Technische Universität München
TUM School of Computation, Information and Technology

Accurate and Lightweight Run-time Power Estimation and Power Forecasting Models for Multi-core Processors

Mark Balazs Hilary Sagi

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr. Michael Gerndt

Prüfende der Dissertation:

1. Prof. Dr. sc.techn. Andreas Herkersdorf
2. Prof. Dr.-Ing. Jörg Henkel

Die Dissertation wurde am 09.02.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 01.08.2024 angenommen.

Abstract

The compute performance of today's multi-core processors are limited due to power and thermal constraints. To improve performance while adhering to these constraints, many different reactive and proactive power and thermal management algorithms have been introduced. These management algorithms rely on accurate and fine-grained — both in the time domain, e.g. μs to ms, and space domain, e.g. *core*-level — run-time estimations of dynamic power consumption. Note, that in the context of this thesis a *core* refers to the ALU, BPU etc. as well as private L1 and L2 caches. The dynamic power consumption is directly related to the switching activity of the underlying circuits. In the absence of fine-grained power sensors for direct power measurement, the power consumption of a core can be indirectly estimated by using performance counters as surrogate run-time activity indicators. These performance counters are then used as input to a power model generated at design-time. However, both the numbers of available performance counters and their scope, in regard to how much activity is directly observable through them, are limited. These limitations in observability lead to inaccuracies in the run-time power estimation if the underlying power model does not account for sources of modeling error. Such sources are for example the multicollinearity of the performance counter inputs as well as nonlinear relations between specific performance counters and dynamic power consumption. In this thesis, multiple statistical methods are proposed to increase the estimation accuracy of run-time power models. For one, independent component analysis is proposed to minimize input multicollinearity and the use of FFNNs as well as non-linear input transformations are proposed to account for non-linear performance counter / power relations. All methodologies are optimized for low-complexity, i.e. kept lightweight, to assure that the run-time overhead of the final power models is almost negligible compared to the processor cores for which power estimations are generated. In addition to these power modeling methodologies, a power forecasting methodology based on LSTM recurrent neural networks is proposed with the goal of forecasting future dynamic power consumption. The accuracy of the proposed power modeling and power forecasting methodologies are evaluated on the state-of-the-art Sniper multi-core simulator using McPAT for simulating fine-grained power consumption. The evaluations of the proposed power modeling approaches show a decrease of 3.0% - 7.5% in relative RMSE compared to linear power modeling approaches. The LSTM-based power forecasting methodology shows 43%, 38% lower worst-case phase change error compared to AR-based reference power forecasting techniques for forecast time-intervals of 1 ms and 10 ms, respectively. In conclusion, the proposed higher accuracy power models enable better decision taking of reactive power and thermal management techniques and the proposed power forecasting model with high phase change accuracy enables specific proactive power and thermal management techniques.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Dr. Andreas Herkersdorf. His tremendous support and academic guidance as well as his patience were instrumental for making my research possible. I am infinitely grateful for his invaluable advice on our scientific endeavors but also on my personal growth and his continuous encouragement. He taught me how to overcome scientific dead-ends, how to effectively collaborate in complex research projects and how to communicate my ideas.

I would like to thank Dr. Thomas Wild for the countless project related and scientific discussions which channeled my work and also thank him for his expert feedback on our publications.

I am very grateful for Dr. Anh Vu Doan and the many research ideas we developed together, his in-depth knowledge and support for developing the mathematical models underpinning this thesis. I also want to thank him for the strong motivation to explore and follow through with the research ideas from their first rough formulation to their final, well-polished publication.

I also want to thank all my colleagues at the Chair of Integrated Systems at TUM for the countless, constructive scientific discussions, the great team spirit to accomplish our goals and the solidarity when carrying out course examinations, especially "DT Korrektur" which would have been insurmountable for any one individual alone. In this regard, I also want to highlight and thank Nael Fasfous for his contributions with the optimized hardware implementations of the neural network models proposed in this work.

I would like to acknowledge the intensive collaboration with the Chair of Embedded Systems at KIT, and would thank personally Martin Rapp, Dr. Heba Khdr and Prof. Dr. Jörg Henkel for their scientific contributions towards developing ideas, evaluating research results and the joint drafting of publications. For me, these collaborations were not only very fruitful but also highly instructive in regard to methodologies and data analysis. Also, I want to thank all other colleagues from the DFG "Invasive Computing" SFB/Transregio with whom I had the pleasure to work together on demonstrators and discuss and learn about a wide range of scientific problems. [1]

I want to acknowledge and thank the many students I had the privilege to supervise who contributed with their curiosity and with initial explorations as well as implementations of some of the research ideas.

Finally, I would like to thank my friends and family who supported me the entire time.

[1] This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — Projektnummer 146371743 — TRR 89 "Invasive Computing".

Contents

Abstract iii

Acknowledgments v

Contents vii

List of Figures ix

List of Tables xi

Acronyms xiii

1 Introduction 1

2 Related Work 7
2.1 Background on Power, Sensors and Managment Algorithms 8
2.1.1 Circuit-level and Microarchitectural-level Power Consumption . . . 8
2.1.2 On-chip Sensor Capabilities 10
2.1.3 Power and Thermal Management Algorithms 12
2.1.4 Applications for Run-time Power Forecasts 13
2.2 Run-time Power Estimation 15
2.2.1 Linear Power Models 15
2.2.2 Nonlinear Power Models 24
2.2.3 Neural Network-based Power Models 26
2.2.4 Discussion of Methods on Minimizing Multicollinearity 28
2.2.5 Discussion of Linear vs. Nonlinear Modeling Methodologies 29
2.3 Forecasting and Prediction of workload-dependent Processor States 30
2.3.1 Forecasting Models 32
2.3.2 Prediction Models 34
2.3.3 Power Forecasting Models 36

3 Experimental Setup for Power Model Evaluation 39
3.1 Overview Experimental System and Workflow 39
3.2 (Hot)Sniper and Performance Simulation 42
3.3 McPAT and Power Simulation 43
3.4 PARSEC and SPLASH-II Benchmark Suites as Workloads 46

4 Novel and Lightweight Power Estimation Models **49**
4.1 Independent Component Analysis-based Power Model 49
4.1.1 Independent Component Analysis 50
4.1.2 ICA-based Power Model Generation 52
4.1.3 Estimating Core-level Dynamic Power 54
4.1.4 Experimental Evaluation 54
4.2 Feedforward Neural Network-based Power Model 61
4.2.1 FFNN Architectures and Hyperparameter Solution Space 62
4.2.2 Single-Objective Neural Architecture Hyperparameter Optimization 66
4.2.3 Multi-Objective Neural Architecture Hyperparameter Optimization 67
4.2.4 Experimental Evaluation 72
4.3 Nonlinear Transformation-based Power Model 82
4.3.1 Power Model Generation and Lightweight Run-time Usage 83
4.3.2 Experimental Evaluation 89
4.3.3 Discussion of Nonlinear Performance Counter/Power Relationship 102
4.4 Comparison and Discussion of the Power Estimation Methodologies . . . 106

5 Novel and Accurate Power Forecasting Model based on LSTM RNNs **109**
5.1 Methodology for Generating LSTM RNN Power Model 111
5.2 Experimental Evaluation 116

6 Conclusion and Outlook **123**

Bibliography **125**

List of Figures

1.1 Motivational example for power modeling and power forecasting: Splash-2 FFT smaller power phase changes (blue area) require accurate run-time estimations to allow optimal reactive power management decisions. At around 340ms (red area), the rapid power phase change leads to a violation of the thermal constraint (red line). Anticipating such thermal violations by the means of accurate power forecasts would allow to proactively avoid the violation. 2

3.1 Experimental setup workflow with a simulated 16-core processor using HotSniper and Multicore Power, Area, and Timing (McPAT) for generating performance counter and power consumption data 40

4.1 Example of scattered data analyzed via Principal Component Analysis (PCA) on the left and Independent Component Analysis (ICA) on the right, showing the resulting two principal components $\hat{x}_1$, $\hat{x}_2$ and the resulting two independent components s_1, s_2 51
4.2 Methodology to generate a ICA-based power model for run-time power estimations [1] 53
4.3 Benchmark usage for obtaining performance counter and power data for ICA-based and reference power model generation [1] 56
4.4 Root-Mean-Square Error (RMSE) of ICA-based power models with increasing number of training workloads compared to a reference-based power model using synthetic workloads, Figure based on [1] 58
4.5 Overview of the ANN architectures investigated; the blue shaded rectangles indicate the ability to parameterize the number of hidden neurons per layer, i.e. from at least one hidden neuron per layer up to the given maximum number [2] 64
4.6 Flowchart of model hyperparametrization using a first 10-fold cross validation, final Feedforward Neural Network (FFNN) generation using a second 10-fold cross validation and the final power estimating FFNN estimation accuracy assessment on holdout data [2] 66
4.7 Flowchart of the multi-objective FFNN optimization of the hyperparameters using Non-dominated Sorting Genetic Algorithm-II (NSGA-II), 10-fold cross validations and a final FFNN power estimation accuracy assessment on the holdout data [3] 69
4.8 Average population performance over 50 generations as well as lowest overhead and RMSE values of the Pareto optimal solutions [3] 74

4.9 Power modeling and estimation flow (contributions of this work denoted with solid rectangles) [4] . . . 84
4.10 Run-time power estimation accounting for nonlinear performance counter / dynamic power relationship [4] . . . 86
4.11 Multivariate polynomial modeling approach with variable degrees per input performance counters [4] . . . 88
4.12 Comparison of Akaike Information Criterion (AIC) values for linear model and nonlinear transformation model with increasing number of performance counter inputs [4] . . . 92
4.13 Relative error values for power models generated with unknown core power for each Princeton Application Repository for Shared-Memory Computers (PARSEC) (P) and Stanford Parallel Applications for Shared-Memory (SPLASH-2) (S) benchmarks with benchmark names abbreviated with first three letters except SPLASH-2 radiosity (S-rao) and radix (S-rax) [4] . . . 98
4.14 Overview of the core microarchitecture composed of front end, execution engine as well as memory subsystem including the private L1 and L2 caches as well as the core's connection with the processor uncore [4] . . . 103

5.1 Overview of the proposed methodology to create and optimize the Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) at design-time and employ it at run-time [5] . . . 112
5.2 Selection of the final LSTM RNN neural hyperparameter architecture in a two-stage Design Space Exploration (DSE). First the depth of the RNN (number of layers) l_{opt} is selected under the simplifications of having the same number of neurons per layer (in blue); then the number of neurons for each layer n_i is optimized for the selected l_{opt} (in green) [5]. . . . 114
5.3 Power forecasts over time-frames of $\tau = 1\,\text{ms}$ by reference Autoregressive (AR) model [6] and the proposed LSTM RNN for the unseen PARSEC *facesim* benchmark [5]. . . . 119
5.4 Cross-correlation of forecast and actual power trace demonstrating that the LSTM RNN model actually forecasts future power [5]. An optimal oracle shows a peak at time delay 0. The power forecasting LSTM RNN closely matches this, demonstrating that this model actually forecasts future power consumption. In contrast, the AR model shows a peak at time delay -1, indicating that the AR model mostly follows the measured power with a delay of one sample. . . . 120

List of Tables

2.1 Overview and classification of related works on run-time power estimation 16
2.2 Overview and classification of related works on forecasting and prediction of power, thermal and performance characteristics 31

3.1 Cache architecture of the 16-core processor 39
3.2 Performance counters which are periodically traced from the simulated 16-core processor 44
3.3 Benchmarks used as generic workloads in this thesis 46
3.4 Benchmark distribution on the combined training/validation data set and on the holdout data set for evaluation of the power Neural Network (NN)-based estimation and forecasting methodologies in Section 4.2.4 and Section 5.2, respectively 47

4.1 Belsley collinearity evaluation results on performance counter data obtained through synthetic and generic workloads [1] 55
4.2 RMSE results for power models using ICA-transformation and for a reference-based power model using synthetic workloads [1] 57
4.3 Example of the encoding of individual solutions within a population including the objective performance metrics RMSE and run-time overhead and the NSGA-II population-relative performance metrics [3] 68
4.4 FFNN architectures found by the single-objective optimization methodology with average RMSE on the randomized validation data [2] 73
4.5 FFNN architectures found by the multi-objective optimization methodology with average RMSE on the randomized validation data [3] 75
4.6 Necessary computations and memory for a single power estimation, i.e. FFNN power model inference [2, 3] 76
4.7 FPGA resource usage, latency and maximum inference rates for an accelerator implementation [3] 78
4.8 Estimation accuracy of FFNNs, linear model, and polynomial model on the holdout data set [2, 3] 79
4.9 Set of nonlinear transformations investigated in this section [4] 85
4.10 Transformations maximizing linear correlation of specific performance counters with dynamic power [4] 90
4.11 Correlation coefficients for each performance counter after applying the nonlinear transformation [4] 91
4.12 Selection order of the performance counters by the Least-Angle Regression (LARS) algorithm [4] 92

4.13 Power Estimation error as absolute RMSE and relative RMSE for the four power modeling approaches [4] 95
4.14 Relative power estimation error for power models with reduced performance counter inputs [4] 97
4.15 Correlation coefficients after applying the nonlinear transformation at different frequencies and delta to the correlation coefficients of the regular performance counters [4] 99
4.16 Relative RMSE at different operating frequencies and unknown core power for power estimation model generation [4] 100
4.17 Computational and memory overhead for a single power estimation [4] . . 101
4.18 caption 103
4.19 Average RMSE estimation errors of different run-time dynamic power models and their run-time computational overhead per model inference, e.g. for 10 kHz estimation rates every 0.1 ms, as well as the total memory to be stored on the processor for each model 107

5.1 Forecasting Mean Absolute Percentage Error (MAPE) values and instantaneous worst case-error values for the LSTM RNN approach and reference-based approaches [5] 118
5.2 Forecasting error for the LSTM RNNs trained at 3 GHz operating frequency and and used to forecast power at operating frequencies of 2 GHz and 1 GHz [5] 120

Acronyms

AIC	Akaike Information Criterion.
APM	Application Power Management.
AR	Autoregressive.
ARMA	Autoregressive–Moving-Average.
BBV	Basic Block Vector.
BPU	Branch Predictor Unit.
CMOS	Complementary Metal-Sxide-Semiconductor.
CPI	Cycles Per Instruction.
CPU	Central Processing Unit.
DRAM	Dynamic Random Access Memory.
DSE	Design Space Exploration.
DVFS	Dynamic Voltage and Frequency Scaling.
EDP	Energy Delay Product.
FFNN	Feedforward Neural Network.
FinFET	Fin Field-Effect Transistor.
FIVR	Fully Integrated Voltage Regulator.
GMM	Gaussian Mixture Model.
GPU	Graphics Processing Unit.
ICA	Independent Component Analysis.
IPC	Instructions Per Cycle.
ISA	Instruction Set Architecture.
LARS	Least-Angle Regression.
LASSO	Least Absolute Shrinkage and Selection Operator.
LLC	Last Level Cache.
LSTM	Long Short-Term Memory.
MAC	Multiply-Accumulate.
MAPE	Mean Absolute Percentage Error.

McPAT	Multicore Power, Area, and Timing.
MMU	Memory Management Unit.
MOB	Memory Order Buffer.
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor.
MPC	Model Predictive Control.
MSE	Mean-Square Error.
NAG	Nesterov Accelerated Gradient.
NIC	Network Interface Card.
NN	Neural Network.
NoC	Network-on-Chip.
NSGA-II	Non-dominated Sorting Genetic Algorithm-II.
OLS	Ordinary Least Squares.
OLS	Register-Transfer Level.
PARSEC	Princeton Application Repository for Shared-Memory Computers.
PCA	Principal Component Analysis.
PCI	Peripheral Component Interconnect.
PCIe	Peripheral Component Interconnect Express.
PSU	Power Supply Unit.
RAPL	Running Average Power Limit.
RF	Random Forest.
RL	Reinforcement Learning.
RLS	Recursive Least Squares.
RMSE	Root-Mean-Square Error.
RNN	Recurrent Neural Network.
SBX	Simulated Binary Crossover.
SGD	Stochastic Gradient Descent.
SPEC	Standard Performance Evaluation Corporation.
SPLASH-2	Stanford Parallel Applications for Shared-Memory.
SVM	Support Vector Machine.
TDP	Thermal Design Power.
TLB	Translation Lookaside Buffer.
Vf	Voltage/frequency.
VIF	Variance Inflation Factor.
WC	Worst Case.

1 Introduction

More than half a century ago, *Moore's law* [7] first described the trend of exponentially increasing Metal–Oxide–Semiconductor Field-Effect Transistor (MOSFET) integration density per integrated circuit. In tandem with increasing integration density, the transistor costs of integrated circuits decreased and — by extension — the compute capabilities of processors also increased exponentially. Moore's observation was followed nine years later by the observation on so-called *Dennard scaling* [8] for MOSFETs which showed that down-scaling of transistor sizes keeps the power density of transistors constant through via continuing decreases in the supply voltage. With constant power densities, the higher transistor densities on newer semiconductor *technology nodes* allowed for processor designs with increasing complexity while keeping chip area and power consumption manageable.

For decades, processor designers aiming for ever higher compute capabilities used the availability of these ever higher transistor counts to increase both the Instructions Per Cycle (IPC) as well as the processor's operating frequencies leading to higher complexity of the processor architectures, e.g. highly pipelined, wide-issue and out-of-order designs. Around the year 2005, Dennard scaling and thus scaling the operating frequencies was not sustainable anymore for newer technology nodes. Abruptly, the challenge of keeping processor power consumption under control became a first-class constraint for processor designers famously forcing Intel to halt and subsequently abandon two of its — then newest — processor designs under development [9]. Thus, the computing world entered the multi-core and many-core era where more than one processor core was integrated within a single chip or a single package, enabling higher computing performance without increasing the operating frequencies and the associated dynamic power consumption.

