilt.

Dennoch lassen sich mit den neuronalen Netzen signifikante Energieeinsparungen erreichen. So werden gegenüber 2,5 GHz Core- und 3,0 GHz Uncorefrequenz ca. 14 % Energieeinsparung erzielt, bei einer Verlängerung der Laufzeit von ca. 5 %. Wird für die Vergleichsmessung die Wahl der Core- und Uncorefrequenz dem Prozessor überlassen, ergeben sich Energieeinsparungen von ca. 24 %, bei einer Verlängerung der Laufzeit um ca. 6 %.

Abbildung 5.23 zeigt beispielhaft das Verhalten von Kripke mit und ohne Optimierung durch Modell-E mit $J_{MRMR,\gamma}$. Der Einfluss der Änderung von Core- und Uncorefrequenz auf die Leistungsaufnahme und die Laufzeit einer Funktion ist klar ersichtlich. Dies demonstriert die Leistungsfähigkeit des hier entworfenen Frameworks zum Optimieren der Energieeffizienz.

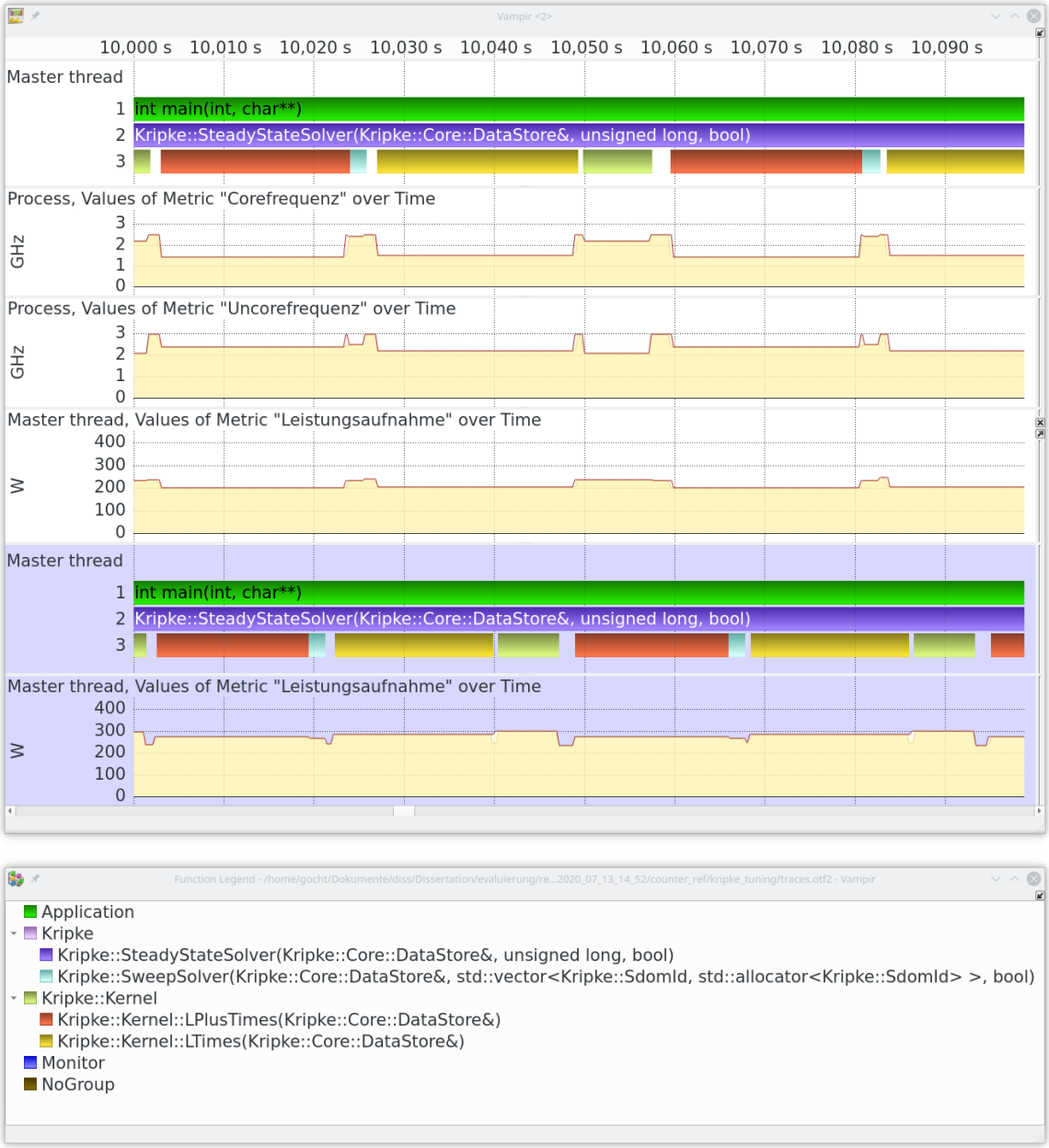


Abbildung 5.23: Vergleich eines Laufs von Kripke mit RRL (oben, weißer Hintergrund) und ohne RRL (unten, blauer Hintergrund). Für den Lauf mit RRL sind die ausgewählten Konfigurationen für Core- und Uncorefrequenz dargestellt, welche sich je nach gerade ausgeführter Funktion ändern. Für Regionen, die nicht optimiert wurden, wie den Bereich zwischen `Kripke::SweepSolver()` und `Kripke::Kernel::LTimes()`, wird die anfängliche Konfiguration von 2,5 GHz / 3,0 GHz Core-/Uncorefrequenz verwendet. Der Lauf ohne RRL nutzt die Einstellungen 2,5 GHz / 3,0 GHz Core-/Uncorefrequenz. Es zeigt sich, dass die Reduzierung der Frequenz zu einer Reduzierung der Leistungsaufnahme und zu einer Steigerung der Laufzeit führt.

6 Zusammenfassung und Ausblick

In den vorangegangenen Kapiteln wurde die Entwicklung und Umsetzung eines Frameworks zur Optimierung der Energieeffizienz auf Basis von Machine-Learning-Methoden vorgestellt, angefangen bei den notwendigen Grundlagen in Kapitel 2, über die Entwicklung eines neuen Ansatzes zur Merkmalsauswahl in Kapitel 3 bis hin zu den Details der Umsetzung in Kapitel 4 und der Auswertung in Kapitel 5. Dabei hat sich besonders in Kapitel 5 gezeigt, dass die Merkmalsauswahl wesentlich die Ergebnisse von neuronalen Netzwerken beeinflusst und deshalb ein zentraler Teil eines Frameworks sein muss, wenn es mit ebendiesen Netzwerken arbeitet.

Weiterentwicklung der Merkmalsauswahl

In Kapitel 3.5 zeigen sich anhand von theoretischen Beispielen die Unterschiede zwischen den verschiedenen Methoden zur Merkmalsauswahl. Daraus leiten sich auch die Notwendigkeit und die Vorteile der neu entwickelten Verfahren ab. Praktisch konnte dies in Kapitel 5 mit realen Daten bestätigt werden. Der in dieser Arbeit entwickelte Algorithmus basierend auf der historischen JMI mit Copula-Dichte normiert auf das Maximum, $J_{HJMI,CD,\text{NormMax}}$, ist bei einer guten Datenbasis, wie sie im Falle des Stream-Benchmarks vorhanden war, in der Lage, stabile Ergebnisse bei möglichst wenigen Merkmalen zu liefern. Gleiches gilt für die ebenfalls in dieser Arbeit abgeleitete normierte Minimum-Redundency Maximum-Relevance mit Copula-Dichte $J_{HJMI,CD,\text{NormGauss}}$.

Es zeigt sich allerdings auch, dass die von Schweizer und Wolff (1981) ebenfalls aus Copulas abgeleitete Maßzahl, γ, mit einer vielfältigeren Datenbasis als bei der beschriebenen Mischung verschiedener Benchmarks besser zurechtkommt. Damit gelingt es der Variablen Selection von Seth und Príncipe (2010), $J_{MRMR,\gamma}$, die diese Maßzahl nutzt, bessere Ergebnisse zu erzielen als $J_{HJMI,CD,\text{NormMax}}$, allerdings für den Preis, dass mehr Merkmale gewählt werden. $J_{MRMR,CD,\text{NormGauss}}$ schneidet nur unwesentlich schlechter ab als $J_{MRMR,\gamma}$ und wählt ähnlich viele Merkmale.

Aus den Ergebnissen aller drei Ansätze lässt sich ableiten, dass sich Copulas für die Auswahl nichtlinearer Merkmale eignen. Allerdings ergeben sich bei der Anwendung auf reale Daten auch einige Fragen. So ist zum Beispiel die Häufung einiger Eventraten bei 0 für die Schätzung der Copulas ein Problem. Dieses kann durch das Überlagern mit einem Rauschen zwar umgangen werden, allerdings stellt sich hier auch die Frage nach einer eleganteren Lösung. Es lohnt sich dafür auf jeden Fall die Forschung im Feld der Copulas weiter zu beobachten, zumal es sich hierbei auch um ein für die Statistik recht junges Themenfeld handelt, wie sich auch an dem Publikationsdatum von verschiedenen Arbeiten wie zum Beispiel Nelsen (2006, „An introduction to copulas“) ablesen lässt.

Für die praktische Anwendung ist aber auch eine andere Idee dieser Arbeit wesentlich: ein Abbruchkriterium für die Merkmalsauswahl. So erweist sich das Beobachten des Verhaltens der Zielgröße zum Rauschen als wirksame Methode, um zu evaluieren, ab welchem Punkt ausgewählte Merkmale keine Mehrinformationen mehr bieten und der Merkmalsauswahlalgorithmus abgebrochen werden kann.

Steigerung der Energieeffizienz von Anwendungen

Gleichzeitig stellt das vorgestellte Framework seine Fähigkeit zum Einsparen von Energie theoretisch wie praktisch unter Beweis. So zeigt sich auf der einen Seite in Kapitel 5, dass

mithilfe dieses Frameworks Machine-Learning-basierte Ansätze zum Energiesparen entwickelt, optimiert, aber auch real eingesetzt werden können. Entsprechend ist es gelungen, für die Mini-App Kripke mithilfe von neuronalen Netzen im Schnitt ca. 24 % Energie im Vergleich zur Standardkonfiguration eines HPC-Systems zu sparen.

Auf der anderen Seite ist es gelungen, den Aufwand beim Einsatz von Frameworks zum Energiesparen weiter zu senken. So ist es im Vergleich zu READEX und der Arbeit von Chadha und Gerndt (2019) nun nicht mehr notwendig, ein Tuning-Model für eine Anwendung zu generieren. Damit entfällt auch die Notwendigkeit, das Tuning-Model in einem separaten Schritt zu aktualisieren, nachdem ein Programm angepasst worden ist. Mit dem in dieser Arbeit entwickelten Framework kann ein Äquivalent des Tuning-Models während des produktiven Einsatzes des Programmes erzeugt werden. Dazu ist es ausreichend, ein neuronales Netzwerk für ein HPC-System zu trainieren, welches dann für verschiedene Programme verwendet werden kann, um optimale Einstellungen abzuschätzen.

Alternativ können mit dem Framework auch andere Ansätze, wie der in Gocht et al. (2019) vorgestellte Reinforcement-Ansatz, verwendet werden, was die Flexibilität des Frameworks demonstriert.

Ausblick

Wie jede Arbeit muss auch diese irgendwo aufhören, und verschiedene Ideen, Themen und Ansätze bleiben vorerst unerforscht. Das gilt im Speziellen mit Blick auf die Energieeffizienz, aber auch darüber hinaus mit Blick auf die anderen in der Arbeit angeschnittenen Themen wie die Merkmalsauswahl.

So bedürfen die Verfahren zur Merkmalsauswahl einer weiteren Analyse. Es stellt sich die Frage, ob es einen konkreten Punkt gibt, an dem eines der drei als gut befundenen Merkmalsauswahlverfahren zu bevorzugen ist, oder ob sich die Ansätze vereinen lassen. Vermutlich lassen sich auch die Abbruchbedingungen weiter verbessern.

Gleichzeitig stellt sich die Frage, ob nicht auch das neuronale Netzwerk weiter verbessert werden kann. Zwar sind die vorgestellten Merkmalsauswahlverfahren in der Lage, auf Annahmen über die Verteilung der Daten zu verzichten, und neuronale Netzwerke sind in der Lage, mit den ausgewählten Merkmalen umzugehen. Dennoch stellt die Auswahl spezifischer Aktivierungsfunktionen sowohl einen gewissen Aufwand als auch eine Limitation dar. Einen möglichen Ausweg bieten Aktivierungsfunktionen auf Basis von Padé-Approximationen, die 2020 von Molina et al. (2020) vorgeschlagen wurden. Mit den sogenannten PAUs (Padé-Activation-Units) lassen sich klassische Aktivierungsfunktionen wie ReLU oder tanh abbilden. Denkbar ist, dass diese eine höhere Flexibilität an den Tag legen und die Nichtlinearität der gewählten Merkmale besser ausnutzen können.