Unfortunately, simply following the multi-core design path with ever increasing number of compute cores alone does not alleviate all of the performance limitations caused by power and thermal limitations as was predicted by Esmaeilzadeh et al. in their ground breaking work on *dark silicon* [10]. Their work showcased the risk that future processors would have to switch off an ever increasing percentage of their area, i.e. keeping the silicon *dark* due to power and thermal limitations. The risk of dark silicon motivated a plethora of new research, both into new architectural designs like heterogeneous computing as well as into run-time power and thermal management algorithms. Both, architectural optimizations as well as run-time management algorithms aim to effectively and efficiently manage the limited power budgets of multi-/many-core processors to keep increasing compute performance with power consumption and the resulting chip temperature staying fundamental constraints [11]. While novel design-time optimization techniques reduce power consumption and thus thermal dissipation, e.g. in [12, 13], they still have to be complemented with effective and efficient run-time power and thermal

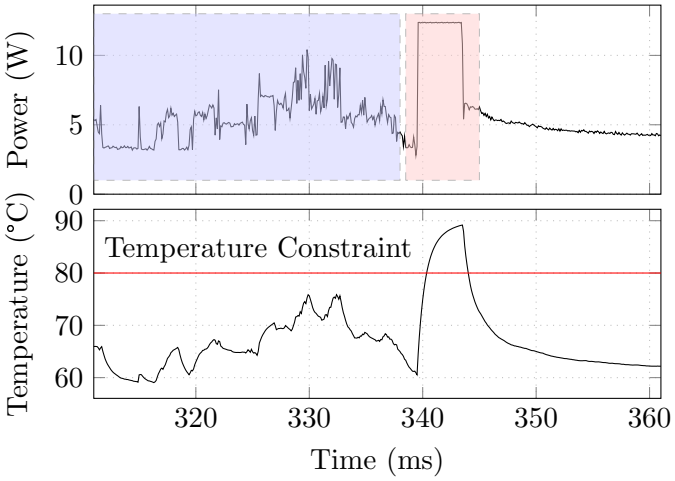


Figure 1.1: Motivational example for power modeling and power forecasting: Splash-2 FFT smaller power phase changes (blue area) require accurate run-time estimations to allow optimal reactive power management decisions. At around 340ms (red area), the rapid power phase change leads to a violation of the thermal constraint (red line). Anticipating such thermal violations by the means of accurate power forecasts would allow to proactively avoid the violation.

management algorithms [14, 15, 16]. Many such run-time power and thermal management algorithms rely on accurate *run-time power information* which is fine-grained in the spatial domain, e.g. on core-level, and the time domain, e.g. with high time resolutions of around 0.1 ms [17, 18, 19, 20, 21, 22]. *Core-level* power information is especially important for resource management as the power densities in a multi-core processor are highest in the compute cores [23]. The main focus of this thesis is on developing workload-dependent models to obtain *core-level* run-time power information at fine-grained time resolutions with negligible run-time overhead. The power consumption of the so-called *uncore*, i.e. chip-level resources not associated with specific cores like shared caches, on-chip communication and I/O interfaces, are not within the scope of this thesis as they are secondary for the run-time resource management.

A short example, showcasing the motivation in obtaining accurate run-time power information in regard to both: *estimates* of current power consumption as well as *forecasting* of future power consumption, is given in Figure 1.1. The execution of the SPLASH-2 *fft* benchmark is shown over an interval of 50 ms with the benchmark's power consumption to be seen at the top and the corresponding thermal response at the bottom.

For *reactive* power management — that is a management that reacts to changes in the workload-dependent power consumption — both the power changes in the blue area (smaller transients) as well as the red area (large transient) need to be accurately known by the management algorithm. Without accurate power consumption estimations, sub-optimal power management decisions will be taken by the integrated power management algorithm.

However, those smaller power transients shown in the green area are not critical enough to necessarily require *proactive* power and thermal management. In contrast, the rapid increase and thus rapid phase change in power consumption indicated in the red area depicts the necessity for proactive power and thermal management algorithms as a purely reactive algorithm would not adapt fast enough ($\leq 1\,\text{ms}$) to avoid hitting the thermal constraint. If the change in power consumption / the thermal gradient were forecast, hitting the thermal constraint would become avoidable and thus the consequent performance degradation could be alleviated.

A multitude of reasons, e.g. shared power grids and challenges in calibration [22] and potential area overheads [24], have prohibited the use of integrated, fine-grained power sensors in multi-/many-core processors. Therefore, model-based run-time power estimation is widely used in today's processor systems, e.g. for Intel [25], AMD [26] and IBM [22] processors. Such run-time power estimation models are generated during design-time and use activity indicators of the currently executed workloads as input during run-time to estimate the workload-dependent contributions to the dynamic power consumption while only incurring small run-time overheads.

However, there are multiple causes for model inaccuracies in run-time power estimations. For example, the activity indicators used as power model inputs can show high levels of multicollinearity [27, 28] and the relation between the observed workload activity and the actual run-time power consumption can be nonlinear [29, 30]. How to automatically account for multicollinearity during model generation and how to accurately capture the nonlinear relation between a workload's activity or current performance and the resulting power consumption with *lightweight* modeling techniques, i.e. with sufficiently low run-time overhead, are still open research questions. The final goal in optimizing the modeling accuracy lies in improving the performance of run-time processor management algorithms and thus the available compute performance for end-users.

Furthermore, estimating the *current* dynamic power consumption is only sufficient for *reactive* power and thermal management algorithms. *Proactive* power and thermal management algorithms which rely on *forecasts* of future power consumption values can offer higher run-time performance [16]. To achieve higher performance, a multi-core processor's power and thermal management algorithms are optimizing the processor execution by directly affecting different parts of the system's state, e.g. by changing the workload-to-core mapping, the core's voltage-frequency pair or its sleep states, and thus also indirectly affecting the processor's instantaneous power consumption, overall energy efficiency and temperature budget. Learning properties of the system, e.g. the forecast power behavior of workloads, that are independent of the management algorithm, allow for reusing the same forecast model. As will be shown, most forecasting models are integrated within specific management algorithms targeting individual resource optimization objectives. A reusable power forecasting model has the advantage of being employable as an input for many (changing) resource management policies and algorithms, irrespective of their objectives and constraints. However, the research question on how to accurately forecast run-time power phase changes with independent forecast model is still open to further research.

The aforementioned open research questions in regard to multicollinearity of activity inputs, nonlinear relations between workload-dependent activity and dynamic power consumption estimation as well as the challenge in forecasting future power phase changes motivate the research undertaken in this thesis.

Contributions of this Thesis The main contributions of this thesis, which have been previously published at three international conferences [1, 2, 5] and as two journal articles [3, 4], are as follows:

- Proposing and evaluating an Independent Component Analysis (ICA)-based methodology to automatically reduce multicollinearity of power estimation model inputs.
- Investigating Feedforward Neural Network (FFNN)-based methodologies to generate nonlinear power models for run-time estimation and proposing a single-objective as well as a multi-objective optimization approach to optimize the FFNN neural architectures towards high estimation accuracy and low run-time overhead.
- Investigating a nonlinear transformation-based methodology for generating easily interpretable nonlinear power estimation models allowing for investigation of the underlying nonlinear activity/power relations.
- Demonstrating that the FFNN-based and the nonlinear input-transformation-based methodologies offer a decrease in relative run-time estimation error in the range of 3.0% - 7.5% compared to linear power and common polynomial modeling techniques while still being sufficiently lightweight to have minor run-time overheads.
- Proposing a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN)-based power forecasting methodology which can more accurately forecast future power phase changes compared to other state-of-the-art approaches.

The main research for these contributions was done in the context and in close collaboration with researchers of the DFG TCRC/SFB 89 Invasive Computing [31, 32]. Most notably, on the topic of power modeling and power and thermal-aware resource management algorithms, there was a close and productive collaboration with the Chair for Embedded Systems (CES) from KIT with the CES group focusing on novel resource management algorithms. Within this collaboration, three papers were developed [4, 5, 33] with the paper on resource management via power- and cache-aware task mapping not being in the scope of this thesis.

Thesis Outline This thesis is structured as follows: First, a short overview on the necessity for run-time power estimation and forecasting as well as an in-depth review of related works is given in Chapter 2 where also the short-comings of the related works in regard to fine-grained power estimation and power forecasting models are described. Afterwards, the experimental framework based on state-of-the-art simulators, their configurations and the limitations in using a simulator-based approach are shown and discussed in Chapter 3. In Chapter 4, the main contributions of this thesis in regard to power models for run-time estimation are presented including their experimental evaluation as well as a comparison of the proposed power estimation methodologies. Afterwards, in Chapter 5 the methodology for generating run-time power forecasting models is shown and the model's performance in regard to its forecasting accuracy is experimentally evaluated. The thesis is concluded in Chapter 6 which also includes an outlook on future works.

2 Related Work

This thesis is concerned with *run-time power estimation and power forecasting* for multi-core processors, where the obtained run-time power information can be used to optimize the run-time operation. Therefore, this chapter introduces related work on processor power consumption, the modeling of such power consumption as well as the forecasting of future power consumption states. First, the mathematical models for circuit-level and microarchitectural-level power consumption are shown, the capabilities and limitations of on-chip sensors are introduced, a short overview of state-of-the-art power and thermal management algorithms is given and *proactive* algorithms using power forecasts are discussed in Section 2.1. Afterwards, a comprehensive literature overview for run-time power estimation modeling techniques is given in Section 2.2 with a focus on the underlying model types and their resolution in the spatial-domain and time-domain. Finally, in Section 2.3 the closely related works on power forecasting and the more loosely related works on power performance prediction are introduced and discussed in detail. In the following, a short definition relating to the core concepts of this thesis is given.

Definitions of Power Prediction/Estimation/Forecast There is currently no well-established consensus in the scientific community on the exact meaning of *prediction*, *estimation* and *forecast* in the context of the performance, power and thermal states of multi-/many-core processors. Thus, the words *prediction*, *estimation* and *forecast* have been used as interchangeable concepts until now. This could lead to confusion about the underlying goals, methodologies and results of the related works and how they relate to this thesis' contributions. Therefore, this thesis proposes and applies the following conceptual definitions for *prediction*, *estimation* and *forecast*, as first presented in [5]:

- **Estimation** will be used when, based on a mathematical model, the value of a dependent variable —the output— is derived from other independent variable(s) — the input(s) — for the same point of time or the same time-interval. For example, the core-level power consumption at time t is **estimated** based on the performance counter values at the same time t. There is no change in the time-domain between the inputs and the outputs of the estimation model and the output at time t can usually not be directly observed or measured.

- **Forecasting** will be used when, based on a mathematical model, the *future* value of a dependent variable is derived from other independent variable(s) or the previous values of the dependent variable itself. For example, the power consumption at time $t+\tau$ is **forecast** based on the performance counter values and power consumption values at time t. There is a change in the time-domain between the inputs and

the outputs of the forecasting model, where the output is a forecast into the future and is for that reason not yet observable.

- **Prediction** will be used when, based on a mathematical model, the value of a dependent variable is derived under the assumption of some future changes in the system state, e.g. task scheduling, task mapping. For example, the potential power consumption and performance of a task is **predicted** for the scenario that the task is remapped from a *little*-core architecture onto a *big*-core architecture. The prediction is always in regard to the effect of intended changes in the system state, e.g. through resource management decisions. Although the prediction is time-domain wise in the future, it would solely happen if actions are taken to change the system state.

Note, that these definitions will be used consistently throughout this thesis when discussing the related work even if the referenced works themselves use different definitions or use these words interchangeably. As stated in the introduction, the contributions of this thesis are in regard to the estimation and forecast of power consumption values, thus, the focus in this chapter will be on discussing comprehensively the related work on power estimation and power forecasting. In addition, related work on the estimation and forecasting of performance and temperature values will also be discussed in detail and their similarities and differences will be compared to pure power estimation and power forecasting to allow for a comprehensive discussion of techniques. The related works which aim to predict the effect of systems changes will also be discussed, however in less detail, when their methodologies are similar to methodologies employed in this thesis and where it is important to clarify the differences between their aims and this thesis' contributions.

2.1 Background on Power, Sensors and Managment Algorithms

First, there is a need to understand how and why power is consumed, i.e. converted into heat, in digital circuits. The leading causes for power consumption will be shown in the next Section 2.1.1 with a focus on dynamic power consumption. Afterwards, the limitations on integrating power sensors will be discussed in Section 2.1.2 as easy-to-be-integrated power sensors would alleviate the necessity for model-based run-time power estimation. Finally, Section 2.1.3 and Section 2.1.4 give an overview on state-of-the-art power and thermal management algorithms and applications for power forecasting.

2.1.1 Circuit-level and Microarchitectural-level Power Consumption

In the following, a short introduction to power consumption of digital circuits based on Complementary Metal-Sxide-Semiconductor (CMOS) technology [34], their relation to today's multi-/many-core processors and the challenges in modeling them is given.

The total power consumption P_{total} of CMOS circuits is given as:

$$P_{total} = P_{dyn} + P_{sc} + P_{leakage} \tag{2.1}$$

where P_{dyn} is the dynamic power consumption of the circuit, P_{sc} is the short circuit power and $P_{leakage}$ is the leakage power.

This total power consumption is commonly divided into the three components:

$$P_{total} = \alpha_{0/1} \cdot C_L \cdot f_{clk} \cdot V_{dd}^x + I_{sc} \cdot V_{dd} + I_{leakage}(T) \cdot V_{dd}, \tag{2.2}$$

where $\alpha_{0/1}$ is the probabilistic activity factor of the circuit, i.e. the average percentage of switches/toggles of the circuit per clock cycle, C_L is the aggregate load capacitance, f_{clk} is the operating frequency of the circuit, V_{dd}^x is the supply voltage with x being a technology dependent exponent (often approximated as two), I_{sc} is the short circuit current when both nMOS and pMOS are conductive during a transition and $I_{leakage}$ is the temperature-dependent leakage current. Throughout this thesis, the operating frequency will be referred to simply as f and usually combined with the supply voltage V_{dd} and thus referred to as Voltage/frequency (Vf)-level in general.

While $P_{leakage}$ does not directly depend on the circuits activity, P_{sc} is directly influenced by how often the circuit switches, however, has also become less critical due to advancements in voltage scaling and new technologies, e.g. Fin Field-Effect Transistor s (FinFETs). Note that the leakage power $P_{leakage}$ is also often referred to as static power consumption P_{static}. The focus of this thesis is on dynamic power consumption and how to estimate it during run-time with limited information.

In the following, the discussion is generalized towards very large and complex designs like a multi-/many-core processor. When looking at the equation for dynamic power consumption in the following:

$$P_{dyn} = \alpha_{0/1} \cdot C_L \cdot f_{clk} \cdot V_{dd}^x, \tag{2.3}$$

the activity factor $0 \leq \alpha_{0/1} \leq 1$ plays an integral part in the consumed power although both the lower and upper limit are theoretical limits of $\alpha_{0/1}$ which will not be encountered in the usual operation of a processor. Note, f_{clk} and V_{dd} can be accurately known (measured) during run-time operation and x can be experimentally determined after initial production. However, the activity factor $\alpha_{0/1}$ is probabilistic in nature and averaged over the activities of millions of basic gates (NOT, NOR, NAND etc.) for a general purpose processors which varies widely over the range of different executed workloads. Observing periodically during run-time how often each gate is switched and their known — simulated or otherwise approximated — load capacitance C_L is charged, is not feasible. One might think that measuring P_{total} on an appropriate level and subtracting approximations of P_{sc} and $P_{leakage}$ could provide run-time information on the consumed dynamic power P_{dyn}, however, on-chip sensors for measuring current are difficult to integrate on-chip and this will be discussed in more detail in Section 2.1.2.

Overall, this leaves the activity factor $\alpha_{0/1}$ and the associated load capacitance C_L as critical unknown factors in determining dynamic power consumption of multi-/many-core processors and their microarchitectural components. On a basic level, the dynamic power modeling challenge is to approximate accurately and periodically the run-time *average* switching activity of millions of circuits through observable activity indicators, e.g. performance counters. A common approach — discussed in depth in Section 2.2 of

this Chapter — taken to model dynamic power consumption $P_{dyn,t}$ at time t is denoted by the following equation [35, 36, 37]:

$$P_{dyn,t} = \boldsymbol{A}_t \cdot \boldsymbol{\beta} \cdot f_{clk,t} \cdot V_{dd,t}^{x}. \tag{2.4}$$

In this equation, the average switching activity factor $\alpha_{0/1}$ is upgraded towards a more complex, observable activity *vector* $\boldsymbol{A}_t$ and the load capacitance C_l is modeled through regression coefficients β determined during experimental evaluation of the processor, or its components, for which a power estimation model is needed. These regression coefficients are also often referred to as *power weights*. By necessity, the length of the *vector* $\boldsymbol{A}_t$, i.e. the overall number of activity indicators, is very limited and many magnitudes lower than the number of integrated gates. The difficulty in generating accurate models for run-time power estimations lies in modeling the factor of $\boldsymbol{A}_t \cdot \boldsymbol{\beta}$ by directly and indirectly capturing the activity of a large number of gates, i.e. large die area, through a very limited set of activity indicators and accurately attribute power consumption — the dynamic reloading of the total capacitance of the underlying gates — to each activity indicator.

Note, that in Equation 2.4 the relation between dynamic power consumption P_{dyn} and:

- operating frequency f_{clk} is linear,
- supply voltage V_{dd}^{x} is nonlinear,
- workload-dependent activity $\boldsymbol{A}$ and associated power weights $\boldsymbol{\beta}$ is linear.

However, this workload-dependent dynamic power contribution has been shown to not necessarily be linear and more advanced nonlinear modeling techniques can be needed to accurately capture these relations [30]. How other related works are modeling this workload-dependent power contribution, i.e. as linear or nonlinear, will also be discussed in Section 2.2 and this thesis investigates multiple nonlinear modeling methodologies in Section 4.2 and Section 4.3. In the following, this thesis will refer to activity indicators as performance counters as that is the most common type of activity indicator used in related power modeling works on multi-/many-core processor run-time power estimation.

2.1.2 On-chip Sensor Capabilities

As was shown in the previous section, the dynamic power consumption $P_{dyn,t}$ of the underlying, power consuming transistors of a processor — or its microarchitectural components for which dynamic power information is needed — can be modeled via Equation 2.4. However, another seemingly obvious approach is to measure the supply voltage $V_{dd,t}$ as well as the current draw I_t at time t to calculate total power consumption as $P_{total} = V_{dd,t} \cdot I_t$. And when solely the dynamic power component is needed, one could subtract a model-based leakage/static power consumption from the measured total power consumption of the processor / microarchitectural components. However, this

is unfortunately not straight-forward as measuring current draw on-chip with a reasonable overhead is an ongoing research topic and therefore, the capabilities and limitations of on-chip sensors are briefly discussed in the following.

Information on both the operating frequency, set through clock generators, and the supply voltage, set by voltage regulators, are transparently and accurately known during run-time operation of a multi-/many-core processor for each core. For example, beginning with the Haswell architecture, Intel processors have Fully Integrated Voltage Regulators (FIVRs) on core-level and can thus configure the supply voltage of each core [25]. Therefore, the supply voltage is deterministic and can be measured for run-time power estimation on suitable time-frames and on core-level as well as the package-level in case of single voltage-island processors.

The difficulties for determining power consumption on a level more fine-grained than the package level lies in measuring and differentiating the current flow within a package for different microarchitectural components or different cores of a multi-/many-core processor. IBM has been quite transparent in the capabilities of their integrated sensors on IBM Power 7 architectures in [38]. Currents are measured on package level and combined with package-level voltage measurements to compute package-level power consumption. For core-level power information it is stated that "It is difficult to isolate the power consumption of each processing element attached to a common power plane or to measure power consumption at the core level using only chip-level current measurements." [38] and therefore an activity-based run-time power estimation model was developed and is used in the Power 7 architecture [38]. An additional challenge, which was mentioned in [22], is that measuring real power consumption would require the processor to be periodically stalled to calibrate the analog sensors. Also, for components which share the same power grid or voltage-island, measuring the individual component's or core's current draw is impossible [22, 39]. In the IBM Power 7 architecture, the digital thermal sensors were also based on models with operating voltage as input and an off-chip thermal diode for calibration.