Auf der Seite des Frameworks ist es naheliegend, den Reinforcement-Ansatz mit dem Supervised-Ansatz zu vereinen, um die Vorteile beider zu verbinden. So könnte das neuronale Netz genutzt werden, um einen guten Ausgangspunkt für das Reinforcement-Learning zu generieren.

Daneben entwickelt sich aber auch das Hochleistungsrechnen weiter, was neue Anwendungsfelder eröffnet: So nutzten sieben der zehn schnellsten Hochleistungsrechner im November 2021 GPGPUs von Nvidia.[1] Auch bei diesen lassen sich die Frequenzen der Compute-Kerne und des Speichers anpassen (Nvidia, 2020). Wie bei Prozessoren kann hier durch das Anpassen der Frequenzen Energie gespart werden (Tang et al., 2019). Damit bietet es sich an, zu untersuchen, inwieweit die Ergebnisse dieser Arbeit genutzt oder erweitert werden können, um mit vier anstelle von zwei Dimensionen im Optimierungsraum umzugehen.

Darüber hinaus kann natürlich auch die Anwendung des Frameworks weiter vereinfacht werden. So wird immer noch eine geschickte Wahl der relevanten Regionen vorausgesetzt. Im

[1] Top 500 (Dongarra und Luszczek, 2011) November 2021.

Moment müssen entweder manuell die wesentlichen von den unwesentlichen Funktionen unterschieden werden, oder diese müssen durch einen Testlauf voneinander abgegrenzt werden. Einen weiteren Ansatz legt die Arbeit von Trommler (2016) nahe, bei der die Funktionen, die beim Betreten und Verlassen einer Funktion aufgerufen werden, zur Laufzeit überschrieben werden können. So könnten Funktionen, die eine Laufzeit von weniger als 50 ms oder 100 ms haben, einfach ignoriert werden. Alternativ könnte auch die Instrumentierung der Anwendung durch Tools wie Score-P intelligenter gestaltet werden. Dabei müsste während des Kompilierens die spätere Laufzeit abgeschätzt werden, was aufgrund der Dynamik eines Programmes eine Herausforderung ist.

Zusammenfassend lässt sich sagen, dass diese Arbeit einen Beitrag zur Energieeffizienz im Hochleistungsrechnen liefert. Gleichzeitig finden sich noch viele Themenfelder, deren weitere Bearbeitung lohnend erscheint.

Literatur

ACPI. (2019). *Advanced Configuration and PowerInterface (ACPI) Specification* (Techn. Ber.) (Zugegriffen 21.03.2022). UEFI Forum, Inc. https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf

AGEB. (2020). *Anwendungsbilanzen zur Energiebilanz Deutschland* (Techn. Ber.) (Zugegriffen 09.11.2020). AG Energiebilanzen e.V. https://ag-energiebilanzen.de/index.php?article_id=29&fileName=ageb_19_v3.pdf

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing. DOI: 10.1007/978-3-319-94463-0.

Akribix. (2006). Informationstheorie - Entropie; Zwei Quellen. X Sendequelle und Y Empfängerquelle information theorie - entropy [Zugegriffen 17.02.2021]. https://de.wikipedia.org/wiki/Transinformation#/media/Datei:Entroy_XY.png

Alpaydin, E. (2014). *Introduction to Machine Learning*. MIT Press.

AMD. (2017). *Processor Programming Reference (PPR) for AMD Family 17h Model 01h, Revision B1 Processors* [Zugegriffen 30.11.2018]. http://developer.amd.com/wordpress/media/2017/11/54945_PPR_Family_17h_Models_00h-0Fh.pdf

AMD Developer Tools Team. (2018). *AMD uProf User Guide* [Zugegriffen 30.11.2018]. https://developer.amd.com/wordpress/media/2013/12/User_Guide.pdf

an Mey, D., Dammer, A., Dietrich, R., Eitzinger, J., Filla, N., Hager, G., Hahnfeld, J., Kapinos, P., Röhl, T., Schürhoff, D., Wienke, S., Winkler, F., und Zeiser, T. (2019). *Process-Oriented Performance Engineering Service Infrastructure for Scientific Software at German HPC Centers* (Techn. Ber.) (Zugegriffen 17.04.2020). https://blogs.fau.de/prope/files/2019/05/ProPE.pdf

Arthur, D., und Vassilvitskii, S. (2007). K-Means++: The Advantages of Careful Seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*.

Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., Huber, H., Panda, R., Thomas, F., und Wilde, T. (2014). A Case Study of Energy Aware Scheduling on SuperMUC. *Lecture Notes in Computer Science*. DOI: 10.1007/978-3-319-07518-1_25.

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., und Weeratunga, S. (1991). The Nas Parallel Benchmarks. *The International Journal of Supercomputing Applications*, *5*(3). DOI: 10.1177/109434209100500306.

Battiti, R. (1994). Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, *5*(4). DOI: 10.1109/72.298224.

Benedict, S., Petkov, V., und Gerndt, M. (2010). PERISCOPE: An Online-Based Distributed Performance Analysis Tool. In M. S. Müller, M. M. Resch, A. Schulz und W. E. Nagel (Hrsg.), *Tools for High Performance Computing 2009* (S. 1–16). Springer Berlin Heidelberg.

Blum, A. L., und Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*(1). DOI: 10.1016/S0004-3702(97)00063-5.

Blumentritt, T., und Schmid, F. (2012). Mutual information as a measure of multivariate association: analytical properties and statistical estimation. *Journal of Statistical Computation and Simulation*, *82*(9). DOI: 10.1080/00949655.2011.575782.

Branch, M. A., Coleman, T. F., und Li, Y. (1999). A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. *SIAM Journal on Scientific Computing*, *21*(1). DOI: 10.1137/S1064827595289108.

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1). DOI: 10.1023/A:1010933404324.

Breiman, L., Friedman, J. H., Olshen, R. A., und Stone, C. J. (1993). *Classification And Regression Trees* ([Repr.]). Chapman & Hall.

Brown, G., Pocock, A., Zhao, M.-J., und Luján, M. (2012). Conditional Likelihood Maximisation: A Unifying Framework for Information Theoretic Feature Selection. *Journal of Machine Learning Research*, *13*(1). http://jmlr.org/papers/v13/brown12a.html

Bundesministerium für Wirtschaftund Energie. (2019). *Die Energie der Zukunft - Zweiter Fortschrittsbericht zur Energiewende* (Techn. Ber.). Bundesministerium für Wirtschaftund Energie (BMWi). https://www.bmwi.de/Redaktion/DE/Publikationen/Energie/fortschrittsbericht-monitoring-energiewende.html

Chadha, M., Ilsche, T., Bielert, M., und Nagel, W. E. (2017). A Statistical Approach to Power Estimation for x86 Processors. *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. DOI: 10.1109/IPDPSW.2017.98.

Chadha, M., und Gerndt, M. (2019). Modelling DVFS and UFS for Region-Based Energy Aware Tuning of HPC Applications. *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. DOI: 10.1109/IPDPS.2019.00089.

Chollet, F., et al. (2015). Keras [Zugegriffen 21.03.2022]. https://keras.io

Desrochers, S., Paradis, C., und Weaver, V. M. (2016). A Validation of DRAM RAPL Power Measurements. *Proceedings of the Second International Symposium on Memory Systems*. DOI: 10.1145/2989081.2989088.

Dongarra, J., London, K., Moore, S., Mucci, P., und Terpstra, D. Using PAPI for Hardware Performance Monitoring on Linux Systems. In: In *Conference on Linux Clusters: The HPC Revolution*. Linux Clusters Institute, 2001, Juni.

Dongarra, J., und Luszczek, P. (2011). Encyclopedia of Parallel Computing. In D. Padua (Hrsg.), *Encyclopedia of Parallel Computing* (S. 2055–2057). Springer US. DOI: 10.1007/978-0-387-09766-4_157.

Free Software Foundation, Inc. (2020). *Using the GNU Compiler Collection (GCC)* [Zugegriffen 29.07.2020]. https://gcc.gnu.org/onlinedocs/gcc/index.html

future thinking. (2014). Deutscher Rechenzentrumspreis [Zugegriffen 10.12.2021]. https://www.future-thinking.de/wp-content/uploads/2020/07/Deutscher-Rechenzentrumspreis-2014.pdf

Gerndt, M., César, E., und Benkner, S. (Hrsg.). (2015). *Automatic tuning of HPC applications : The periscope tuning framework*. Shaker Verlag. DOI: 10.2370/9783844035179.

Ghiselli, E. E. (1964). *Theory of psychological measurement*. McGraw, Hill.

Gocht, A., Bendifallah, Z., Mian, U. S., und Bouizi, O. (2016). *D1.1 Hardware and System-Software Tuning Plugins* (Techn. Ber.) (Zugegriffen 21.03.2022). READEX. https://www.readex.eu/wp-content/uploads/2018/05/D1_1.pdf

Gocht, A., Lehmann, C., und Schöne, R. (2018). A New Approach for Automated Feature Selection. *2018 IEEE International Conference on Big Data (Big Data)*. DOI: 10.1109/BigData.2018.8622548.

Gocht, A., Schöne, R., und Bielert, M. (2019). Q-Learning Inspired Self-Tuning for Energy Efficiency in HPC. *2019 International Conference on High Performance Computing Simulation (HPCS)*. DOI: 10.1109/HPCS48598.2019.9188112.

Gocht, A., Schöne, R., und Frenzel, J. (2021). Advanced Python Performance Monitoring with Score-P. *Tools for High Performance Computing 2018 / 2019*. DOI: 10.1007/978-3-030-66057-4_14.

Gottwald, A. (Hrsg.). (1993). *Nachrichtenübertragung 1: System- und Informationstheorie*. De Gruyter. DOI: 10.1515/9783486784107.

Guyon, I., Gunn, S. R., Ben-Hur, A., und Dror, G. (2003). NIPS 2003 workshop on feature extraction Challenge result analysis [Zugegriffen 21.03.2022]. http://clopinet.com/isabelle/Projects/NIPS2003/analysis.html

Guyon, I., Gunn, S. R., Ben-Hur, A., und Dror, G. (2004). Result Analysis of the NIPS 2003 Feature Selection Challenge [ACMID: 2976109]. *Advances in Neural Information Processing Systems*. http://dl.acm.org/citation.cfm?id=2976040.2976109

Hackenberg, D., Ilsche, T., Schöne, R., Molka, D., Schmidt, M., und Nagel, W. E. (2013). Power measurement techniques on standard compute nodes: A quantitative comparison. *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. DOI: 10.1109/ISPASS.2013.6557170.

Hackenberg, D., Schöne, R., Ilsche, T., Molka, D., Schuchart, J., und Geyer, R. (2015). An Energy Efficiency Feature Survey of the Intel Haswell Processor. *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. DOI: 10.1109/IPDPSW.2015.70.

Hall, M. A. (2000). Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, 359–366.

Hartung, J., Elpelt, B., und Klösener, K.-H. (2005). *Statistik Lehr- und Handbuch der angewandten Statistik* (14., unwesentlich veränd. Aufl.). Oldenbourg.

Hastie, T., Tibshirani, R., und Friedman, J. (2009). *The Elements of Statistical Learning* (Bd. Second Edition). Springer New York. DOI: 10.1007/978-0-387-84858-7.

Haykin, S. O. (2009). *Neural Networks and Learning Machines* (Third Edition). Pearson.

Hintemann, R. (2020). *Effizienzgewinne reichen nicht aus: Energiebedarf der Rechenzentren steigt weiter deutlich an* (Techn. Ber.) (Zugegriffen 21.03.2022). Borderstep Institut für Innovation und Nachhaltigkeit gemeinnützige GmbH. https://www.borderstep.de/wp-content/uploads/2020/03/Borderstep-Rechenzentren-2018-20200511.pdf

IEEE. (2019). IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 1–84. DOI: 10.1109/IEEESTD.2019.8766229.

Ilsche, T., Schöne, R., Bielert, M., Gocht, A., und Hackenberg, D. (2017). lo2s – Multi-core System and Application Performance Analysis for Linux. *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. DOI: 10.1109/CLUSTER.2017.116.

Ilsche, T. (2020). *Energy Measurements of High Performance Computing Systems: From Instrumentation to Analysis* [Dissertation, Technischen Fakultät Universität Dresden]. https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-716000

Ilsche, T., Hackenberg, D., Schone, R., Bielert, M., Hopfner, F., und Nagel, W. E. (2019). MetricQ: A Scalable Infrastructure for Processing High-Resolution Time Series Data. *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*. DOI: 10.1109/daac49578.2019.00007.

Ilsche, T., Schöne, R., Schuchart, J., Hackenberg, D., Simon, M., Georgiou, Y., und Nagel, W. E. (2018). Power Measurement Techniques for Energy-Efficient Computing: Reconciling Scalability, Resolution, and Accuracy. *SICS Software-Intensive Cyber-Physical Systems*. DOI: 10.1007/s00450-018-0392-9.