There have been some works, e.g. [40, 24], proposing circuits for on-chip power (current) sensors with comparatively high time resolution. Voltage drop across a sleep transistor, which connects the permanent power supply to circuit power supply, is exploited to measure the current through the sleep transistor by adding an additional capacitance which is charged proportional to the voltage drop. This is done with fast measurement intervals of 0.5 μs. However, this approach requires specific on-chip resistors and thus the area overhead at 45 nm is still quite significant and to the best of the author's knowledge none of today's commercially available multi-/many-core architectures have integrated such fine-grained, on-chip current sensors. Thus, accurate and fine-grained power estimation models are still required for run-time power and thermal management purposes and other applications of run-time power information as will be discussed in the next section.

2.1.3 Power and Thermal Management Algorithms

Power consumption and the resulting chip temperature are not only fundamental design constraints for modern multi-/many-core processors but also require ongoing run-time monitoring and optimization to achieve high compute performance. In the following, a brief overview of some power and thermal management algorithms and the their inputs used for observing the processor state are given. A plethora of algorithms for run-time power and thermal management which observe and optimize the processor system state have been published and comprehensive, up-to-date survey papers can be found in [16, 41]. One of the main scientific challenges for power and thermal management algorithms is the dynamic behavior of workloads which leads to rapid variations of the processor's — and its microarchitectural component's — power consumption and thermal response. These rapid variations in power consumption and the temperature response require continuous optimization of the multi-/-many-core processor system to the changing conditions. Without continuous optimization, processors will operate in disadvantageous operating states leading to [16, 41]:

- overheating and emergency thermal throttling with significant performance impacts;
- suboptimal usage of thermal headroom in multi-core architectures leading to avoidable performance degradation;
- increased power consumption leading to decreased battery performance for mobile systems;
- long-term reliability degradation due to unnecessary thermal stress on transistors and wires.

Many processors integrate so called frequency *boosting* algorithms which increase the operating frequency of selected cores and their power consumption above the thermally safe level of power consumption for short durations of time, i.e. the algorithms increase the Vf-level above the long-term stable and Thermal Design Power (TDP) of the chip. This is only possible when the power consumption and temperature response is continuously observed and the boosting algorithm reduces the Vf-level to a long-term stable operating point before entering the emergency thermal throttling regime. Such frequency boosting algorithms often require core-level power information to decide when to start/stop the frequency boosting and to chose which boosted frequency level is optimal in regard to performance, e.g. [17, 18, 19]. An novel power-aware boosting algorithm was proposed in [42] for FinFET-based multi-core systems. The algorithm uses core-level power information with estimation rates of 1 kHz to deliberately operate the cores in higher temperature regions as the delay decreases for FinFETs in super-threshold voltage regions and therefore the high temperatures allow for higher throughput. However, targeting higher operating temperatures decreases the margin towards emergency thermal throttling which would substantially decrease performance. Hence, such a methodology requires very accurate, fine-grained run-time power information to operate the processor

on its workload-dependent "sweet-spots". If such power information were not accurately and consistently available, the algorithm would more likely miss these "sweet-spots" and increase the risk for emergency thermal throttling.

Another power and thermal management algorithm aims to completely replace TDP with so called *Thermal Safe Power* (TSP) which computes maximum core-level power consumption values based on the current workload-dependent power consumption values throughout the multi-/many-core processor [20] and sets the Vf-levels accordingly. This allows for significantly higher compute performance while still guaranteeing that no individual core exceeds its safe temperature constraint which would damage the core. However, this approach of course relies on accurate, core-level power information during run-time to be both effective and safe to use. One would expect similar performance degradation scenarios as in [42] if the power information were inaccurate. Also in [21], a novel Dynamic Voltage and Frequency Scaling (DVFS) and task migration algorithm was proposed which requires core-level power information at time intervals of 5 ms, although, other DVFS algorithms have been shown to make DVFS decisions on time scales as low as 32 μs [22]. In conclusion, the combination of:

1. power consumption and heat dissipation being fundamental constraints for computational performance of modern multi-/many-core processor designs and

2. the increasingly complex run-time management algorithms being proposed to be able to fully utilize any potential headroom in regard to power and temperature,

leads to accurate and fine-grained run-time power estimation information becoming even more important.

2.1.4 Applications for Run-time Power Forecasts

Aside of run-time power estimations, another important topic for efficient and effective power and thermal management algorithms is in regard to power forecasts, i.e. models which forecast how future power consumption values will change based on the current workload characteristics. In this section, the motivation and applications for such power forecasts are described.

Referring back to the motivational Figure 1.1 from the beginning, where the rapidly-changing power consumption of the SPLASH-2 [43] *fft* benchmark is shown. This illustrates the importance of both, proactive management and generating power forecasts, which is discussed in more detail in the following. One can observe that around t=330 ms, the instantaneous power consumption reaches up to 10.4 W, however, because these peaks are very short and are interleaved with much longer phases of lower power consumption, the resulting on-chip temperature stays below the temperature constraint, which is usually 80° C. If no temperature sensor were available and without a power forecast, the thermal management — based on current power estimations — would need to throttle the processor to avoid a potential emergency and thus unnecessarily decrease the core's compute performance. With available power forecasts in combination with a design-time temperature model or with available fine-grained temperature sensors—

and their associated area cost, throttling can be avoided. In contrast, at t=339.6 ms, power suddenly increases sharply and stays high for approximately 3 ms. This results in a thermal violation at t=340.4 ms, i.e. only 0.8 ms later, which is too fast for a reactive thermal management algorithm to successfully avoid the thermal violation, e.g. by remapping the task onto a colder core or by proactively decreasing the Vf-level to an appropriate Vf-level allowing for continuous throughput. Such a thermal violation can only be avoided if a power phase accurate forecast is available which would avoid a drastic processor-safety related management decisions, i.e. dropping most processor cycles to avoid overheating and damaging the chip. Only *proactive* management techniques, which anticipate future system behavior, lead to optimal behavior in such a scenario, whereas *reactive* techniques, which only react after the system behavior has changed, always lag one step behind [16].

This lagging effect leads to two different detrimental behaviors of the overall multi-core processor system:

- Power and thermal characteristics changing at time t while the power and thermal management algorithms by necessity had have to take their decision on management actions for time t based on information from time $t-\tau$. These taken actions do not match the power and thermal characteristics until the management algorithm reacts again. Therefore, the employed actions when power and thermal characteristics are rapidly changing are not optimal, possibly resulting in performance, e.g. longer run-time or worse user experience, and energy losses, e.g. inefficient task mapping in heterogeneous systems leading to higher power consumption for the same workload [16].

- The system constraints, e.g. maximum temperature for safe operation, can be violated during the reaction time. To avoid such violations, reactive approaches can enforce guard bands, e.g. a power guard-band throttling the processor before the temperature constraint is hit [44]. This is, however, a conservative approach leading to avoidable performance losses.

A key component to achieve proactive management is the *forecast* of the future system states [16], which means the workload-dependent system states that did not directly arise from management interventions. Also, the integrated power and thermal management of a processor, and of course also on higher level the operating systems, switch between several different resource management policies focusing on different optimization objectives [45]. While it is possible to directly learn the management actions from system observations, e.g. with reinforcement learning [46], such techniques require separate training for each policy [15]. Thus, separately forecasting power and providing such a forecast to any management algorithm that requires it, is key for the development of novel management techniques.

2.2 Run-time Power Estimation

In this section, the related works in regard to run-time power estimation are discussed with a focus on works which are closest to the proposed power estimation methodologies described in Chapter 4 and which explicitly or implicitly aim to capture also the workload-dependent power consumption. To understand and compare the different works, a classification system is used in the following which differentiates the works according to the following characteristics:

- the model type used for power estimations, e.g. linear regression, polynomial regression, Neural Network (NN),
- the Instruction Set Architecture (ISA), e.g. x86, ARM, PowerPC (PPC), of the underlying processors,
- the spatial resolution of the models, e.g. core-level, package-level, system-level,
- their time resolution, i.e. for what time-intervals the power consumption is estimated,
- the workloads used to generate the models, e.g. synthetic workloads, general benchmarks.

Both the spatial resolution and time resolution denote how fine-grained the power models are. For the spatial resolution, system-level means that aside of the Central Processing Unit (CPU), other system components like the Dynamic Random Access Memory (DRAM) were also power modeled and *micro-comps.* denotes that micro-components of the CPU like the execution engine were modeled, i.e. that the resulting power models are more fine-grained than core-level. The type of workloads used for generating the power models show if the power models can be either generated solely through synthetic workloads — occasionally also called microbenchmarks in the related works — or via a combination of synthetic workloads and generic workloads or through generic workloads, e.g. PARSEC and SPLASH-2, alone. An overview of all related works discussed in this section and their classification is given in Table 2.1.

In the following sections, first an in-depth overview of related works on linear and classification power models is given in Section 2.2.1, followed by related works on nonlinear and NN-based models in Section 2.2.2 and Section 2.2.3, respectively. Finally, a discussion on the methods employed in related works on minimizing multicollinearity is given in Section 2.2.4 as well as a discussion on the differences in related works using linear vs. nonlinear modeling techniques in Section 2.2.5.

2.2.1 Linear Power Models

Isci and Martonosi were the first to propose a methodology to model a processor's power consumption based on performance counters [47]; a fine-grained methodology in regard to spatial and time resolution. This was still in the single-core era of processor design

Table 2.1: Overview and classification of related works on run-time power estimation

Related Work	ISA	Model Type	Spatial Res.	Time Res.	Workload Model Gen.
Linear Models					
[47]	x86	pw-linear	micro-comps.	1 ms	synthetic
[48]	x86	class./BBV	CPU	100 MI	generic
[49, 50]	x86	linear	CPU	20 ms & 10 ms	syn.& gen.
[51]	x86	linear	system, CPU	1 s	syn & gen.
[52]	x86, PPC	linear	system, CPU	10 ms	syn. & gen.
[53, 36]	x86	linear	micro-comps.	10 ms	synthetic
[39, 22, 38, 37]	PPC	linear	core	32 μs	synthetic
[27]	x86	linear	CPU	workload	syn. & gen.
[54]	x86	linear	core	200 ms	generic
[55]	x86	pw-linear	core	1 s	synthetic
[56]	x86	linear	(un)core	1 s	synthetic
[57]	ARM	linear	system, CPU	10 s	synthetic
[58]	ARM	linear	CPU	-	synthetic
[59, 28]	ARM	linear	core	50 ms	syn. & gen.
[60]	ARM	linear	core	10 ms-100 ms	syn. & online
[61]	ARM	linear	core	500 ms	syn. & gen.
[62, 63, 64, 25]	x86	linear	core	1 ms	synthetic
[65, 66, 26]	x86	linear	core	10 ms & 1 ms	-
[67]	ARM	linear	core	cycle	synthetic
Nonlinear Models					
[30]	x86	lin. & nonlin.	core	1 s	syn. & gen.
[68]	x86, PPC	nonlinear	system	-	generic
[69]	x86	nonlinear	system	5 s	generic
[70]	x86	nonlinear	CPU	200 ms	synthetic
Neural Network-based Models					
[71, 72]	x86	FFNN	system, CPU	1 s & 333 ms	synthetic
[73]	x86	RF, NNs	system	workload	generic
[74]	x86	NNs	system	1 s	generic
[75]	x86	NNs	system	1 s	generic
[76]	x86	linear, NNs	CPU	30 ms	synthetic

and done for an x86 ISA Intel Pentium 4 processor and established a fundamental approach for processor power estimation. They instrumented each of the 17 processor's 12 V power supply lines with a current probe and periodically measured the current to compute the total power consumption of the processor. In parallel, specific of performance counters were also read and the data gathered for later model generation. To

generate a run-time power model, first the single-core processor was subdivided into 22 different microarchitectural-components (micro-comp.), e.g. Branch Predictor Unit (BPU), caches, Memory Order Buffer (MOB), integer/floating point execution units. Then, custom-designed synthetic (micro)-benchmarks were executed on the system to entice a targeted power response of each individual microarchitectural-component. This allows the regression of a microarchitectural-component's observed activity level with the measured processor power which leads to approximate power scaling factors of each component in dependence of its activity. This work also identified some nonlinear relations and used a piece-wise (pw) linear model for a subset of the components. Component-level power can then be derived by multiplying the microarchitectural-component activity levels with their specific architectural scaling factor, a max power scaling factor and adding a non-gated-clock power value. Total processor power was derived by adding up the component-level power consumption as well as the pre-determined idle power of the processor. Finally, Isci and Martonosi verified their approach by comparing estimated total power — using their power model with performance counter inputs — with the actual measured processor power when full Standard Performance Evaluation Corporation (SPEC) benchmarks and linux desktop applications were executed. They observed good modeling accuracy of their proposed performance counter approach for total processor power. This work demonstrated the viability of performance counter-based power estimation, proposed the modeling of microarchitectural-components using synthetic benchmarks and highlighted the possibility of nonlinear relations between power response and observable component activity. In regard to nonlinear relations, they specifically highlighted that small increases in performance counter values above zero indicated a large power response. Therefore, for a subset of the 22 used performance counters, a piece-wise linear modeling approach was chosen. However, no explicit nonlinear modeling approach was investigated nor used for modeling core power consumption in this work.

The work in [47] was followed up with a work in [48] by the same authors with a focus on power phase characterization where a performance counter-based approach is proposed and compared to a Basic Block Vector (BBV)-based approach with the goal of distinguishing five different power phases. Although power phase characterization can be interpreted as a form of power estimation where the goal is to characterize and classify longer program phases — in the range of 100 MI (mega instructions) — and make such power phases comparable throughout different workloads. These phase characterizations can then be used for power and thermal management as well as software optimization with a focus on minimizing power consumption, e.g. by adapting the program code towards shorter high-power phases. The proposed performance counter-based phase characterization clusters the observed values of 15 different performance counters and corresponding power consumption values when SPEC benchmarks are executed on an Intel Pentium 4 processor. In contrast, the BBV-based approach instruments the program code and counts the number of touched basic blocks over time, generates a BBV based on that and afterwards clusters these BBV according to their average power consumption. The interesting finding of this work in regard of power phase characterization, is that their proposed performance counter-based power phase characterization is significantly more accurate compared to also in this work investigated BBV-based

approach. This means that even low-level and detailed program code information yields less accurate power consumption / power phase information compared to the usage of performance counters. Thus the actually observed activity on the processor, due to different operand values in otherwise identical code segments, changing data locality during execution and possibly other characteristics emerging from complex processor architectures provide crucial information for modeling run-time power consumption.

Bircher et al. investigated the correlation between power consumption and activity observed through performance counters in [49] for a single-core Pentium 4 CPU. Although this work does not propose a power model, it provides insights into the relations of performance counters and power consumption at measurement intervals of 20 ms. Significantly differing correlation coefficients between different benchmarks were observed for the same performance counter and power response combination. Therefore, this work warns of "looking at only a limited set or workloads" to determine correlation coefficients which implies that using also generic benchmarks — if possible — during power model generation could increase modeling accuracy. Similarly to [29], the conclusion is that the power consumption of a CPU can be estimated using available performance counter information. This work was followed up in [50] with a proposal for a linear regression power model which was generated by using both synthetic as well as generic SPEC benchmarks. The measurement and estimation interval was decreased to 10 ms while otherwise keeping the same experimental setup of the previous work. One finding was that long-latency speculative instruction execution and their contribution to power consumption is not easily modeled and therefore, custom-designed synthetic benchmarks were proposed to clearly distinguish and incorporate such effects into the linear regression power model. In contrast, this thesis proposes an approach on how to use purely generic rather than synthetic workloads to generate run-time power models which would allow to make the power modeling approach more architecture-agnostic. Both of these works [49, 50] were extended in [51] with full system power estimation including the CPU, memory, I/O, chipset, disk and network of computer system with four Pentium IV Xeon processors. The four processors were modeled as a single CPU and power consumption estimated at a rate of 1 Hz. Notably, the power of all system-level components could be derived from processor-level performance counter values. Afterwards, in [52] the previous system-level power estimation methodology was extended to multi-core systems and evaluated on IBM PowerPC (PPC) server processors and an AMD dual-core processor where again other parts of the system were also power modeled. This work also argues for large amounts of generic workloads to generate power models as they state that: "Without a sufficiently large range of samples, complex power relationships may appear to be simple linear ones". However, the workload-dependent dynamic power consumption is — similarly to [49, 50, 51] — modeled as a linear regression. Also, the dynamic power consumption of individual cores is not differentiated in this work.

Another work in [53] aimed for so called *decomposable* power models, i.e. power models on microarchitectural-component level similar to the approach in [29], for multi-core processors specifically an Intel Core 2 Duo processor with two cores. This work designed 97 different synthetic workloads to isolate and decouple the observed performance counter values and the overall power responses of the different microarchitectural-components.

The necessity for these many different synthetic workloads arises due to the power response of some components can be highly correlated and some components do not offer a directly observable, associated performance counter. In their power model, each of the individual cores consist of a static power contribution added to the sum of the core's components power consumption which are modeled through their activity ratio multiplied with a so called power weight. These power weights are derived using multiple linear regressions on the performance counter and power consumption data for each components associated synthetic workloads, i.e. the overall model again is a linear power model. Notably, an in-depth knowledge of the processor's microarchitecture is needed to generated such a decomposable power model on microarchitectural-component level. However, in this author's opinion there has been a trend of processor manufacturing companies towards providing less information on microarchitectural implementation details and power models being integrated into the processors themselves without in-depth information on their models. This work [53] was followed up and extended in [36] with a comparison of the microarchitectural-component level power model with four other power modeling approaches. The same experimental setup and again 97 synthetic workloads — unfortunately without many implementation details — were used to generate the microarchitectural-component level power model with the workload-dependent core power model being a linear regression model. Core-level power consumption estimations are generated by only adding the specific core's microarchitectural-component level power values and splitting the overall static power of the whole processor onto the cores and the uncore. The models for comparison were linear regression models with lower numbers of performance counter inputs, e.g. only IPC or IPC+memory associated performance counters, using a single linear regression over performance counter and power consumption data. Notably, these comparison models were generated with data from the generic SPEC *CPU2006* benchmarks and resulted in significantly higher power estimation errors. These results highlight that using only generic benchmarks / workloads to generate power models needs either a sophisticated method to incorporate / reduce the correlation of the observed performance counters or more complex modeling methodologies, for example nonlinear power models.