Ilsche, T., Schuchart, J., Schöne, R., und Hackenberg, D. (2015). Combining Instrumentation and Sampling for Trace-Based Application Performance Analysis. *Tools for High Performance Computing 2014*. DOI: 10.1007/978-3-319-16012-2_6.

Intel. (2015). *Intel® Xeon® Processor E5 and E7 v3 Family Uncore Performance Monitoring Reference Manual* (Techn. Ber.) (Zugegriffen 31.01.2020). Intel. https://software.intel.com/sites/default/files/managed/ea/25/331051-intel-xeon%20processor-e5-e7-v3-uncore.pdf

Intel. (2017). *Intel® Xeon® Processor E5 v3 Product Family Specification Update* (Techn. Ber.) (Zuigegriffen 21.03.2022). Intel. https://cdrdv2.intel.com/v1/dl/getcontent/330785

Intel. (2019a). *Intel® Math Kernel Library (Intel® MKL)*.

Intel. (2019b). *Intel® 64 and IA-32 Architectures Optimization Reference Manual* (Techn. Ber.) (Zugegriffen 21.02.2020). Intel. https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf

Intel. (2019c). *Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes:1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4* (Techn. Ber.) (Zugegriffen 02.01.2019). Intel. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

Intel. (2020). *Intel® C++ Compiler 19.1 Developer Guide and Reference* [Zugegriffen 29.07.2020]. https://software.intel.com/content/dam/develop/external/us/en/documents/19-1-cpp-compiler-devguide.pdf

Jolliffe, I. T. (2002). *Principal Component Analysis* (Second Edition). Springer, New York, NY. DOI: 10.1007/b98835.

Kapern, P., und von der Leyen, U. (2020). Von der Leyen setzt auf starke deutsche Präsidentschaft [Zugegriffen 30.09.2020]. *Deutschlandfunk - Interview der Woche*. https://www.deutschlandfunk.de/kommissionspraesidentin-zur-lage-der-eu-von-der-leyen-setzt.868.de.html?dram:article_id=479398

Kingma, D. P., und Ba, J. (2015). Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations*. http://arxiv.org/abs/1412.6980

Kjeldsberg, P. G., Gocht, A., Gerndt, M., Riha, L., Schuchart, J., und Mian, U. S. (2017). READEX Linking Two Ends of the Computing Continuum to Improve Energy-efficiency in Dynamic Applications. *Proceedings of the Conference on Design, Automation & Test in Europe*. DOI: 10.23919/DATE.2017.7926967.

Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S., und Nagel, W. E. (2008). The Vampir Performance Analysis Tool-Set. *Tools for High Performance Computing*. DOI: 10.1007/978-3-540-68564-7_9.

Knüpfer, A., Rössel, C., Mey, D. a., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., und Wolf, F. (2012). Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir. *Tools for High Performance Computing 2011*. DOI: 10.1007/978-3-642-31476-6_7.

Kumaraswamy, M., Chowdhury, A., Gocht, A., Zapletal, J., Diethelm, K., Riha, L., Sawley, M.-C., Gerndt, M., Reissmann, N., Vysocky, O., Bouizi, O., Kjeldsberg, P. G., Carreras, R., Schöne, R., Sabir Mian, U., Kannan, V., und Nagel, W. E. (2021). Saving Energy Using the READEX Methodology. *Tools for High Performance Computing 2018 / 2019*. DOI: 10.1007/978-3-030-66057-4_2.

Kunen, A. J., Bailey, T. S., und Brown, P. N. (2015). *KRIPKE - A Massively Parallel Transport Mini-App* (Techn. Ber.) (Zugegriffen 12.10.2020). Lawrence Livermore National Laboratory. https://computing.llnl.gov/projects/co-design/kripke_ans_2015_paper.pdf

Kunen, A. J., Brown, P. N., Bailey, T. S., und Maginot, P. G. (2020). KRIPKE [Zugegriffen am 12.10.2020]. https://github.com/LLNL/Kripke/

Larabel, M. (2020). Intel Uncore Frequency Driver On Linux Is Closer To Mainline With Latest Patches [Zugegriffen 06.10.2020]. *phoronix*. https://www.phoronix.com/scan.php?page=news_item&px=Intel-Uncore-Frequency-Driver

Lewis, D. D. (1992). Feature Selection and Feature Extraction for Text Categorization. *Proceedings of the Workshop on Speech and Natural Language*. DOI: 10.3115/1075527.1075574.

libMSR library and msr-safe kernel module. (2013). DOI: 10.11578/dc.20171025.1459.

Lieber, M., Grützun, V., Wolke, R., Müller, M. S., und Nagel, W. E. (2012). Highly Scalable Dynamic Load Balancing in the Atmospheric Modeling System COSMO-SPECS+FD4. *Applied Parallel and Scientific Computing*.

likwid. (2020). likwid 5.0.1 [Zugegriffen 08.10.2020]. https://github.com/RRZE-HPC/likwid

Ma, J. (2021). Variable Selection with Copula Entropy. *CHINESE JOURNAL OF APPLIED PROBABILITY AND STATISTICS*, *37*(4), 405.

Ma, J., und Sun, Z. (2011). Mutual information is copula entropy. *Tsinghua Science and Technology*, *16*(1). DOI: 10.1016/S1007-0214(11)70008-6.

Malhi, A., und Gao, R. X. (2004). PCA-based feature selection scheme for machine defect classification. *IEEE Transactions on Instrumentation and Measurement*, *53*(6). DOI: 10.1109/TIM.2004.834070.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... und Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org]. https://arxiv.org/abs/1603.04467

McCalpin, J. D. (1995). Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, 19–25.

Message Passing Interface Forum. (2015). *MPI: A Message-Passing Interface Standard* (3.1) [Zugegriffen 21.03.2022]. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

Miceli, R., Civario, G., Sikora, A., César, E., Gerndt, M., Haitof, H., Navarrete, C., Benkner, S., Sandrieser, M., Morin, L., und Bodin, F. (2013). AutoTune: A Plugin-Driven Approach to the Automatic Tuning of Parallel Applications. *Applied Parallel and Scientific Computing*. DOI: 10.1007/978-3-642-36803-5_24.

Molina, A., Schramowski, P., und Kersting, K. (2020). Padé Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks. *8th International Conference on Learning Representations*. https://openreview.net/pdf?id=BJlBSkHtDS

Molka, D. (2016). *Performance Analysis of Complex Shared Memory Systems* [Dissertation, Technischen Universität Dresden].

Molka, D., Schöne, R., Hackenberg, D., und Nagel, W. E. (2017). Detecting Memory-Boundedness with Hardware Performance Counters. *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. DOI: 10.1145/3030207.3030223.

Morales, J. L., und Nocedal, J. (2011). Remark on "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained Optimization". *ACM Trans. Math. Softw.*, *38*(1). DOI: 10.1145/2049662.2049669.

Müller, M. S., Baron, J., Brantley, W. C., Feng, H., Hackenberg, D., Henschel, R., Jost, G., Molka, D., Parrott, C., Robichaux, J., Shelepugin, P., van Waveren, M., Whitney, B., und Kumaran, K. (2012). SPEC OMP2012 — An Application Benchmark Suite for Parallel Systems Using OpenMP. *OpenMP in a Heterogeneous World: 8th International Workshop on OpenMP, IWOMP 2012, Rome, Italy, June 11-13, 2012. Proceedings*. DOI: 10.1007/978-3-642-30961-8_17.

Nelsen, R. B. (2006). *An introduction to copulas* (2. ed.). Springer. DOI: 10.1007/0-387-28678-0.

NERSC. (2018). DGEMM [Zugriff durch WaybackMachine möglich, Zugegriffen 21.03.2022]. http://web.archive.org/web/20180307220721/http://www.nersc.gov:80/research-and-development/apex/apex-benchmarks/dgemm

Nvidia. (2020). *NVML API Reference Guide* [Zugegriffen 13.10.2020]. https://docs.nvidia.com/deploy/nvml-api/group___nvmlDeviceCommands.html/#group___nvmlDeviceCommands_1gc2a9a8db6fffb2604d27fd67e8d5d87f

OpenMP Architecture Review Board. (2018). *OpenMP Application Programming Interface* (5.0) [Zugegriffen 21.03.2022]. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

Peng, H., Long, F., und Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8). DOI: 10.1109/TPAMI.2005.159.

Rinne, H. (2003). *Taschenbuch der Statistik* (3., vollst. überarb. und erw. Aufl.). Verlag Harri Deutsch. http://slubdd.de/katalog?TN_libero_mab2)1000012238

Rountree, B., Lownenthal, D. K., de Supinski, B. R., Schulz, M., Freeh, V. W., und Bletsch, T. (2009). Adagio: Making DVS Practical for Complex HPC Applications. *Proceedings of the 23rd International Conference on Supercomputing*. DOI: 10.1145/1542275.1542340.

Schöne, R., Ilsche, T., Bielert, M., Gocht, A., und Hackenberg, D. (2019). Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance. *2019 International Conference on High Performance Computing Simulation (HPCS)*. DOI: 10.1109/HPCS48598.2019.9188239.

Schöne, R. (2017). *A Unified Infrastructure for Monitoring and Tuning the Energy Efficiency of HPC Applications* [Dissertation, Technischen Universität Dresden]. https://d-nb.info/1144299985/34

Schöne, R. (2020). libfreqgen [Zugegriffen am 06.10.2020]. https://github.com/readex-eu/libfreqgen

Schöne, R., Ilsche, T., Bielert, M., Velten, M., Schmidl, M., und Hackenberg, D. (2021). Energy Efficiency Aspects of the AMD Zen 2 Architecture. *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. DOI: 10.1109/Cluster48925.2021.00087.

Schöne, R., und Molka, D. (2014). Integrating performance analysis and energy efficiency optimizations in a unified environment. *Computer Science - Research and Development*, *29*(3). DOI: 10.1007/s00450-013-0243-7.

Schöne, R., Nagel, W., und Pflüger, S. (2008). Analyzing Cache Bandwidth on the Intel Core 2 Architecture. *Parallel Computing: Architectures, Algorithms and Applications*, *15*, 365–372. https://www.parco.org/ParCo2007/parco2007.html

Schöne, R., Tschüter, R., Ilsche, T., Schuchart, J., Hackenberg, D., und Nagel, W. E. (2017). Extending the Functionality of Score-P Through Plugins: Interfaces and

Use Cases. *Tools for High Performance Computing 2016*. DOI: 10.1007/978-3-319-56702-0_4.

Schuchart, J., Gerndt, M., Kjeldsberg, P. G., Lysaght, M., Horák, D., Říha, L., Gocht, A., Sourouri, M., Kumaraswamy, M., Chowdhury, A., Jahre, M., Diethelm, K., Bouizi, O., Mian, U. S., Kružík, J., Sojka, R., Beseda, M., Kannan, V., Bendifallah, Z., ... und Nagel, W. E. (2017). The READEX formalism for automatic tuning for energy efficiency. *Computing*. DOI: 10.1007/s00607-016-0532-7.

Schweizer, B., und Wolff, E. F. (1981). On Nonparametric Measures of Dependence for Random Variables. *The Annals of Statistics*, *9*(4). DOI: 10.1214/aos/1176345528.

SciPy. (2020). *scipy.optimize.least_squares* [Zugegriffen 06.03.2020]. SciPy. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html

Seth, S., und Príncipe, J. C. (2010). Variable Selection: A Statistical Dependence Perspective. *2010 Ninth International Conference on Machine Learning and Applications*. DOI: 10.1109/ICMLA.2010.148.

Shende, S. S., und Malony, A. D. (2006). The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications*, *20*(2). DOI: 10.1177/1094342006064482.

Song, F., Guo, Z., und Mei, D. (2010). Feature Selection Using Principal Component Analysis. *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*, *1*. DOI: 10.1109/ICSEM.2010.14.

Spiliopoulos, V., Kaxiras, S., und Keramidas, G. (2011). Green governors: A framework for Continuously Adaptive DVFS. *Green Computing Conference and Workshops (IGCC), 2011 International*. DOI: 10.1109/IGCC.2011.6008552.

Tang, Z., Wang, Y., Wang, Q., und Chu, X. (2019). The Impact of GPU DVFS on the Energy and Performance of Deep Learning: An Empirical Study. *Proceedings of the Tenth ACM International Conference on Future Energy Systems*. DOI: 10.1145/3307772.3328315.

Technische Universität Dresden. (2020). Taurus [Zugegriffen 21.07.2020]. https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareTaurus

The Clang Team. (2020). *Clang 12 documentation - Clang command line argument reference* [Zugegriffen 29.07.2020].

The kernel development community. (2020a). *Perf Events and tool security* [Zugegriffen 18.08.2020]. https://www.kernel.org/doc/html/v5.2/admin-guide/perf-security.html

The kernel development community. (2020b). *The Linux kernel user's and administrator's guide - Power Management* [Zugegriffen 27.07.2020]. https://www.kernel.org/doc/html/latest/admin-guide/pm/index.html

Timme, N., Alford, W., Flecker, B., und Beggs, J. M. (2014). Synergy, redundancy, and multivariate information measures: an experimentalist's perspective. *Journal of Computational Neuroscience*, *36*(2). DOI: 10.1007/s10827-013-0458-4.