As previously mentioned, IBM has described the difficulties in integrating power sensors on microarchitectural-component or core level and therefore only measures power on package/CPU-level and for other parts of their IBM Power 7 computing systems, e.g. memory and I/O [38]. Due to the necessity of fine-grained power information, a run-time power estimation model — in this case so called *power proxies* — was implemented on IBM's 8-core Power 7 chip line-up [39, 22, 37]. These core-level power estimation models rely on more than 50 architectural performance events, corresponding to performance counters in other works but not described in detail, to provide run-time activity information. The dynamic power model itself is a linear model in regard to the workload-dependent power consumption with an added power offset factor to derive total core-level power consumption. In-depth knowledge of the processor microarchitecture was used to choose an optimal set of performance events/counters. The final core-level power model is constituted again of small power regression models for each pre-selected microarchitectural component for which the performance / power relation was determined by using

a set of — not further specified — targeted benchmarks, i.e. synthetic workloads. This core-level power information obtained at very high estimation rates of 32 μs is then used in a DVFS power and performance management algorithm to optimize overall power consumption, e.g. with steep power take-down curves in case of lower loads. In addition, the power information is used for rebalancing of the power consumption between compute cores and on-chip memory as well as for frequency boosting. This approach relies on in-depth microarchitectural knowledge, similarly to other approaches [53, 36], to find maximum correlation between activity information and power consumption for many different microarchitectural components of a processor's core.

A statistical approach for performance counter selection for a CPU-level power model of an Intel Xeon (Haswell) processor was proposed in [27]. To determine which performance counters to incorporate into the power model, the Variance Inflation Factor (VIF) of each performance counter in relation to all other performance counters is computed, i.e. each performance counter is once the dependent variable while the other performance counters function as independent variables. The VIF is used as an indicator for multicollinearity, the higher the VIF value the higher the multicollinearity, and performance counters with too high multicollinearity are removed as input to the power estimation model. This approach leads to a systematically simpler power model, i.e. less inputs, and resulted in high total power consumption estimation accuracy. The workload-dependent part of the power model itself is again a linear regression model and both synthetic as well as generic workloads were used to generate the power model. Finally, no specific estimation rate could be found throughout this work and it is assumed by this thesis' author that this work does not necessarily describe or evaluates a run-time power model, but rather a power model for full workload executions, i.e. how much power a specific workload at a specific Vf-level consumed on average. This could potentially be extended and be implemented as a fine-grained run-time power estimation model. The authors also highlighted the challenge of multicollinearity and that an approach solely using synthetic workloads is not sufficient to reduce multicollinearity and the associated modeling errors.

A run-time power estimation *and* power prediction for different Vf levels — which will be discussed in Section 2.3 — methodologies were presented and evaluated on AMD CPUs in [54]. The power estimation model is a linear regression model on core-level using eight performance counter inputs and solely uses generic benchmark suits to generate the data for model generation. This is interesting as other works — at least partially — required some form of synthetic workloads to accurately capture the power behavior for wide ranges of workloads. The reason for this likely lies with the low time resolution of only 200 ms and the types of benchmarks used for both training and evaluation where the authors describe that specific PARSEC benchmarks showed very high *average* error of up to 49%. Such high average errors for single benchmarks might be an indication that the power behavior of more complex workloads was not accurately captured by this power modeling methodology with this affect being masked due to the large amount of older (and usually) simpler employed benchmark suites with overall 156 different benchmarks being used in this work.

An early work on multi-core power modeling of AMD and Intel processors investigating nonlinear, core level power models is shown in [55]. The idea of using nonlinear transformations for the performance counters is mentioned in the methodology with the authors deciding to use piece-wise *linear* functions — similar to [29] — to generate the workload-dependent, i.e. dynamic power, portion of the run-time power model with the reasoning that this offers lower modeling complexity. Again, synthetic benchmarks are used to elicit targeted power responses of the different systems. The mentioned nonlinear transformation functions and their possibility for nonlinear modeling of the workload-dependent power consumption are not further investigated in this work. Overall, good estimation accuracy of total system power measured with a wall-meter was achieved by this approach. However, the coarse-granularity of the methodology in the time-domain (1 s estimation intervals) and the mentioned but missing investigation of the nonlinear transformation functions motivates this thesis to further investigate nonlinear transformation based approaches for very fine-grained run-time power estimation models in Section 4.3. The work in [55] was followed up later on in [56] systematically generating explicit power models for the dynamic/static, uncore/core power consumption using a set of synthetic benchmarks for an Intel Haswell multi-core system. For the dynamic core power model, a simple linear regression approach rather than the previous piece-wise linear approach was used again with a time-resolution of 1 s.

With ARM processor designs becoming ever more prevalent in the computing space, e.g. in smartphones and other embedded systems, determining their run-time power consumption also gained research interest with one of the first works proposed by Zhang et al. in [57] where the authors described their so called *PowerBooter*. A system-level power modeling approach incorporating separate power models for each of an HTC mobile phone with an ARM11 CPU was proposed. The workload-dependent power contributions are again modeled as linear. To derive accurate regression coefficients, a set of synthetic workloads is used which periodically varies the utilization of each mobile phone component (e.g. CPU, LCD display, Wi-Fi, GPS, cellular) over its operating range, i.e. from zero to maximum activity. The CPU power model itself is very simple with two workload / activity-dependent regression coefficients and one static, additive coefficient. This seems to be an accurate approach as the integrated ARM CPU is still low architectural complexity and thus has low variance in regard to its dynamic power consumption.

Another methodology for power estimation of embedded systems using comparatively simple ARM9 cores and more complex ARM Cortex-A8 and ARM Cortex-A9 cores was published in [58]. The power behavior of these ARM processors was characterized by executing synthetic workloads on them, measuring board power and performance counters and using the such generated data for multivariate linear regression analyses for each ARM core architecture with the time resolution of the generated power model not being explicitly stated. However, the time resolution or accuracy in regard to power consumption attributed towards performance counter values is sufficiently high to allow the resulting power models to be then incorporated into a cycle accurate simulator for fast design space explorations. The evaluation of this cycle accurate simulator with power simulation is then done again on workload-level, i.e. how close the mean simulated

power consumption is to the mean measured power consumption of different workloads. One interesting result of this work is that the variation in mean power consumption between different benchmarks on the ARM processors is quite low ($\leq 10\%$) which implies that this approach might not be easily transferable towards fine-grained run-time power estimation of more complex x86 architectures.

These investigations of ARM processor architectures were followed by Walker et al. in [59] which was substantially extended in [28] investigating ARM Cortex-A7 and Cortex-A15 CPU clusters. Core-level, medium high estimation rate power models where generated which in addition also distinguishes between static and dynamic power consumption. Both synthetic and generic workloads where used to generate the experimental data for generating the linear regression power estimation models. In addition, a similar methodology to the work in [27] is provided to minimize the multicollinearity of the performance counter input information. In a first stage, only performance counters are included in the model which have low level of multicollinearity, quantified by computing the VIFs between each possible performance counter input in relation to all other performance counters. In a second stage, performance counters which would have been discarded in the first stage due to their high VIFs are manually transformed — using in-depth microarchitectural knowledge — to reduce their multicollinearity and then evaluated for model inclusion. In contrast, our approach generates core level power models and does not use any synthetic workloads. Besides, it minimizes multicollinearity through microarchitecture agnostic statistical methods and does not rely on manual transformations of the activity information which requires knowledge of a system's properties.

A notable approach with online updates of a combined power estimation model and performance estimation model for a mobile platform with ARM Cortex-7 cores, ARM Cortex-A15 cores and a Mali Graphics Processing Unit (GPU) is given by Mandal et al. in [60]. At first, linear workload-dependent power and performance estimation model are generated — similarly to the other related works — during design-time using a set of synthetic workloads. During run-time, each executed application is subdivided into epochs of 10 to 100 millions of instructions based on their performance characteristics with the power-performance estimation model being updated after each epoch using an online Recursive Least Squares (RLS) algorithm. Although the power model itself is linear in regard to performance counters, high online power modeling accuracy is still achieved for unknown applications after observing multiple epochs, i.e. the results show convergence after 10 epochs the latest. This run-time power-performance estimates are feed into a NN-based policy maker deciding the number of simultaneously active ARM Cortex-7 cores and ARM Cortex-A15 cores as well as their Vf-levels to optimize power consumption and performance. However, the power dynamics of the mobile platform with ARM Cortex cores seems comparably low — as can also be seen in [58] — and the estimation rate is per variable application epoch, i.e. in the range of 10 Hz-100 Hz. For higher estimation rates, the regression coefficients of the power estimation model would have to be updated with higher frequencies using the RLS algorithm. With 6 μs computational overhead per estimation (on the mobile platform), this would add 6%

computational overhead at an estimation rate of 10 kHz which is computationally cost prohibitive.

In [61], a power estimation model for the heterogeneous ARM big.LITTLE multi-core architecture as well as cross-core power prediction techniques are proposed. For the power estimation models, generic benchmarks and targeted synthetic benchmarks are executed on the system to generate varying performance counter and power consumption data. The total power consumption of the small ARM Cortex-A7 cores is modeled as basically static due to the workload-dependent power changes being very small with less than 10% variation throughout different workloads, while the dynamic power of the big cores is modeled as a linear regression model. This work also indicates that for embedded, mobile computing systems the power consumption can still be modeled well through linear models and nonlinear modeling approaches with higher model complexity and thus possibly higher estimation accuracy are not *yet* required. However, with the increasing architectural complexity and computational performance of embedded systems, nonlinear modeling approaches might also be required for future embedded systems when a high degree of power estimation accuracy is desired.

Due to the importance of run-time power consumption information, major processor designers — in addition to the already discussed IBM power proxies — like Intel and AMD also integrate their own power measurement and power modeling approaches in their products. Intel integrates so called Running Average Power Limit (RAPL) [62, 63] in their processor line-up providing run-time power consumption information on core-level, the uncore-level, the package-level and the -level. The power information on core-level and uncore-level was at least in the early generations derived from linear power estimation models using performance counter inputs. In contrast to for example IBM processors, there is less public information on how the power model is generated, however, the use of synthetic benchmarks was confirmed in [62]. There are multiple independent research works investigating the Intel RAPL power models [64, 66, 25] in regard to their capability as well as possible security risks due to side-channel attacks. With the spatial resolution being quite high on core-level, the time resolution is reported at an similarly fine-grained resolution of 1 ms for end-users with CPU internal power and thermal management possibly having access to an even higher time resolution which is not communicated off-chip at that those high rates. Note that DVFS decisions can be taken on the same Intel processor platform with 0.5 ms periodicity. Independent measurements on package and system level show that the estimate of the total power consumption — added up from the core and uncore power estimates — has substantial room for improvement in regard to its accuracy and error variance for different workloads.

AMD also integrates a power estimation model in their processor line-up called Application Power Management (APM) [65, 66] and has been removed for the newer *Zen* architecture which now also integrates RAPL [26]. The core-level power information is in both cases derived from power models using performance counters as for IBM and Intel processors. Unfortunately, in-depth information on the used power models is not publicly available and it is not known what type of workloads (synthetic or generic) were used to generate the underlying power models. However, due to the inaccuracies of the power estimations on package level which were measured independently in [66, 26] as

well as the fact that IBM and Intel use linear power models for the workload-dependent power contributions, one can assume AMD has also integrated linear power models.

Finally, a very interesting combined design-time and run-time power modeling approach, so called *APOLLO*, was presented in [67]. Note that this work has been published after all power estimation related papers which were contributing towards this thesis had already been submitted or been published and thus was not discussed and incorporated into those papers. The approach is kept generic in regard to the ISA for different processors with a final evaluation of the obtained design-time and run-time power models on simulated ARM Cortex-A77 and ARM Neoverse N1 processors. Comprehensive data for model generation is obtained through an automated synthetic workload generation methodology which generates over 1,000 different synthetic workloads to provoke a wide range of different power behaviors of the given processor architecture. First, based on Register-Transfer Level (OLS)-level performance and power simulations and using Least Absolute Shrinkage and Selection Operator (LASSO), activity indicators are chosen from the available signal set to accurately capture the workload-dependent dynamic power behavior while minimizing the number of needed activity indicators, i.e. these basically take over the function of performance counters as discussed for previous works. This step also generates the power weights for each selected activity indicator signal. Less than 0.05% of all available OLS signals are selected as activity indicator signals for the power estimation model and are used as binary toggle activity indicators which do not count performance events like previously used performance counters. The final power model is again a linear power model, however, defined as a binary cycle-level power model, i.e. each cycle the toggling of activity indicators are observed and associated if the respective activity indicators where toggled, power weights are added towards that specific cycle's dynamic power consumption. This approach allows the power model to be integrated on-chip through relatively few `XOR` and `NAND` gates and low bit-width `ADDERs` while avoiding costly large bit-width `MULTIPLIERs` and `COUNTERs`; resulting in area/power overhead of less than 1% compared to the area/power requirements of the cores themselves. Overall, this approach promises a low-overhead, high accuracy integrated run-time power estimation model.

2.2.2 Nonlinear Power Models

McCullough et al. [30] were the first to identify and highlight the importance of *nonlinear* relations between observable performance counters and power consumption due to changing workloads and the associated dynamic power response on core-level. Therefore, this work argued for and investigated both a set of linear (Ordinary Least Squares (OLS), LASSO) power modeling approaches as well as nonlinear power modeling approaches (polynomial, exponential and Support Vector Machine (SVM)-based) power models, to account for such nonlinear performance / power relations. First, they traced the power consumption of an Intel Core i7 multi-core processor, broke down power consumption on core-level, and explored both linear and nonlinear modeling techniques (polynomial regression and support vector regression) to model power consumption on core-level. Interestingly, the power measurement setup was able to distinguish core

power, uncore power and integrated graphics power of the processor based on the Intel Capella platform. Although the spatial resolution of the power model is well-suited for power and thermal management purposes, the time resolution with a sampling rate and estimation rate at 1 Hz was comparatively low. Overall, their results showed that their nonlinear models did not significantly improve upon the linear models in regard to their accuracy of modeling total power consumption. More detailed investigation yielded that estimation accuracy degraded on core-level for both linear and nonlinear models. However, the nonlinear models yielded overall lower estimation rates, especially in regard to worst case estimation errors. Furthermore, the results indicated that the proposed nonlinear modeling techniques still suffered from not capturing the nonlinear relations well enough, i.e. possibly due to overfitting parts of the performance counter and power consumption data which then leads to worse power estimations in other parts of the data. Another possible explanation for these inaccuracies mentioned by the authors might be hidden power states. Overall, these results yields the question if a more complex nonlinear modeling approach could lead to higher accuracy power estimations, especially on core-level. Also, two additional questions arise in regard to higher power estimation rates: how is the accuracy of nonlinear power models affected when one has estimation rates applicable for fast power and thermal management algorithms, e.g. 10 kHz? What would be the run-time overhead of such a nonlinear power estimation model at such high estimation rates and would its integration into a processor be feasible? These questions will be tackled in Section 4.2 and Section 4.3.

Hsu and Poole analyzed the power response in regard to utilization over a wide range of aggressively power managed — including DVFS — server systems from different vendors and using Intel, AMD and IBM processors which were executing *SPECpower* benchmark suite in [68]. Different linear functions and nonlinear functions were compared in regard to overall system-level power modeling with utilization/activity as input and how accurately one can fit the shape of the system-level power curve of servers when both utilization *and* the voltage/frequency operating points change dynamically. However, this work does not propose a run-time power modeling approach, i.e this work is for server system power consumption comparison, nor are specifically nonlinear workload-dependent power responses at fixed Vf operating points investigated.

Another work [70] generated stochastic power models for an AMD and Intel processor using only the CPU utilization as model input rather than a larger set of performance counter inputs. Both, utilization of the processors as well as their power consumption were modeled as random stochastic variables based on experimental data and the stochastic relationship between both was modeled as rather simple linear and quadratic functions. This power model incorporates the different Vf-levels of the processors as well as implicit power and thermal management decisions, i.e. Vf-levels are not explicitly modeled but the processor usually set optimized Vf-levels based on utilization and with utilization being the only input to the power model, it is implicitly captured by the power model. This is an interesting approach towards power modeling, likely well-suited for server-level power modeling used for power budgeting and power accounting. However, due to the models spatial and time coarseness and Vf-levels being model-implicit,

the power estimations are not suited as input for run-time power and thermal management.

Another coarse-grained (server-level and 5 s estimation time intervals) machine learning was proposed by Dhiman et al. to model the power consumption of virtual machines [69]. Gaussian Mixture Models (GMMs) were proposed to identify and classify operating points in the performance / power space based on performance counter characteristics of an application which are then used to attribute the corresponding run-time power consumption when similar performance operating points are identified during runtime. This work shows that even at identical utilization levels, power consumption can deviate by a factor of 2 due to differing workload behavior at least on this specific coarse-grained spatial and time resolution. Notably, the nonlinear GMM-based model trained solely with generic workloads performed better than linear regression models although it also required a large number of operating points to be classified. This again leads to the question if nonlinear modeling approaches and/or using generic workloads are suitable for fine-grained run-time power estimation models.

2.2.3 Neural Network-based Power Models

Further nonlinear modeling techniques gained traction in the research space for more coarse-grained power/energy modeling for use on data-center systems and cloud servers. Neural networks in general and especially deep learning NN are well-suited for nonlinear modeling. The first to propose the use of FFNN for power modeling and run-time power estimation were Cupertino et al. in [71]. They investigated FFNN architectures with two hidden layers for modeling the power consumption of an Intel multi-core CPU with the training data in regard to performance counters and power consumption were generated with synthetic benchmarks. The neurons per layer were subsequently increased in the first and second layer while keeping the second layer smaller than the first layer to optimize the neural architecture for power modeling. A neural architecture with 20 neurons in the first layer and five neurons in the second layer provided the highest accuracy improvements compared to state-of-the-art linear power modeling techniques. However, the very low sampling rates, and thus time resolution of 1 Hz - 3 Hz for power and performance counter data, limits the applicability to energy accounting and load balancing in data-centers as power and thermal management algorithms require far core-level spatial resolution and estimation rates in the magnitude of up to 10 kHz. Also, the run-time overhead of the FFNN-based power model in regard to necessary FFNN inference was not evaluated and the solution space of the neural architecture search was rather limited. In contrast, 4.2 proposes a FFNN-based power modeling approach for high rate power estimations of 10 kHz and provides two different methodologies to systematically generate well-suited FFNN architectures with both high power estimation accuracy and low run-time overhead with a software and a hardware based proposal for run-time inference of the power estimating FFNN model. The work in [71] was extended and embedded in a thermal management algorithm in [72] using the FFNN-based power models of the different system components (CPU, DRAM, Network Interface Card (NIC)) during run-

time to manage heat dissipation, to manage energy and optimize data-centers towards more cost-effective operation of the cooling operations.

Another work investigating NN-based power models for multi-cores is [73] where Random Forest (RF)-based and NN-based power models were generated and compared to linear regression models for two Intel multi-core server platforms. The so-called *additivity* of performance counters regarding power modeling of multi-cores is explored where additivity denotes the robustness of reusing a performance counter as model input for a wide range of applications. It was found that even highly correlated performance counters can be used as long as their input to the power model is also highly additive, at least in case of the NN-based power models as these showed highest accuracy for highly correlated *and* highly additive performance counter inputs. This could be an indication that NN-based power models are also well suited even for performance counter inputs with high multicollinearity. However, this work did not investigate optimization of the neural network architectures for power modeling. Time resolution was only on workload level, i.e. power estimations for full workload/benchmark runs and thus not periodic power estimations. Also the spatial resolution was on system-level and thus for spatial and time resolution not applicable for power and thermal management but applicable for data-center management and power accounting.