Treibig, J., Hager, G., und Wellein, G. (2010). LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. *2010 39th International Conference on Parallel Processing Workshops*. DOI: 10.1109/ICPPW.2010.38.

Trommler, P. (2016). *Dynamische Anpassung Compiler-basierter Instrumentierung unter Nutzung von Laufzeitstatistiken* (Bachelorarbeit) [, Technische Universität Dresden – Professur für Rechnerarchitektur].

Wagner, M., Llort, G., Mercadal, E., Gimenez, J., und Labarta, J. (2017). Performance Analysis of Parallel Python Applications. *Procedia Computer Science*, *108*. DOI: 10.1016/j.procs.2017.05.203.

Walker, M. J., Diestelhorst, S., Hansson, A., Das, A. K., Yang, S., Al-Hashimi, B. M., und Merrett, G. V. (2017). Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *36*(1). DOI: 10.1109/TCAD.2016.2562920.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* [Dissertation, University of Cambridge].

Weber, M., Ziegenbalg, J., und Wesarg, B. (2019). Online Performance Analysis with the Vampir Tool Set. *Tools for High Performance Computing 2017*. DOI: 10.1007/978-3-030-11987-4_8.

Weiser, M., Welch, B., Demers, A., und Shenker, S. (1994). Scheduling for Reduced CPU Energy. In *Mobile Computing* (Bd. 353). Springer US. DOI: 10.1007/978-0-585-29603-6_17.

Weste, N. H. E., und Harris, D. M. (2011). *CMOS VLSI design a circuits and systems perspective* (4. ed.). Addison-Wesley.

Whitney, B. (2012). *SPEC OMP2012 Documentation - Benchmark Documentation* [Zugegriffen 16.4.2020]. https://www.spec.org/omp2012/Docs/

Wolff, E. F. (1977). *Measures of dependence derived from copulas* [Dissertation, University of Massachusetts].

Wolff, E. F. (1980). N-dimensional measures of dependence. *Stochastica*, *4*(3), 175–188. http://eudml.org/doc/38838

Yang, H. H., und Moody, J. (1999). Data Visualization and Feature Selection: New Algorithms for Nongaussian Data. *Advances in Neural Information Processing Systems*, *12*, 687–693.

Zeng, X., und Durrani, T. S. (2011). Estimation of mutual information using copula density function. *Electronics Letters*, *47*(8). DOI: 10.1049/el.2011.0778.

Zhu, C., Byrd, R., und Nocedal, J. (2020). *scipy.optimize.fmin_l_bfgs_b* [Zugegriffen 06.03.2020]. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

Zhu, C., Byrd, R. H., Lu, P., und Nocedal, J. (1997). Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization. *ACM Trans. Math. Softw.*, *23*(4). DOI: 10.1145/279232.279236.

Abbildungsverzeichnis

1.1 Klassischer Ablauf einer Energieoptimierung eines Programmes. 8
1.2 Ablauf der Optimierung mit dem in dieser Arbeit entworfenen Framework. . 9

2.1 Vereinfachter Aufbau von READEX. 20
2.2 Darstellung der Entropie $H(X_k)$ und $H(Y)$ und der Transinformation $I(X, Y)$ als Informationskanal und als Venn-Diagramm. 26
2.3 Venn-Diagramm für die Transinformation unter Berücksichtigung von zwei Merkmalen X_k, X_j und der Zielgröße Y. 28
2.4 Vereinfachte Darstellung eines neuronalen Netzes mit zwei versteckten Layern. 34
2.5 Signallauf für ein einzelnes Neuron. 34

3.1 Indikatorfunktion im Vergleich mit verschiedenen logistischen Funktionen. . . 41
3.2 Perfekt linear voneinander abhängige Wertepaare. 43
3.3 Wertepaare aus unterschiedlichen monotonen Funktionen. 44
3.4 Simulationen des Zusammenhangs zweier zufällig generierter Variablen. . . . 46
3.5 Vergleich verschiedener Merkmalsauswahlverfahren ohne wiederholte Merkmale. 49
3.6 Vergleich verschiedener Merkmalsauswahlverfahren mit wiederholten Merkmalen. 50

4.1 Prinzipieller Aufbau der RRL, mit den Anpassungen für diese Arbeit. 54
4.2 Entscheidungsprozess, ob ein Knoten im Call-Tree der RRL optimiert werden soll. 56
4.3 Schematischer Aufbau der verwendeten Intel-Haswell-Prozessoren. 59
4.4 Schematischer Aufbau eines Kerns der verwendeten Intel-Haswell-Prozessoren. 60
4.5 Speicherbandbreite der vier modifizierten Stream-Kernel. 63
4.6 Anzahl von Regionen der vier Stream-Kernel `copy`, `scale`, `add` und `triad` mit der angegebenen optimalen Frequenz abhängig davon, in welche Speicherebene die verwendeten Daten des Kernels passen. 64
4.7 Verschiedene optimale Frequenzen der unterschiedlichen Benchmarks. 68
4.8 Vergleich der gemessenen Daten und des regionsspezifischen Energiemodells. . 71
4.9 Vergleich zweier auf 2,0 GHz und 1,5 GHz Core- respektive Uncorefrequenz normierter Regionen 72

5.1 Gemessene Speicherbandbreite der vier Stream-Kernel. 82
5.2 Beispieleventrate von `hswep_unc_cbo::UNC_C_TOR_OCCUPANCY:OPC_RFO` über die Datenmenge und die unterschiedlichen Kernel. 83
5.3 Verlauf der durch $J_{HJMI,CD}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 84
5.4 Verlauf der durch $J_{HJMI,CD,\text{NormMax}}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 86
5.5 Verlauf der durch $J_{MRMR,CD}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 86
5.6 Verlauf der durch $J_{MRMR,CD,\text{NormGauss}}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 88
5.7 Verlauf der durch $J_{MRMR,\gamma}$ ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 88