The work in [74] also investigates NN-based power modeling approaches (FFNN, Elman NN and LSTM RNN) for server and data-center management and optimization for an Intel multi-core system. Neural architectures of the different NNs were not explicitly optimized with the FFNN and the Elman NN parameterized with a single 25 neuron hidden layer and the LSTM RNN parameterized with two hidden layers and ten neurons per layer. The resulting NN-based power models provided high power estimation accuracy at course-grained spatial (system-level) and time (1 Hz) resolution with the FFNN and LSTM RNN offering the highest accuracy, also when compared to multiple linear regression power models as well as a SVM-based power model. Notably, both Elman NNs and LSTM RNNs are able to learn from time-series information, however, it was not clear from this work if this additional modeling capacity contributed towards better power modeling accuracy in case of the LSTM RNN which noted to have had the highest run-time overhead of all investigated NNs.

Elman NNs for power modeling were also investigated in [75] for data-center and server-level power estimation with the goal of capturing potential time-series effects in the power consumption and performance counter data which included performance counter data on CPU, DRAM, I/O and hard disk. The neural architecture of the Elman NN was optimized by comparing architectures with six, nine, twelve and fifteen neurons in the single-layered Elman architecture as well as *step back*, i.e. how far back previous inputs are used for the current power estimation, values of one, two and three. A comparatively simple architecture of twelve neurons and a step back of one offered the highest power estimation accuracy. This indicates that for such coarse-grained power estimation models based on NNs, low complexity NN work quite well which leaves open the question what kind of NN architectures would be needed for fine-grained NN-based power models applicable for power and thermal management.

A run-time power estimation model as well as a performance prediction model for multi-core system under multi-threading inference were explored in [76]. In addition to a linear regression model for power estimation, also a three-layered FFNN with sigmoid activation function was investigated. High modeling accuracies were found on the training data for both the linear power estimation model as well as the FFNN-based power estimation model, however, due to the FFNN model not performing significantly better while being more complex than the linear model, the FFNN power estimation approach was not further investigated. No details were given on how/if the neural architectural hyperparameters were optimized and the power models are coarse-grained in the spatial domain and thus less relevant in regard to their application for power and thermal management. In contrast, this thesis will investigate the hyperparameter optimization for fine-grained — in the spatial and time domain — FFNN-based power estimation models.

In another work [77], which is less related to the aims to this thesis as it concerns the power consumption of the GPU and its memory, three different machine learning techniques are investigated — sequential minimal optimization regression, simple linear regression, and decision trees — for power consumption estimation for different Vf-levels. Notably, the nonlinear machine learning approaches yield far higher estimation accuracies than linear regression models which are further improved by combining different machine learning approaches into ensemble models, i.e. averaging the estimated power values. However, no information for a potential run-time application and its associated overhead of the thus generated power estimation models is provided.

Besides run-time estimation models discussed above, there are also a multitude of design-time power models which are simulation based, e.g. McPAT [78], for architectural power modeling used for architecture-design, the development of run-time power and thermal management and modeling algorithms. Although, such power simulators are highly accurate, they cannot be used for run-time power estimation due to their large computational overhead, e.g. one invocation of McPAT takes 40 seconds on an Intel Core i5-3470 to estimate the power consumption of an Alpha 21364 processor. However, simulation-based power modeling and its trade-offs will be discussed in more detail in Chapter 3 as this thesis also relies on design-time power simulations to evaluate methodologies for run-time power estimation.

2.2.4 Discussion of Methods on Minimizing Multicollinearity

One of the goals of this thesis in regard to run-time power estimation is to avoid minimize multicollinearity of the performance counter inputs for model generation. Some works [59, 28, 27] have dealt with this problem through automated identification of multicollinearity and either removing inputs with low information value and high multicollinearity and manually transforming performance counter inputs with high multicollinearity and high information value for power modeling into derived variables with low multicollinearity. One downside of this approach is that still manual transformation of the performance counters is needed. Many works, as can be seen in Table 2.1, are using synthetic benchmarks to successfully avoid or reduce multicollinearity by generating

power models from many microarchitectural components which target those specific microarchitectural components such that the resulting performance counter and power data cannot show multicollinearity. There are, however, multiple downsides to this approach when relying purely on manually-generated synthetic workloads:

- they do not capture a wide range of workload behavior,
- they do not capture the interaction of microarchitectural components when those are active in parallel,
- they do not allow for online refinement of the power model,
- they rely on in-depth microarchitectural knowledge and are not easily reproducible.

Some works, e.g. [49, 52, 79, 51, 50], combine generic workloads with synthetic workloads to achieve both low multicollinearity and large workload coverage. The downsides of needing in-depth microarchitectural knowledge of the processor design — which might not be publicly available — and the problem of reproducibility remain. Overall, it would be advantageous to have a methodology which automatically reduces multicollinearity when generating power models and which does not rely on manual fine-tuning of the power model and or the workloads used for generating the performance counter and power data. One such ICA-based methodology will be presented in Section 4.1. A final observation from the above discussion of these related works is that also when generating coarse-grained NN-based power models with generic workloads and not using any proactive measures to remove/reduce multicollinearity seems to have lead to accurate run-time models in [73, 74, 75]. This could indicate that the added complexity of NN models is sufficient to also automatically account for multicollinearity in the performance counter input data.

2.2.5 Discussion of Linear vs. Nonlinear Modeling Methodologies

As can be seen in Table 2.1, most related works on run-time power estimation model the relationship between performance counters and dynamic power consumption as linear, i.e. when the Vf-levels are held constant and the workload's activity varies, the dynamic power contribution to total power is assumed to be linear. Already in the beginning of run-time power modeling and estimation research, even before the advent of multi-/many-core processors, possible nonlinearities in the workload-dependent performance and power relationship had been identified [29]. However, most following works did not aim to model or account for possible nonlinearities until a work on multi-core processors [30] again highlighted the possibility of nonlinearities and investigated multiple nonlinear modeling techniques (polynomial, exponential and SVM-based) to capture these effects with no significant accuracy improvement over linear regression models.

Some works [55, 70, 69], with rather coarse-grained spatial and time resolution and server-level power modeling, also used piece-wise linear or nonlinear power modeling approaches. Other works [71, 72, 73, 74, 75], on the same coarse-grained resolution, use

NN-based models. Such NN-based models, which are by definition nonlinear power estimation models due to the effective nonlinear modeling capabilities of NNs, also showed high run-time estimation accuracy. Nonlinear models are by definition more complex than linear models and thus will necessarily lead to higher run-time overhead when making periodic power estimations which might be one of the reasons why fine-grained (core-level and high estimation rates) nonlinear power models have not been further investigated. In summary, to the best of this author's knowledge, no work has been published which accurately models the nonlinear relationship between a processor's performance counters and dynamic core-level power consumption at high estimation rates.

This leads to the question if nonlinear power modeling approaches for fine-grained power estimation models might improve estimation accuracy while still keeping run-time overhead low enough. If single invocations of the power model are too computationally costly, the run-time overhead at sufficiently high estimation rates for run-time power and thermal management algorithms will become cost-prohibitive in regard to power or area overhead negating any advantages of possible improvements in estimation accuracy. In Section 4.2, FFNN-based fine-grained power models will be generated and optimized towards estimation accuracy and low run-time overhead and in Section 4.3 a nonlinear transformation-based fine-grained power model is proposed and its accuracy and overhead evaluated.

2.3 Forecasting and Prediction of workload-dependent Processor States

In the following, the related works in regard to power forecasting are reviewed while focusing on methodologies which are run-time applicable, i.e. provide information on future system states and metrics during the run-time operation of a multi-/many-core processor. As per the proposed nomenclature for this thesis in regard to *estimation*, *prediction* and *forecasting*, none of the techniques in the prediction category forecast the workload characteristics in the future. All in the following discussed related works will be categorized — similarly to the power estimation related works in Section 2.2 — in regard to their:

- outputs and inputs of the forecast or prediction model,
- time resolution, i.e. forecast or prediction rate,
- spatial resolution, e.g. core-level, package level,
- model type, e.g. table-based, NN-based, SVM-based,
- applicability for unknown workloads, i.e. can the model be used for workloads it was not trained for,
- model inputs, e.g. previous power, previous performance counter values.

Machine learning provides powerful and well-established means to build prediction and forecasting models, with many works employing machine learning in multi-core systems to predict workload power/performance, while only a few of them propose to forecast the workload in the future. Forecasting problems are in general a difficult set of problems, especially for forecasts of non-periodic behavior. Due to the complexity of forecasting workload-dependent power, thermal and performance characteristics, comparatively fewer works have tackled the problem of forecasting in contrast to estimation and prediction problems despite its general relevance. To give a more comprehensive overview of forecasting of workload-dependent compute system states, some works in regard to performance, energy and thermal forecasting will also be discussed. In addition, some *prediction* methodologies will also be discussed although such works are less closely related to the overall topic of power forecasting, they are useful to gain a more comprehensive picture of the forecasting and prediction techniques used for multi-/many-core resource management.

An overview of the related works and their classification according to above characteristics is given in Table 2.2 starting with forecasting works in the upper half and with prediction works in the lower half.

Table 2.2: Overview and classification of related works on forecasting and prediction of power, thermal and performance characteristics

Related Work	Model Out	Model In	Model Type	Spatial Res.	Time Res.	Unknown Workloads
Forecasting Models						
[80]	$perf$	P, Δt	table	sys.	100 MI	no
[79]	P	PC	table	core	var.	no
[6]	P	P	AR	core	2 μs	yes
[81]	P, $perf$	PC_{GPU}	class.	GPU	var.	yes
[82]	T, $perf$	T, $perf$	ARMA	core	100 ms	yes
[83]	n_r	n_r	ARMA	cloud	1 h	yes
[84]	$perf$	PC	SVM	core	0.5 ms	yes
[85]	$perf$	PC	LSTM	core	1 ms	yes
Prediction Models						
[86]	$perf_f$	PC_{GPU}	RLS	GPU	50 ms	yes
[87]	$perf_{mig}$	PC	FFNN	core	10 ms	yes
[88]	$perf_{mig}$	PC	FFNN	core	0.25 s	yes
[89, 90]	$perf_{mig}$	PC	SVM	core	10 ms	yes
[91]	$perf_{f/th.}$	PC	Markov	core	100 ms	yes
[54]	P_f	PC	func.	core	200 ms	yes
[61]	P_{mig}	PC	func.	core	500 ms	yes

In the following sections, first an in-depth overview of related works on forecasting different processor system states is given in Section 2.3.1, followed by related works on

predicting the effect of not-yet-executed resource management decisions on the system state in Section 2.3.2 and Section 2.3.3 discussing specifically related works on power forecasting.

2.3.1 Forecasting Models

An early work in [80] proposed to classify workloads on a single-core system into program phases (very compute-bound to very memory-bound) with performance counters as input which are encoded in a so-called global phase history table. The history table encodes phase patterns which are constituted of the compute/memory boundlessness of each individual phase corresponding to 100 million executed instructions (100 MI). When a workload has been observed once, its phase pattern is stored and a repeat of the starting pattern is used to recognize the full-length phase pattern. The output of the history table forecast methodology is the forecast of the next following program phase (and its compute/memory boundlessness) from the already recognized phase pattern. These forecasts are then used for a *proactive* power management, in this case *proactive* DVFS, scheme. This work is also one of the first arguing for proactive power management for better processor performance compared to a reactive power management scheme. Although the forecasts concern performance states, this is one of the first works to introduce workload-dependent forecasts for proactive power management and is one of the fundamentals for this problem.

The aforementioned history table-based approach was re-targeted and simplified in [79] towards power-forecasting by identification of program phase patterns based solely on their active phase lengths Δt. In this work, the history table encodes average power consumption for each observed program phase. If a recurring program phase pattern is recognized, the following program phase in this pattern and its encoded program phase power-level are used as the power forecast for the full duration of the program phase. In contrast to [80], the phase duration is not a fixed number of executed instructions but depends on the changing performance counter information which is used to identify the start times and end times of the phase. The main limitation of this work is its coarse granularity in regard to the length of identified power phases — usually varying between 10 ms - 100 ms — which does not allow for modeling with any accuracy the substantial intra-phase variation of power consumption.

The main drawback of these table-based works [80, 79] is that they are designed for known applications. In contrast, this thesis tries to tackle the problem of forecasting power for unknown workloads. There have been related works which also tackle forecasting of workload characteristics for unknown workloads which are discussed in the following.

More recently, the work proposed in [6], so-called *SmartDPM*, employs a linear AR-based predictor to capture sporadic variations in the workload. Previous power consumption values as well as the current power consumption of the core are used as input to the model which forecasts the power consumption for the upcoming time period. This modeling approach is very lightweight and allows for dynamic readjustment of the model coefficients during run-time to improve forecasting accuracy with the forecasts

then being used to select appropriate Vf-levels that minimize the power consumption, i.e. it enables a proactive DVFS management. A limitation of this work which will be discussed in more detail through experimental results in Section 5.2 is the inaccuracy in forecasting rapidly changing power phases due to the very low model complexity and its resulting limitations to have any capacity for in-depth phase recognition.

A classification methodology to forecast the power consumption and performance of GPU kernels and to adapt the GPU hardware states (DVFS) to minimize overall power consumption was proposed in [81]. The inputs of the classification-based forecasting model are GPU performance counters PC_{GPU} and the outputs are forecasts of future power consumption of the executed kernel and its performance characteristics. These outputs are then used in a Model Predictive Control (MPC)-based approach for dynamic GPU power management. The forecasting model operates on individual GPU kernel level and thus the time-resolution is variable and dependent on the kernel length, i.e. there is no fixed periodicity of the power forecasts. This is a run-time model which was generated at design-time and classifies the different kernel characteristics using a forest regression algorithm. Based on these design-time classes and the identification of run-time kernels — also of unknown kernels — and their classification, the future GPU kernel power consumption values are forecast. The forecasting methodology achieves a MAPE of 25% for performance characteristics and 12% for power levels. In contrast to the work in [81], this thesis tackles the problem of CPU core-level power forecasting with a fixed periodicity.

Another Autoregressive–Moving-Average (ARMA)-based work on forecasting of future temperature T values as well as some performance values, e.g. core utilization was presented in [82] and compared to history-based, OLS-based and exponential moving average forecasting methodologies. These forecasts where then used as input to a thermal management algorithm which was thus able to operate proactively leading to far fewer on-chip thermal hotspots. In regard to the accuracy of the thermal and performance forecasts, the ARMA-based approach showed the lowest temperature error. This work further motivates the search for power forecasting methodologies where the same forecast model can be re-used for different power and thermal management algorithms. An overview for on-chip thermal estimation techniques can be found in [92] which also highlights the importance of accurate estimation methodologies for safe operation of processors.

Many forecasting techniques rely on rather lightweight models, in contrast, [84] employs a deep learning SVM to forecast the next low-throughput of the workload, in order to downscale proactively the Vf-level such states state, i.e. a targeted, proactive DVFS algorithm. The overall goal is to reduce power consumption. Inputs of the SVM model are again performance counters — which were dimensionality reduced with a PCA — and the output is the likelihood for a low-throughput state in the next time-step(s). The SVM-based approach is compared to a linear regression and a last value forecaster, i.e. if a dip is observed this is also the forecast for the future time-step, with the SVM-based approach showing the highest forecast accuracy. Interestingly, the look-ahead window, i.e. how many time-steps into the future is forecast, is investigated up to a value of 16 time-steps and the forecast accuracy diminishes with larger look-ahead windows showing

the difficulties of forecasting further into the future. In regard to the run-time overhead of such a deep learning approach, no further details were given. However, this work still motivates further research into more complex forecasting models, i.e. also the proposed complex LSTM-based power forecasting methodology in this thesis.

Another work [85], focusing on performance forecasting, proposes the use of LSTM RNNs to generate forecasting models and compares them to a Kalman filtering approach. As inputs, again performance counters are used which allow the characterization of the workloads and the output of the forecasting LSTM RNN models is the forecast Cycles Per Instruction (CPI) and total number of instruction for the upcoming time-interval. The parametrization of the LSTM RNN in regard to the input data format is closely modeled on a previous Kalman filtering approach and the overall LSTM RNN neural architecture is kept comparatively simple with a single layer and four LSTM neurons. This performance forecasting model is trained with 20,000 training samples (time steps) of different benchmarks and is embedded in a DVFS control loop optimizing each core's Energy Delay Product (EDP). Unfortunately, there is no specific discussion in this work of the forecasting accuracy/errors with the main evaluation showing improved EDP when using the LSTM RNN as forecasting model rather than a Kalman filter. However, this work using comparatively small — neural architecture-wise — LSTM RNNs motivates further research into LSTM RNN-based power forecasting methodologies.

On a higher forecasting level in [83] — for the provisioning of virtual machines in cloud environments — an ARMA-based model was proposed to predict the future workload demand, i.e the number of requests n_r for virtual machine resources as well as their specific type, e.g. gaming and web hosting. The input to this forecasting model are past requests and their specific associated type and workload demand. Although this is on a far higher abstraction level than the other works discussed in this section, it is interesting to see that AR/ARMA models are successfully used on such a cloud level to identify patterns in user needs to anticipate computational resource requirements.

2.3.2 Prediction Models

As can be seen, the amount of works on forecasting is rather limited, therefore, a selection of interesting prediction works are discussed in the following. The state-of-the-art techniques in the prediction category employ machine learning methodologies to predict how potential management decisions, such as DVFS, scheduling and task migration would affect the performance, power or thermal characteristics of — often currently executing — workloads.

For instance, the technique proposed in [86] uses a RLS algorithm to predict the change in the graphics performance $perf_f$, i.e. the frame processing time, for graphics applications if DVFS management changes the Vf-levels of the GPU. This models the sensitivity of the currently executed graphics application's performance towards changes in the operating frequency of the GPU and thus predicts how the performance would change for different operating frequencies. Based on this performance prediction, suitable Vf-level for each graphics application can be proactively selected to optimize for performance and power consumption. This work shows that even lightweight regression algorithms

are able to accurately predict GPU performance sensitivity towards frequency changes and was later on extended with an online learning methodology — so called *STAFF*, in [93]. Note that the frequency sensitivity is workload dependent but does not infer any information on how the workload characteristics itself would change in the future by itself.

In [87], an FFNN-based model is proposed which uses performance counters to predict the performance of a thread $perf_{mig}$ if it were to be migrated to another core in an S-NUCA architecture. The FFNN-based model is sufficiently lightweight to be executed, i.e. running periodic inferences, during run-time to generate performance predictions which are used in combination with power budgets and task mapping information to obtain optimal future task mappings in regard to task performance and power consumption.

Another FFNN-based model is presented in [88] to predict both power and performance of a thread observed on one core, e.g. a small core, once migrated to another core, e.g. a big core, on a heterogeneous platform. As inputs, again performance counters are used to assess the workload behavior. For the prediction itself, multiple FFNNs with a single hidden layer and 100 hidden neurons are trained for each target core architecture and with individual FFNNs distinguishing power and performance prediction. Interestingly, this low complexity FFNN architecture already achieves a high cross-platform prediction accuracy. In addition, this work also implements a power estimation methodology based on a linear power model on core-level, however, the power estimation itself is only a smaller contribution of this work and therefore not discussed in detail as other estimations works in the previous section.

SVM and linear regression models have been employed in [89], which was later extended in [90], to predict thread performance indicators, like IPC and last level cache misses, for potential core type — in this work Intel i7 cores and Intel Atom cores — changes. An SVM-based model and a linear regression model is generated for each to-be-predicted performance indicator with the SVM-based model offering consistently far better prediction accuracy with negligible overhead at a prediction rate of 10 ms.

In [91], periodic workloads like video encoding with a high dependency on the input data, i.e. frames to be encoded, and their program phases are classified into different performance classes based on performance counters. The class of the workload is predicted with a Markov (MK) model which is also updated at run-time to capture workload transitions. This classification enables a prediction of the throughput of the workload at different operating frequencies and levels of multi-threading, i.e. $perf_{f/th.}$. Using this predicted throughput information, a proposed DVFS manager in combination with scheduler is able to optimize the system for high quality of service video encoding with frame-rate guarantees.

The work of [54] was already mentioned in the previous section as it also proposed a power estimation methodology which is used together with a power prediction methodology. This power prediction methodology aims to predict the power consumption of workloads observed at a one Vf-level for different Vf-levels to guide a power management DVFS algorithm. The prediction model itself is purely analytical in form of different

linear and nonlinear functions and their parametrization derived from in-depth microarchitectural knowledge of the AMD processor design.

As discussed in Section 2.2, the work in [61] proposed both a power estimation approach as well as a cross-core power prediction approach for the heterogeneous ARM big.LITTLE multi-core architecture. The goal of prediction approach is to predict how the performance and power consumption would change when a task, mapped and observed on the small/large ARM Cortex-A7/Cortex-A15, would be migrated onto the corresponding other core architecture. The challenge is to account for the differing compute capabilities and memory architectures of the two core architectures and a linear mathematical model is derived to predict how performance and power would change due to task migration in the heterogeneous processor.

Finally, for further reading on related works on these topics, the two comprehensive literature surveys of methodologies for predicting and forecasting power, performance and temperature in [94, 95] can be recommended.

2.3.3 Power Forecasting Models

As mentioned, previously one of the aims of this thesis is to provide a methodology for *forecasting future workload-dependent power consumption values* for the use of power and thermal management algorithms for unknown, potentially non-periodic workloads. Referring back to Table 2.2, one can see that few works [80, 79, 6, 54, 61] are targeting the forecasting or prediction of power values on processors, and only a subset [80, 79, 6] of those target the forecasting of power values. All presented prediction techniques rely — at least partially — on run-time performance counter inputs for classification and prediction of their target prediction outputs and, notably, all of these works are suitable prediction of unknown/untrained workloads. Also, one can observe that there is a wide range of machine learning and NN-based techniques [87, 88, 88, 89, 90, 91] deployed for predicting the impact of potential multi-/many-core resource management decisions on workload and system states. However, for the problem of forecasting fewer works [84, 85] employ NN-based methodologies and most works are either table-based [29, 79] or AR-based [83, 82, 6]. This implies that predicting the effects of management actions in regard to a specific, previously unknown/untrained workload can be well modeled through machine learning and NN models. In contrast, for forecasting either rather simple models are used for unknown workloads or table-based models are used for known workloads, i.e. encoding previously encountered and periodic workloads. The question arises how well such simple models truly anticipate and forecast future power phase changes which will be tackled in Section 5.2.

Overall, including both the forecasting and prediction categories, the closest related works in Table 2.2 in regard to forecasting future power consumption levels on core-level are [80, 79, 6]. However, [80] is the oldest of these works, was proposed for single-core processors and has a rather coarse-grained time and spatial resolution for its power forecasts while [79] is newer, has more fine-grained time-resolution and is also table-based. In addition, [6] is very new, on core-level with high time-resolution and complements [79] well as reference model as it uses an AR-model. The two works [79, 6] forecast future

power states of workloads, irrespective of potential actions of the power and thermal management algorithms, on the CPU core-level. The main limitation of history table-based methodology [79] is that a workload has to be observed at least once before its power can be forecast for future occurrences of the same workload. A possible limitation of the AR-based methodology [6], which will be shown in detail in Section 5.2, is its very low model complexity which might not allow it to capture the dynamic of rapid power phase changes. These two works will be used as reference models for the proposed standalone power forecasting methodology enabling proactive power and thermal management.

3 Experimental Setup for Power Model Evaluation

The methodologies on run-time power estimation as well as run-time power forecasting — which will be presented in detail in Chapter 4 — are evaluated with the experimental setup as described in the following. For different models there are only minor differences in the generation and evaluation steps which will also be highlighted in the later chapters. Section 3.1 gives an overview of the simulated system and the workflow for data generation and model training and evaluation. This is followed by a more in-depth discussion of the used performance and timing simulator in Section 3.2 which also discusses the performance counters used throughout this thesis as model inputs. Section 3.3 then discusses the power simulator providing power consumption data and its limitations. Finally, in Section 3.4 the benchmarks suites used as workloads on the processors system are described.

3.1 Overview Experimental System and Workflow

A simulated 16-core multi-core system, as shown in Figure 3.1, is used for the evaluation of the proposed power estimation and power forecasting methodologies in this thesis. The multi-core system has a 2×2 tile architecture with four cores — each having an Intel Nehalem/Gainestown architecture — per tile and shared L3 caches on tile-level. This is a homogeneous multi-core system. Intra-tile communication between the four cores and the L3 cache happens through a bus interconnect. All of the tiles are inter-connected via a Network-on-Chip (NoC) for inter-tile communication and the NoC also connects to the memory controller which accesses the DRAM. The available cache memory and its distribution throughout the processor is given in Table 3.1. Note, that the power consumption of the private L1 and L2 caches will be included in the power consumption values of their respective cores.

Table 3.1: Cache architecture of the 16-core processor

Arch. Level	Type	Size	Access
Per core	L1 Cache	32 KB	private
Per core	L2 Cache	256 KB	private
Per tile	L3 Cache	8 MB	shared

The general experimental workflow, as can be seen in Figure 3.1, is the following:

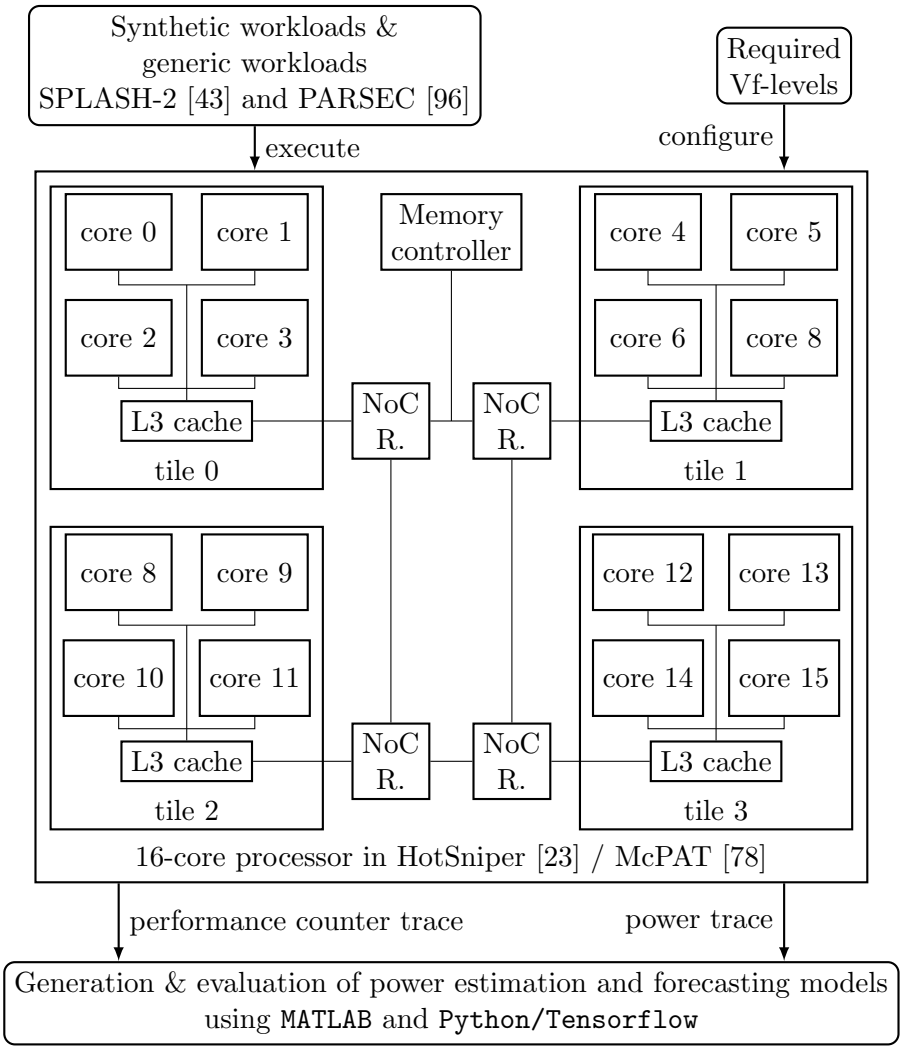


Figure 3.1: Experimental setup workflow with a simulated 16-core processor using HotSniper and McPAT for generating performance counter and power consumption data

1. The required Vf-levels for the specific experiments are configured.

2. Both synthetic workloads and the generic workloads SPLASH-2 [43], PARSEC [96] are executed on the 16-core processor.

3. The 16-core processor itself is simulated with HotSniper [23] which generates the performance counter data and which uses an integrated McPAT [78] to generate the power consumption data.

4. The traced performance counter and power data is then used in a `MATLAB` environment to generate different power estimation models and in a `Python/Tensorflow` environment to generate the power forecasting models.

5. The generated power estimation/forecasting models are evaluated on traced performance counter and power data with the specific distribution onto training data set, validation data set and holdout data set described in the respective sections of Chapter 4.

As mentioned, the experimental setup is simulation-based in contrast to many of the related works discussed in Chapter 2 which instrumented individual physical computing systems mostly on processor-level, Power Supply Unit (PSU)-level or system-level (e.g. through wall socket power meters). However, development, improvement and evaluation of different power modeling methodologies — including run-time power estimation — have also been done through simulation-based experimental setups, e.g. in [97, 67, 98, 99, 100, 101], although more rarely than through instrumentation of physical computing systems. The use of simulation-based trace data has a few distinct advantages as it offers separate information on power consumption, distinguished between dynamic, static and short power, of microarchitectural components with very high time-resolution rather than for example a single aggregated measurement at platform-level. This allows generated power models to be directly evaluated on very fine-grained level under full knowledge of the operational Vf-levels and not through the otherwise necessary, more indirect, coarse-grained levels of the physical instrumentation of the computing system/processor — as there are no integrated, fine-grained power sensors available to validate power models directly. However, using simulation-based trace data has also distinct disadvantages. Mainly, the validity of the evaluation and comparison of the different proposed power estimation/forecasting models hinges on the accuracy of the underlying simulators which is one major limitation of this experimental setup, i.e. one major underlying assumption for this thesis is that the inaccuracies of the employed simulators distort the evaluated power models the same way and do not inadvertently favor specific power models. This thesis will refer to this assumption as *neutral-distortion* assumption.

In the following sections, the specific performance and power simulators and their configuration for this thesis will be described and discussed in detail. Also independent accuracy assessments of the simulators done by other research groups as well as reasons for and against the assumption of neutral-distortion will be discussed. The performance

counters used as power model inputs and the specific power information traced from the simulator framework will be shown.

3.2 (Hot)Sniper and Performance Simulation

The focus of this section is on how the performance and timing data is generated and obtained from HotSniper [23]. Although McPAT is integrated into HotSniper, McPAT will be discussed separately in Section 3.3 due to its importance for this thesis.

HotSniper itself is an extension of the Sniper multi-core simulator [102] which adds a native integration of HotSpot [103] — a state-of-the-art thermal simulator — to the base capabilities of Sniper, hence its name *Hot*Sniper. First, the base capabilities of the Sniper multi-core simulator, which was proposed and developed by Carlson et al. at the Ghent University and at Intel ExaScience Lab, are discussed in regard to performance and timing simulation of workloads. The main idea behind the Sniper multi-core simulator is to only do interval simulations and only switch to cycle-accurate simulation when the program flow is interrupted. This is in contrast to for example *gem5* [104] — an also often used and configurable simulator of processor microarchitectures — which simulates each program cycle individually. Interval simulation is an intermediate level of abstraction between cycle-accurate simulation, e.g. gem5, and so-called *one-IPC* simulation where it is assumed that the IPC of an executed program is always one as long as there is no memory-bottleneck in the system. The Sniper multi-core simulator, as well as its derivative HotSniper, focus on accurately simulating inter-core miss events, e.g. branch and cache misses, and miss-events which concern intra-core and off-chip DRAM communication, e.g. Translation Lookaside Buffer (TLB) and Last Level Cache (LLC) misses. While such miss events which impact the program flow are cycle-accurately simulated, and thus by extension also for later power simulation, in regard to the changing run-time performance, program phases with continuous program flow are simulated by averaging the performance of the executed instructions over longer than a single-cycle intervals. Thus, the simulator achieves high accuracy for both rapidly changing program phases as well as for the average execution of an application when the program flow is continuous [105, 102].

Instruction traces are obtained by the Sniper simulator by instrumenting the application's binary, the application which should be executed on the simulated multi-core system, with the Intel *PIN tool* and then executing the application on the host system where the Sniper simulator is also running. Thus, the number and type of executed instructions is highly accurate during the simulation while the interval-based approach only changes the timings of the application progress depending on the when and where miss events impacting the processor performance would occur due to the underlying processor architecture. Further improvements in simulation accuracy were obtained by extending the interval simulations towards *instruction-window* centric simulations which more breaks down the out-of-order processing of instructions into more fine-grained out-of-order processing of *micro-ops*. A detailed description of this extension as well as accuracy validation against a physical Intel Server processor with a Nehalem/-

Gainestown core architecture can be found in [106]. The main advantage of using such an interval/instruction-window simulator in contrast to common cycle-accurate simulators is that its natively multi-threaded, with one thread representing one core, and allows the simulation of many different workloads over a large Vf-space in a reasonable time-frame.

As mentioned, HotSniper [23] integrates HotSpot [103] with the base Sniper simulator and was used due to the thus readily available thermal information of application runs for power and thermal management algorithm development with colleagues from the Karlsruhe Institute for Technology (KIT). To obtain the highest available simulation accuracy, HotSniper is configured for `detailed` simulation using Nehalem/Gainestown core architecture — as the accuracy of this architecture has been verified against a physical system. General overview of the microarchitectural components of this core architecture can also be found in Figure 4.14 in Section 4.3.3 where nonlinear performance / power effects are discussed. For the on-chip NoC in the processor architecture, the base configuration of HotSniper/Sniper for NoCs is used.

The traced performance counters are specified in Table 3.2 and were chosen based on commonly used performance counters in previous works on run-time power estimation modeling. The *C0* performance counter describes the percentage of time an individual core remained in the C0 power state, i.e. is fully operational and not clock-gated or power-gated. Values of C0 smaller than 100 imply that the core spent some time in lower, energy-saving power states. The *L2CLK* and *L3CLK* performance counters count the number of cycles lost due to cache access (read and write) misses *per individual core*. Note that all L3 cache performance counters still count the cache related events towards individual cores similarly to actual performance counters available in today's multi-core processors. This performance counter data is gathered through instrumentation of the `mcpat.py` script of the HotSniper simulator which was configured to be called with microsecond intervals in simulation time to write out the values of the described performance counters. In contrast to some common performance counter implementations which count upwards until wrap-around, this setup directly outputs the specific counts for the last intervals, i.e. the counter resets to zero after each read operation and directly starts counting again. For actual integrated or software-based implementations of power estimation or power forecasting models, the same behavior could be achieved by adding an additional register holding the previous performance counter value and subtracting it from the current performance counter value to calculate the delta count for the last interval. No performance counters regarding the NoC nor the I/O interface of the chip itself were traced due to such performance counters not being readily associated with the performance, and indirectly power consumption, of specific cores.

3.3 McPAT and Power Simulation

The well-known traditional multi-core power simulation technique is McPAT [78] which has been used by the majority of the power and thermal DSE as well as run-time management literature. As stated previously, the accuracy of the power data, especially in regard to core-level dynamic power consumption, used for the generation and evaluation

Table 3.2: Performance counters which are periodically traced from the simulated 16-core processor

Performance related information		
Processor Unit	Performance Counter	
Core	Instructions per Cycle	IPC
	# Branch Instructions	BPU
	# Floating Point Instructions	FP
	% C0 State Residency of a Core	C0
L2 private cache	# Load Instructions	L2LI
	# Store Instructions	L2SI
	# Load Misses	L2LM
	# Store Misses	L2SM
	% Cycles Lost due to Cache Misses	L2CLK
L3 shared cache	# Load Instructions	L3LI
	# Store Instructions	L3SI
	# Load Misses	L3LM
	# Store Misses	L3SM
	% Cycles Lost due to Cache Misses	L3CLK

of the various power estimation models as well as the power forecasting models is highly important for the contributions of this thesis. Also, the type and consistency of error sources for the power data has to be discussed to be able to assess if the assumption of neutral distortion is tenable. McPAT [78] is a power consumption and area simulator for multi-core and many-core processor architectures which provides dynamic power P_{dyn}, short power P_{short} and leakage power P_{leak} for technology nodes starting from 90 nm down to 22 nm. Power simulations for processing cores are fine-grained down to the wire, array, logic and clock network with detailed simulations of microarchitectural components, e.g. instruction fetch unit, execution unit and load and store unit. Caches are modeled as coherent, and the power simulation of the NoC is broken down into basic components like flit buffers, arbiters and crossbars. The clock distribution network, which commonly consumes a significant portion of power on-chip, is simulated as a separate circuit model on the global, domain and local levels.

The McPAT power simulator was, among other physical processors, validated against an Intel Xeon processor (Tulsa core architecture on 65) and there were deviations for the simulated microarchitectural components for the absolute power consumption, see Figure 3(d) in [78]. Some of these absolute power deviations are likely due to components, e.g. I/O and internal management logic, of the processor not being simulated due to missing publicly available information on their implementation. However, most importantly the relative power consumption of dynamic and static power consumption compared to total power consumption between the physical system and the simulation

results were negligibly $\leq 1\%$, e.g. simulated dynamic core power as a percentage of total simulated processor power consumption and actual dynamic core power as percentage of total physical power consumption were basically the same. This shows high *relative* accuracy for the dynamic/static core-level power consumption values of the simulator and implies that the assumption of neutral distortion of thus generated run-time power models can be valid. Basically, one has to expect the absolute values of the simulated power data to deviate from any actual physical system, however, the relative power values between dynamic/static (intra)-core power consumption are shown to be accurate. This minimizes the risk that any of the differently proposed and evaluated run-time power models are unexpectedly favored due to the usage of this simulated power data, i.e. all will be similarly distorted in regard to absolute estimation/forecast values while their relative performance compared to each other should be accurate. However, this assumption is still based on qualitative investigations and not on quantitative investigations and therefore one major limitation of this thesis.