5.8 Verlauf der mithilfe von Random Forest ausgewählten Merkmale über die reale Größe der in einer Stream-Region verwendeten Arrays. 89
5.9 Median des Energiemehrverbrauchs bezogen auf die maximal möglichen Energieeinsparungen für Stream. 90
5.10 Vergleich eines guten und eines schlechten Trainingsergebnisses. 91
5.11 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `copy` und `scale` sowie Modell-Chadha. 92
5.12 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `add` und `triad` sowie Modell-Chadha. 93
5.13 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `copy` und `scale` sowie Modell-E. 94
5.14 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `add` und `triad` sowie Modell-E. 95
5.15 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `copy` und `scale` sowie Modell-E-Funktion. 96
5.16 Ergebnisse der Kreuzvalidierung für die Stream-Kernel `add` und `triad` sowie Modell-E-Funktion. 97
5.17 Median des Energiemehrverbrauchs bezogen auf die maximal möglichen Energieeinsparungen für verschiedene Benchmarks. 101
5.18 Auswahl der durch Modell-E generierten normierten Energiemodelle und eines zu erwartenden Energiemodells für verschiedene Merkmale, mit denen Modell-E trainiert wurde 102
5.19 Ergebnisse der Kreuzvalidierung über verschiedene Benchmarks zusammen mit dem Modell-Chadha. 104
5.20 Ergebnisse der Kreuzvalidierung über verschiedene Benchmarks zusammen mit dem Modell-E. 105
5.21 Ergebnisse der Kreuzvalidierung über verschiedene Benchmarks zusammen mit dem Modell-E-Funktion. 106
5.22 Mittelwert der verschiedenen Messungen für Energieverbrauch und Laufzeit von Kripke zusammen mit den minimal und maximal gemessenen Werten. . 108
5.23 Vergleich eines Laufs von Kripke mit RRL und ohne RRL. 109

Tabellenverzeichnis

3.1 Vergleich verschiedener Maßzahlen für eine perfekt lineare Abhängigkeit: $\mathbf{x} = \mathbf{y}$. 44
3.2 Vergleich verschiedener Maßzahlen für Abhängigkeit von Wertepaaren generiert aus unterschiedlichen monotonen Funktionen. 45

4.1 Unterschiedliche mögliche Zustände eines Knotens. 56
4.2 Anzahl der verfügbaren Performance-Counter pro PMON-Block. 61
4.3 Gesamter Speicherbedarf für die drei in der Matrixmultiplikation verwendeten Matrizen. 66
4.4 Optimale Frequenzen der unterschiedlichen Matrix-Multiplikationen sowie Anzahl der Wiederholungen der eigentlichen Matrix-Multiplikation. 67
4.5 Gesamter Speicherbedarf für die drei in der Matrixmultiplikation verwendeten Matrizen. 67
4.6 Als relevant für die Optimierung des Energieverbrauchs ermittelte Performance-Events von Chadha und Gerndt (2019). 76
4.7 Performance-Events zur Bestimmung verschiedener Flaschenhälse nach Molka et al. (2017). 77
4.8 Performance-Events zur Erkennung ineffizienter Anwendung nach an Mey et al. (2019). 77
4.9 Modell-Chadha: Aufbau des neuronales Netzes nach Chadha und Gerndt (2019). 79
4.10 Modell-E: Aufbau des neuronales Netzes mit verschiedenen Nichtlinearitäten. 79
4.11 Modell-E-Funktion: Aufbau des neuronalen Netzes unter Verwendung von Gleichung 4.2, S. 70. 79

5.1 Mittels $J_{HJMI,CD}$ ausgewählte Merkmale. 99
5.2 Mittels $J_{HJMI,CD,\text{NormMax}}$ ausgewählte Merkmale. 99
5.3 Mittels $J_{MRMR,CD}$ ausgewählte Merkmale. 99
5.4 Mittels $J_{MRMR,CD,\text{NormGauss}}$ ausgewählte Merkmale. 99
5.5 Durch Variable Selection $J_{MRMR,\gamma}$ ausgewählte Merkmale. 100
5.6 Durch Random Forest ausgewählte Merkmale. 100
5.7 Energie- und Laufzeitunterschied von Kripke für zwei Optimierungsansätze im Vergleich zu 2,5 GHz Core- und 3,0 GHz Uncorefrequenz. 107
5.8 Energie- und Laufzeitunterschied von Kripke für zwei Tuningansätze, im Vergleich zu einer freien Corefrequenz (auch *Turbo* genannt) und einer freien Uncorefrequenz. 107

Quelltextverzeichnis

2.1 Beispielcode zur Verwendung von Call-Paths zur Identifikation von Regionen. 21

4.1 Originale Version des Stream-Copy-Kernels. 63
4.2 Für diese Arbeit modifizierte Version des Stream-Copy-Kernels. 63
4.3 Matrixmultiplikation nach NERSC (2018). 66
4.4 Algorithmus zur Berechnung der zweidimensionalen Copula-Dichte. 74
4.5 Optimierter Algorithmus zur Berechnung der zweidimensionalen Copula-Dichte. 75

5.1 Konfiguration der verwendeten Mini-App Kripke. 107

Danksagung

Am Ende dieser Dissertation steht der Dank bei den Menschen, die diese Arbeit ermöglicht haben. Natürlich kann diese Liste nicht vollumfänglich sein, da speziell die vielen Kolleginnen und Kollegen am ZIH in den verschiedensten Facetten ihrer Arbeit die Grundlagen gelegt haben, auf der diese Dissertation aufbaut. Dennoch möchte ich einigen Menschen besonders danken.

Als Direktor des ZIH und Betreuer dieser Arbeit möchte ich zuerst Prof. Nagel danken, der mir die Möglichkeit gegeben hat, zu diesem Thema zu promovieren und die Promotion immer unterstützt hat. Als Nächstes muss ich Christoph Lehmann danken, der mir mit regelmäßigen Diskussionen geholfen hat, eine Idee von Statistik zu bekommen und der mich bis zum Schluss auf Ungenauigkeiten hingewiesen hat. Ohne die Kollegen meiner Arbeitsgruppe, im speziellen Robert Schöne, Mario Bielert, Thomas Ilsche, Daniel Molka und Daniel Hackenberg, wäre diese Arbeit aber ebenfalls nicht möglich gewesen. Ich möchte ihnen für die Einblicke in die Rechnerarchitektur und Performanceanalyse danken, die sie mir gewährt haben, sowie für die unermüdlichen Diskussionen zu C, C++ und Python. Natürlich darf an dieser Stelle auch mein Zweitgutachter Krisitan Kersting nicht fehlen, der mir neue Einblicke in die künstliche Intelligenz gewährt hat.

Zu guter Letzt möchte ich meiner Familie danken. Besonders möchte ich mich dabei bei meinen Eltern Birgit und Burkhard bedanken, die mich lebenslang unterstützt haben. Auch bei meiner Frau Johanna möchte ich mich bedanken, da ich ohne sie nicht da wäre, wo ich jetzt bin.

Danke.