The work in [107] also investigates the power simulation accuracy, limitations and error sources of McPAT for an IBM Power 7 CPU and highlights the importance of calibration using microarchitectural implementation details when deviating from pre-calibrated power simulation configurations like the Intel Nehalem architecture. For example, missconfigurations on the read/write ports of register files can lead to highly conservative dynamic power consumption assumptions when accessing register files. Also, dedicated logic which is assumed to be duplicated in McPAT is not duplicated in the IBM core architecture and thus leads to avoidable inaccuracies in the power simulations. Overall, there is no specific error source which would invalidate the assumption that the power data from McPAT would systematically favor any kind of — low-complexity compared to power simulations — run-time power estimation/forecasting model, i.e. all derived power models should be distorted in the same way and thus a comparison of their relative performance between each other should hold true.

The technology node parameter is set to 22 nm as this was the newest available parameterized technology node for McPAT at the time when most of the work of this thesis had been done. For future work, it would be advisable to use newer variants of McPAT like *McPAT-Calib* [108] which updates the power simulation framework for simulating a 7 nm technology node. There are also other extensions of McPAT like *WattWatcher* or *PowerTrain* which use dynamic calibration approaches against physical processors to minimize the absolute power error for different microarchitectures and thus provide very high simulation accuracy [109, 110]. Unfortunately, these techniques have not yet been closely integrated into state-of-the-art multi-core performance and timing simulators like HotSniper or base Sniper.

The simulated dynamic and static power consumption values on core-level, l3 cache level and package level are traced with the same microsecond periodicity as the performance counter values. For the power estimation, the target power estimation rate is 10 kHz so the power and performance counter data to be used as model input is averaged over 100 μs time intervals, the same as the inputs for the power forecasting models while in contrast, the outputs for the power forecasting models are averaged power consumption values over time-intervals of 1 ms and 10 ms.

3.4 PARSEC and SPLASH-II Benchmark Suites as Workloads

In the following, a short overview of the two parallel benchmark suites PARSEC [96] and SPLASH-2 [43] used in this thesis is given. Both of these benchmarks suites are integrated with HotSniper/Sniper simulator and their performance characteristics in the simulator were validated against physical processors in [102]. SPLASH-2 is the older of these benchmarks suites and covers a range of scientific, graphical and optimization workloads, some of which have been superseded by newer algorithms, and was developed when multi-processor, single-core systems were dominating the high performance computing industry. However, SPLASH-2 is still widely used in the scientific community to have a large workload coverage during experiments and due to the benchmark suite being open source and free to use in contrast to for example SPEC benchmark suites. Although SPLASH-3 [111] had been made available since this thesis work has been started, it mostly provides smaller bug fixes and reducing compiler warnings and does not provide conceptually different workloads. With SPLASH-2 being validated within HotSniper, it was decided to use SPLASH-2 rather than SPLASH-3 for the experiments in this thesis.

PARSEC is a newer, parallel and open source, free to use benchmark suite developed at the Princeton university by C. Bienia. It covers a set of well-known scientific and graphical workloads as well as emerging workloads from the domains of financial analysis, enterprise storage, animation and data mining. Emphasis has been put on high diversity in the parallelism of the workloads, i.e. data-parallel, pipeline and unstructured, their granularity and their diversity in data and memory usage. PARSEC gained very significant section in the scientific community where high performance benchmarking over widely different application domains is needed for experiments, i.e. where there is not sufficient to solely benchmark specific application domains like video encoding/decoding, physics simulation or networking applications. When this thesis work started, the follow up version of PARSEC 2.1 — PARSEC 3.0 which adds additional networked versions of the *dedup*, *ferret* and *streamcluster* benchmarks — was still in beta status and therefore, PARSEC 2.1 was used throughout this thesis for the experimental evaluations.

All the individual benchmarks used in this thesis are given in Table 3.3 and were executed with problem size of `simmedium`. Note, that the PARSEC benchmarks *vips*, *ferret* as well as SPLASH-2 benchmarks *volrend*, *water-nsquared* and *water-spatial* had to be left out due to errors with the PIN tool used by Sniper for benchmark instrumentation.

Table 3.3: Benchmarks used as generic workloads in this thesis

PARSEC 2.1 [96]	blackscholes, bodytrack, canneal, dedup, facesim, fluidanimate, freqmine, raytrace, streamcluster, swaptions, x264
SPLASH-2 [43]	barnes, cholesky, fft, fmm, lu, ocean, radiosity, radix, raytrace

The above benchmarks are used in combination of various cross-validation techniques for generating and evaluating the proposed power estimation methodologies in the following Chapter in Section 4.1 and Section 4.3, i.e. each benchmark is used — usually in different combinations of other benchmarks — either for generating the power estimation models or for evaluating their accuracy. For the NN-based power estimation and power forecasting methodologies proposed and evaluated in Section 4.2.4 and Section 5.2, a holdout data set as specified in Table 3.4 for final model evaluation is pseudorandomly chosen. The remainder of these benchmarks are then used for both training and validation of a set of FFNN and LSTM hyperparameters with no overlap of conceptually similar benchmarks between the holdout data set and training/validation data set, e.g. both the PARSEC raytrace as well as the SPLASH-2 raytrace benchmarks are put into the same set. This difference in the usage of the benchmarks and the associated performance and power data is due to the need for optimizing the hyperparameters of NNs where any data used interchangeably between optimizing the architecture and later evaluating the NN model performance would contaminate the final evaluation results.

Table 3.4: Benchmark distribution on the combined training/validation data set and on the holdout data set for evaluation of the power NN-based estimation and forecasting methodologies in Section 4.2.4 and Section 5.2, respectively

Data Set	**PARSEC**	**SPLASH-2**
Training/ validation	bodytrack, dedup, x264, streamcluster, fluidanimate, swaptions, freqmine, raytrace	fft, fmm, lu, ocean, radiosity, radiy, raytrace
Holdout	blackscholes, canneal, facesim	cholesky, barnes

4 Novel and Lightweight Power Estimation Models

This chapter introduces and evaluates the proposed run-time power estimation and power forecasting models and is organized as follows: First, Section 4.1 investigates an ICA-based linear power estimation methodology for automatically minimizing the multicollinearity of performance counter inputs. Afterwards, Section 4.2 investigates FFNN-based nonlinear power estimation methodologies to account for potential nonlinear relations between performance counters and dynamic power consumption and thus increase estimation accuracy. Following up on nonlinear power estimation methodologies, Section 4.3 investigates a nonlinear methodology using nonlinear functions for transforming performance counter inputs and thus reducing nonlinear relations to increase estimation accuracy. Finally, the proposed linear and nonlinear power estimation methodologies are compared in regard to their estimation accuracy, run-time overhead and other model characteristics in Section 4.4.

4.1 Independent Component Analysis-based Power Model

This section presents and evaluates the methodology for ICA-based power modeling [1].

Motivation and Contributions Chapter 2 showed that many related power modeling approaches rely on synthetic workloads which are tailored to specific processor microarchitectures and their sub-components. While executing such synthetic workloads, the observed package/system level power consumption can be attributed to the activity of different sub-components of the processor. This allows for a bottom-up approach of power modeling where ideally all sub-components of the processor have been activated by custom synthetic workloads and their power consumption values are then summed up towards core/processor/system power values.

However, as discussed in Section 2.2.4, solely relying on synthetic workloads to minimize multicollinearity of the performance counter inputs has multiple drawbacks in regard to: the availability of synthetic workloads for reproducing power models, their ease-of-reuse for newer processor architectures and the risk of overlooking interdependencies of sub-components when more general workloads are executed. To circumvent the limitations of relying on synthetic workloads, this thesis first proposes a methodology using standard benchmarks to derive core-level power models. One reason for using synthetic workloads is to minimize the multicollinearity of the performance counter information in regard to core sub-components. Rather than minimizing this multicollinearity by relying on synthetic workloads, the proposed methodology executes standard benchmarks and

reduces the multicollinearity of the obtained performance counter data by using ICA which minimizes the mutual information of a set of signals [112]. This methodology for multicollinearity reduction allows for the generation of core-level power model which do not rely on neither specific microarchitectural knowledge nor synthetic workloads.

This section on ICA-based power models makes the following contributions:

1. Showing that ICA is an effective method to minimize multicollinearity of multi-core activity information from general workloads.
2. Using the transformed performance counter data to generate a per core power model.
3. Demonstrating that the generated core-level models achieve similar or even better accuracy compared to state-of-the-art power models without relying on synthetic workloads and microarchitectural information.

First, the background on PCA and ICA in Section 4.1.1 is given. Afterwards, the ICA-based power model generation is introduced in Section 4.1.2 and the power model estimation is described in Section 4.1.3. The results in regard to the accuracy of the ICA-based power models as well as their dependency on changing workloads, their responsiveness to power phase changes and their run-time overhead are presented in Section 4.1.4.

4.1.1 Independent Component Analysis

A short introduction of the well-established multivariate statistical analysis methods, ICA [112] and PCA [113] used in the proposed power modeling methodology, is given in the following. Note, that this description of ICA [112] and PCA [113] is not a novel contribution of this thesis and is solely added to this section for the convenience of the reader. Only a short description of PCA is given as it is solely used for dimensionality reduction of the performance counters. In contrast, a more thorough description of ICA is given due to its usage being one of the main contributions towards the proposed power modeling approach. Both ICA and PCA are used as specific statistical transformations of data sets. An example showing the principles behind both transformations is shown in Figure 4.1 for the same data points with PCA shown in the left half and ICA shown in the right half. As a first step in a PCA, the subspace describing the greatest variance in the data set is determined, i.e. principal component 1. Afterwards, the second subspace orthogonal to the first subspace, which describes the remaining variance, is determined, i.e. principal component 2. For n-dimensional data, n subspaces and described by n principal components, which are denoted as $\hat{x}$ in this section, are determined. These principal components correspond to the eigenvectors of the covariance matrix of the original data set. Through this transformation, the subspaces of the principal components in the data set are linearly uncorrelated. Furthermore, PCA can be used for dimension reduction by removing principal components of lower variance as these describe less of the data set, i.e. the eigenvalues of the covariance matrix $\boldsymbol{K}_{xx}$ are zero or close to zero.

Similarly to PCA, ICA transforms the data set and generates independent components with maximized statistical independence by reducing higher order dependencies

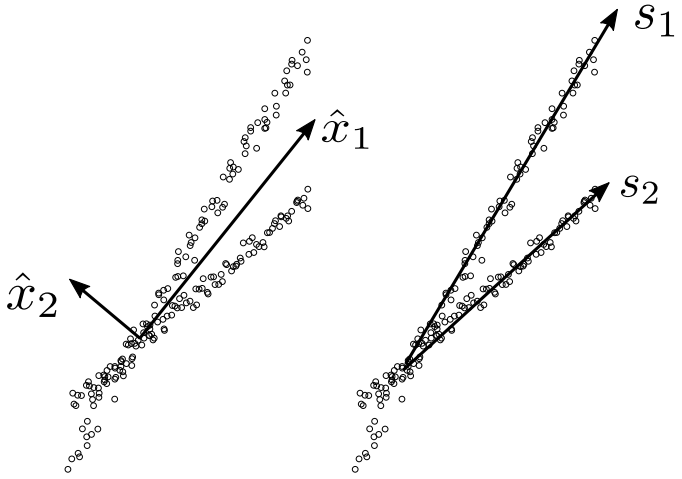


Figure 4.1: Example of scattered data analyzed via PCA on the left and ICA on the right, showing the resulting two principal components $\hat{x}_1$, $\hat{x}_2$ and the resulting two independent components s_1, s_2

in the data. As shown in Figure 4.1 on the left side, and in contrast to principal components, the independent components are not orthogonal. Also in contrast to PCA, all independent components are equally important in describing the underlying data. By removing the higher order dependencies, the multicollinearity of the different predictors or signals within the activity information of the multicore system can be minimized. To achieve this, ICA minimizes the mutual information between multivariate signals. In the following, the information theoretic concepts of an ICA are shortly described with the observed signal being denoted as $\boldsymbol{x} = (x_1, x_2, ..., x_m)^T$, the independent components as $\boldsymbol{s} = (s_1, s_2, ..., s_n)^T$ and the so-called *unmixing matrix* as $\boldsymbol{W}$ where $n \leq m$. The goal of ICA is to derive an unmixing matrix $\boldsymbol{W}$, such that

$$\boldsymbol{s} = \boldsymbol{W}\boldsymbol{x} \tag{4.1}$$

yields independent components $\boldsymbol{s}$ which are as statistically independent as possible [112].

To determine the unmixing matrix $\boldsymbol{W}$, the differential entropy H is used which can be computed for a random vector $\boldsymbol{y}$ with a given density function $f(.)$ as [112]:

$$H(\boldsymbol{y}) = -\int f(\boldsymbol{y}) log(f(\boldsymbol{y}))d\boldsymbol{y}. \tag{4.2}$$

The differential entropy H describes the amount of information generated by the continuous variable $\boldsymbol{y}$.

ICA only works on nongaussian variables as the unmixing matrix cannot be derived for Gaussian, uncorrelated variables. To determine the non-gaussianity of variables, the negentropy J is used:

$$J(\boldsymbol{y}) = H(\boldsymbol{y}_{Gauss}) - H(\boldsymbol{y}), \tag{4.3}$$

which is a measure of the random vector $\boldsymbol{y}$'s distance to normality, i.e. a Gaussian random variable $\boldsymbol{y}_{gauss}$ having the same covariance matrix as $\boldsymbol{y}$. Negentropy is always

non-negative, invariant to linear transformations and becomes zero only if the signal $\boldsymbol{y}$ is Gaussian. The more variable $\boldsymbol{y}$ is structured or "least random", the bigger the negentropy $J(\boldsymbol{y})$. In contrast, a Gaussian distribution is the least structured or "most random" distribution [112]. The mutual information $I(y_1, y_2, ..., y_m)$ of a set of m random variables is defined as

$$I(y_1, y_2, ..., y_m) = \sum_{i=1}^{m} H(y_i) - H(\boldsymbol{y}). \tag{4.4}$$

If two variables are statistically independent, their mutual information is zero. In regard to ICA, it is helpful to describe mutual information using negentropy [112]:

$$I(y_1, y_2, ..., y_m) = J(\boldsymbol{y}) - \sum_{i=1}^{m} J(y_i) + \frac{1}{2} log \frac{\prod_{i=1}^{m} c_{ii}}{det \boldsymbol{C}^y}, \tag{4.5}$$

where $\boldsymbol{C}^y$ is the covariance matrix of $\boldsymbol{y}$ and c_{ii} being its diagonal elements, i.e. the i-th variable's variance. The objective for the unmixing matrix $\boldsymbol{W}$ is then to minimize the mutual information of the independent components $\boldsymbol{s} = (s_1, s_2, ..., s_n)^T$. Thus, determining such an unmixing matrix $\boldsymbol{W}$ is a search for directions where the negentropy is maximized. Hyvarinen et al. proposed a novel way for maximizing negentropy [112]:

$$J_G(\boldsymbol{w}) = [EG(\boldsymbol{w}^T \boldsymbol{x}) - EG(v)]^2, \tag{4.6}$$

with $\boldsymbol{w}$ as an m-dimensional weight-vector, v as a standardized Gaussian random variable and G as contrast function. By maximizing J_G, one independent component can be determined. This approach is then extended to n contrast functions G where each weight-vector $\boldsymbol{w}$ is one of the rows of the unmixing matrix $\boldsymbol{W}$. Hyvärinen discussed different contrast functions G and how to choose them in [112, pp. 11-12] and proposed a fixed-point algorithm for computing the independent components. This proposed fixed-point algorithm is used throughout the remainder of this section on ICA-based power modeling.

4.1.2 ICA-based Power Model Generation

The proposed methodology in this thesis is to use ICA as a preprocessing step before generating linear power models. An overview of the methodology is shown in Figure 4.2. Rather than executing a wide range of highly specific synthetic workloads, only generic benchmarks, e.g. PARSEC [96] and SPLASH-2 [43], are executed. In parallel, the performance counters PC_i, $i = 1, ..., m$ are traced for the system as well as the package power P_{pack}. The performance counters used as activity information are shown in Section 3.2. The first preprocessing step is using a PCA to reduce the dimensionality of the activity information by removing all eigenvalues λ_i of the covariance matrix $\boldsymbol{K}_{xx}$ — where x stands for the performance counter features — for which $\lambda_i \leq \lambda_{thres}$. The value of λ_{thres} is empirically chosen by balancing the robustness of the power model against sporadic divergences with the power model's prediction accuracy. If λ_{thres} is

chosen too large, performance counters necessary to accurately describe the per-core power consumption will be removed. In contrast, when λ_{thres} is chosen too small, performance counters which do not significantly contribute to the modeling of the power consumption are left as input to the ICA. This negatively impacts the robustness of the resulting power models as independent components with marginal relation to power consumption are generated. Such marginal independent components can then lead to unstable power models where small changes in the performance counter values result in large and erroneous deviations in the power estimation.

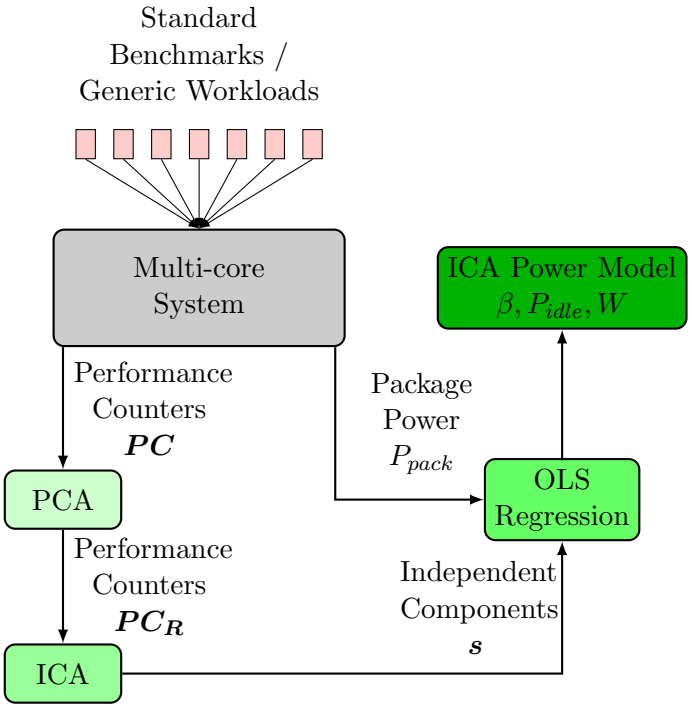


Figure 4.2: Methodology to generate a ICA-based power model for run-time power estimations [1]

Afterwards, the reduced performance counter information $PC_{R,j}$ with $j = 1, ..., n$ and $1 \leq n \leq m$, denoted as vector $\boldsymbol{PC_R}$, is transformed via an ICA into its independent components s_j where n is the number of independent components equal to the number of principal components. The unmixing matrix $\boldsymbol{W}$ of the ICA is stored to be able to use the same ICA transformation for estimating run-time power.

For generating the power model, aside of the activity information, the package power is needed for the linear regression model and modeled such that:

$$P_{pack} = \boldsymbol{s}^T \boldsymbol{\beta} + \epsilon, \tag{4.7}$$

where $\boldsymbol{\beta}$ denotes the regression coefficients and ϵ the error term. For this linear regression, solely activity information and power information is used when only a single core is active at a time. Inter-application inference on shared resources is thus ignored.

However, as the final power model is only concerned with the core-power consumption and disregards the power consumption of shared resources, limiting the data used for model generation, on time intervals when only a single core is active and generates a power response, seems reasonable. The average idle power including the uncore power consumption $P_{idle+uncore}$ of the multicore system is computed by filtering for short time frames when no cores are active and averaging the power consumption for those idle durations. A limitation of the proposed methodology is that dynamic uncore power is not explicitly modeled which can introduce estimation errors when cores are active and parts of the dynamic uncore power are counted towards core power. Finally, the core-level power model is fully described with the regression coefficients $\boldsymbol{\beta}$, static idle power including uncore power $P_{idle+uncore}$ and the unmixing matrix $\boldsymbol{W}$.

4.1.3 Estimating Core-level Dynamic Power

For run-time power estimation of per-core power for an arbitrary workload, the dimensionality-reduced — after executing the PCA — performance counter information is transformed by the previously obtained unmixing matrix $\boldsymbol{W}$ into its independent components:

$$\boldsymbol{s} = \boldsymbol{W}\boldsymbol{P}\boldsymbol{C}_{\boldsymbol{R}}. \tag{4.8}$$

Afterwards, the independent components are multiplied with the regression coefficients and the idle system power is subtracted from the intermediate power-estimate to obtain per core level power as:

$$P_{dyn} = \boldsymbol{s}^T\boldsymbol{\beta} - P_{idle+uncore} \tag{4.9}$$

Compared to other state-of-the-art methodologies, the usage of synthetic workloads and/or manual transformations of performance counter inputs for minimizing multi-collinearity are replaced with generic workloads. To achieve high accuracy while having correlated model inputs in form of performance counter (activity information), a PCA and an ICA is used for data preprocessing before generating the linear regression power model.

4.1.4 Experimental Evaluation

The aim of this section is to assess the accuracy and overhead of the proposed ICA-based power model. To do this, the experimental setup described in detail in Chapter 3 is used to generate the underlying power and performance data. First, the multicollinearity of the performance counter input data using ICA and synthetic workloads is investigated. Afterwards, run-time power estimation models using the proposed ICA methodology and using synthetic workloads are generated and their estimation accuracy evaluated and compared. Then, the dependency of the ICA power models on the number of generic training workloads used for model generation is assessed. Finally, the added run-time overhead of the ICA step is discussed.

Multicollinearity Results using ICA and Synthetic Workloads

As a preliminary investigation, the multicollinearity in the performance counter information, when using synthetic benchmarks and generic benchmarks to generate the data as well as when the synthetic and generic benchmark data is transformed via the ICA, is assessed. David Belsley proposed a method for multicollinearity diagnostics and provides a guideline on interpreting the resulting information on the magnitude of multicollinearity in [114]. Synthetic workloads, as described in [56, 36], and the generic workloads (SPLASH-2 and PARSEC benchmarks) were executed on the experimental system to generate performance counter data. Afterwards, this performance counter data is transformed with an ICA as proposed in Section 4.1.2. The built-in MATLAB function *collintest* is used to compute the so called *condition indices* of the multicore activity information. The larger the value of these condition indices, the larger the multicollinearity of the underlying data and the worse it is suited for linear regression modeling. The resulting two largest condition indices for the different performance counter data using synthetic and generic workloads and transforming them with an ICA are shown in Table 4.1.

Table 4.1: Belsley collinearity evaluation results on performance counter data obtained through synthetic and generic workloads [1]

	Condition Index 1	**Condition Index 2**
Synthetic workload	98	120
Generic workloads	$3.92 \cdot 10^{14}$	$1.06 \cdot 10^{16}$
Synthetic workloads after ICA	38	48
Generic workload after ICA	1.9	2.5

As defined in [114], condition indices above 30 often indicate multicollinearity levels which can lead to modeling errors. Note that this is a best practice threshold. One can see that the performance counter data from synthetic workloads, both before and after applying the ICA transformation, have some degree of multicollinearity above this best practice threshold. The performance counter data from generic workload leads to very high levels of multicollinearity making the data badly suited for direct regression-based linear power modeling purposes. However, after the ICA transformation, the performance counter datan is well suited for linear regression analysis and thus for generating power estimation models.

Statistical Accuracy Analysis

While the experimental setup in regard to the multi-core system and performance counters used, is the same for all proposed power and forecasting models in this thesis, the specific usage of the benchmarks for model generation and model evaluation differs. The specific benchmark usage is shown in Figure 4.3. Overall, the 20 different PARSEC [96]

and Splash-2 [43] benchmarks as specified in Section 3.4 are used as either the input to the ICA model generation or to evaluate the final model performance. The training set for model generation consists of 9 benchmarks and the evaluation set consists of 11 benchmarks with the two sets being disjunct which results in $\binom{20}{9} = 167960$ evaluations. These sizes for the training and evaluation set were chosen, to make the evaluation of the ICA methodology in regard to its capability to generate accurate power models with limited amount of observed benchmarks, on balance conservative.

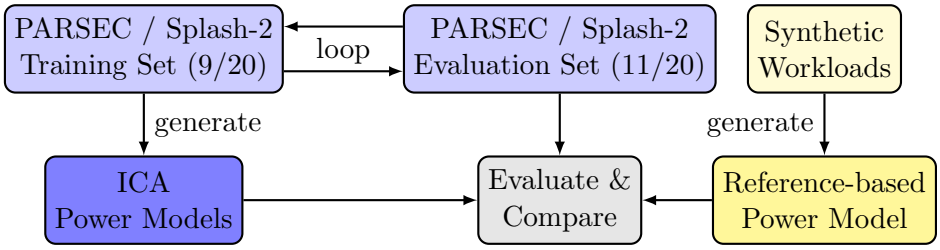


Figure 4.3: Benchmark usage for obtaining performance counter and power data for ICA-based and reference power model generation [1]

The evaluation flow then loops over all possible sets training/evaluation distributions and generates for each possible training set an ICA power model and evaluates it on the remaining benchmarks in the evaluation set. In addition, a linear power model based on a combination of reference works, mostly based on information from [36] and [56] and will in the following be referred to as *based on [36]*, using synthetic benchmarks is generated and evaluated on the same evaluation sets as the ICA power models. The final evaluation and comparison of the accuracy of ICA power models and reference-based power model is then done on core-level power values.

In the following and throughout the thesis, comparisons of error values will be given in absolute values also when comparing relative error values, e.g. relative RMSE values. For example, if two relative RMSE values $A = 50\%$ and $B = 30\%$ are given, this thesis will denote B having 20% lower relative RMSE rather than a relative decrease of $1 - \frac{30\%}{50\%} = 60\%$ percentage points. This is in contrast to the original paper [1], where relative decreases / increases were given which might lead the reader to overestimate the absolute change in error values.

To assess the accuracy of the generated ICA power models, the average RMSEs over all evaluation sets as well as instantaneous worst case error are given in Table 4.2. The comparison between the ICA power model using generic workloads and the reference-based power model using generic workloads, yields an improvement of the RMSE of 2 percentage points while the instantaneous worst case error of the ICA power model is 10 percentage points worse compared to the reference-based power model. Note, that the instantaneous worst case error is by definition outlier data, however, it can still be important for power and thermal management if a single large deviation from the actual core-level power consumption leads to an incorrect management decision. Multiple observations can be made from these results. First, the relative error values for the power estimations are overall similar to the error values in similar works, e.g. [22]. Notably,

Table 4.2: RMSE results for power models using ICA-transformation and for a reference-based power model using synthetic workloads [1]

RMSE	**Power model using synthetic workloads**	**ICA Power Model using generic workloads**
Average	0.73 W	0.55 W
Worst Case	2.1 W	2.7 W
Relative Error Values		
Average	12%	10%
Worst Case	35%	45%

this is achieved without any synthetic workload activating specific sub-components of the processor and evoking a specific sub-component power response. Secondly, the remaining average errors of both the ICA power model as well as the reference-based power model can be due to nonlinearities in the power response for specific core activities. Both modeling approaches are fully linear and assume linear relations between performance counters and core power. For example, a high number of branch mispredictions could lead to a higher than average core power consumption which would not be captured by either model. These inaccuracies, likely due to nonlinear relations, will be investigated in later sections. An additional likely source of estimation error is that dynamic uncore power is not directly modeled in these ICA models and thus implicitly, evenly divided onto each core's dynamic power estimation. Thirdly, the increase in the instantaneous worst case error for the ICA power model could be due to instability of the ICA transformation for certain processor states, i.e. the observed performance counter combinations. Finally, the training workloads used to generate the ICA power models might not cover all core activity characteristics and the resulting models would then misestimate power consumption when they encounter unknown core activities. This potential effect is investigated in the following section.

Dependency on Training Workload Coverage

As the avoidance of synthetic workloads for model generation is the main advantage of using the ICA methodology, it is useful to investigate how many generic benchmarks are needed to sufficiently cover core activity to achieve high modeling accuracy. To assess the dependency of the ICA power models on the coverage of the training workloads used to generate aforementioned models, exhaustive sets of models are generated with different number of training workloads. For this, the number of training benchmarks in the training set is varied from 1 to 19 and the accuracy of the resulting ICA power models is assessed on the evaluation test workloads. As before, the training and evaluation sets are kept disjunct. Starting with a training set size of 1, the evaluation set contains 19 workloads. For this first set, an ICA power model is generated for each of the 20 possible training sets, i.e. individual benchmarks being in the training set of size 1. In the next step, the training set size is 2 and the evaluation set size is 18 and so forth.

However, for the training set sizes 2-19 it is computationally not feasible to iterate over all possible training set / evaluation set distributions as the number of combinations increases exponentially with a maximum when both sets are of equal size. Therefore, for each step from 2 to 18, pseudorandomly 100 training workload combinations are chosen such that there is no repetition in the training / evaluation set distributions within each step. For the last step of 19 where only one workload remains in the evaluation step, again all 20 possible combinations are evaluated. With this approach, outlier distributions of the workloads onto the training and evaluation set, which could lead inadvertently to highly accurate or highly inaccurate power models, should not dominate the resulting accuracy results. The estimation errors on each evaluation run are then averaged for each step such that an average error depending on the number of used training workloads is determined. Both average RMSE for the ICA power model estimations and the standard deviation of the error values for each step, i.e. increase of training set by one, are shown in Figure 4.4. Please note, that this figure is based on an original figure first published in [1] which had an erroneous y-axis scaling due to incorrect operation of a MATLAB script for figure generation. The erroneous y-axis scaling has been fixed in this thesis. In addition, the static error line for the reference-based power model for model generation is also shown. It is a static line as that power model is generated only once using the set of synthetic workloads to generate a fixed bottom-up power model.

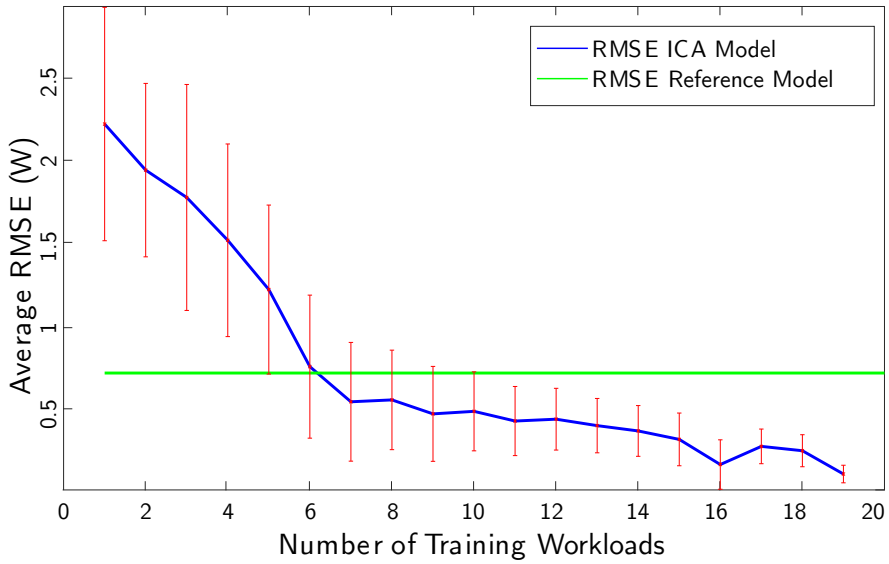


Figure 4.4: RMSE of ICA-based power models with increasing number of training workloads compared to a reference-based power model using synthetic workloads, Figure based on [1]

One can observe that the accuracy of the ICA power models rapidly increases until the training workload set size reaches 7. Afterwards, the improvement in accuracy is more gradual until the maximum of 19 training workloads in the training set is reached. There is one unexpected behavior in the accuracy graph when going from a training set

size of 16 to 17 with accuracy becoming *worse* after the increase in training size. This is probably due to an outlier at training set size 16 which, likely due to a distribution of random, above-average-performing matches between training and evaluation sets, has unexpectedly good average accuracy. However, overall the behavior shows a gradual trend towards better accuracy when increasing training set sizes, i.e. allowing the ICA power model to observe more different performance counter and power data. These results show on one hand that the ICA power model is able to improve effectively with increasing number of training workloads. On the other hand, if less than seven training workloads are used on average, this leads to worse accuracy than existing power modeling techniques using synthetic workloads.

In addition to above analysis, the power estimation accuracy for known workloads was also analyzed as follows. First, the estimation accuracy for workloads in the evaluation set was determined, then these *unknown* workloads were added to the existing training workload set, thus became *known* workloads, and the estimation accuracy for the newly known/trained set of workloads was reevaluated. In such cases, the estimation accuracy consistently improved for the previously unknown/untrained workloads. This reaffirms that the ICA power modeling approach would be able to improve its estimation accuracy in a run-time environment by observing package power and readjusting the power model.

Run-time Overhead of the ICA Unmixing Step

Finally, it is interesting to investigate the run-time overhead of the ICA power model. The power model itself is constituted of two parts which have to be executed during run-time to obtain a power estimation based on the performance counter inputs. First, the statistical transformation using ICA, i.e. multiplying the unmixing matrix $\boldsymbol{W}$ with the performance counter inputs. Second, multiplying the transformed input vector $\boldsymbol{s}$ with the regression coefficients and adding $P_{idle+uncore}$. The second part is identical to state-of-the-art linear power modeling approaches using synthetic workloads. Therefore, only the first part of the ICA power model — using the unmixing matrix $\boldsymbol{W}$ — adds additional run-time overhead compared to state-of-the-art linear power models. The unmixing matrix $\boldsymbol{W}$ is an $n \times n$ matrix with n being the PCA-reduced number of performance counters. In computational overhead terms, multiplying the unmixing matrix $\boldsymbol{W}$ with the vector $\boldsymbol{PC_R}$ to obtain $\boldsymbol{s}$ translates to n^2 Multiply-Accumulate (MAC) operations.

Within the experimental setup of this thesis, see Chapter 3, $n \leq 14$ leading to a maximum of 196 MAC operations per run-time estimate as well as an additional 6 kBit to hold the coefficients of the unmixing matrix. Although, this is an overhead compared to the minimum of 14 MAC operations needed for the linear regression part, although, it still fits easily into an on-chip micro-controller dedicated for estimating per-core power consumption. Such a dedicated micro-controller implementation is discussed in Section 4.2.4 and can support up-to 830 MAC operations at a power estimation rate of 10 kHz. However, the quadratic scaling of the ICA transformation makes the run-time application of ICA for much larger numbers significantly more difficult. This is a limitation on the general applicability of the proposed ICA approach.

Summary In this section, a methodology was presented to generate multicore power models using the statistical PCA and ICA methods with the goal of minimizing the collinearity of multi-core performance counter information when generating power models. This methodology allows the generation of core-level, linear regression power models with high accuracy without using any synthetic workloads or in-depth microarchitectural knowledge of the system. The estimation accuracy of the such generated ICA-based power models was on average better compared to a state-of-the-art approach using synthetic workloads. In worst case scenarios, when the randomly chosen generic workloads to train the ICA-based model did not exhibit a wide-range of performance and power behaviors, the ICA-based power models were performing worse than the state-of-the-art power models. However, this can effectively be countered by increasing the amount of training workloads, as one would use all available workloads for training for an actual deployment in a multi-core processor. The proposed ICA-based approach is limited in regard to its ability to generate power models on finer spatial resolution below core-level, however, such even finer-grained power information is not commonly used for run-time power and thermal management. The run-time overhead of the ICA-based power models is within a reasonable amount for an on-chip power estimating micro-controller, however, for larger numbers of performance counter inputs, it scales quadratically which can become cost-prohibitive in regard to overhead. Finally, this section opens up the question of nonlinear modeling approaches for higher accuracy power models, as ICA is a linear transformation not allowing for modeling nonlinear performance counter / power relations.

4.2 Feedforward Neural Network-based Power Model

This section presents and evaluates the methodology for FFNN-based power modeling [2, 3].

Motivation and Contributions As discussed in Chapter 2, previous [30] work has shown that the relationship between performance counters and dynamic core power can also be nonlinear at core-level, at least for large time-intervals of 1 s. The need for accurate fine-grained power models for power and thermal management, see Section 2.1.3, motivates the investigation of high estimation rates, core-level nonlinear power models which account for possible nonlinear performance counter / power relationships. Although multiple works, e.g. [71, 73, 74, 75], have already proposed using FFNNs for power/energy modeling to capture such nonlinear relationships, these were on comparatively coarse-grained server-level with time resolutions of 0.5 Hz-1 Hz. Therefore, the use of FFNNs for fine-grained power estimation and the associated run-time inference overhead are investigated in the following. In contrast to previous work, this paper focuses on generating FFNN-based power models accounting for nonlinear effects with fine-grained spatial resolution, i.e. on core-level, and time resolution, i.e. time intervals of 0.1 ms / estimation rates of 10 kHz. Extending the power estimations to the core-level and increasing the time resolution by four orders of magnitude, make a thorough investigation of the needed FFNN complexity — regarding the number of hidden layers and hidden neurons per layer — and their associated overhead for run-time inference necessary. As throughout this thesis, the power estimation models are optimized both for model accuracy and low run-time overhead with the resulting power estimations being applicable for run-time power and thermal management purposes.

This section on FFNN-based power models makes the following contributions:

1. Investigation of FFNNs for power estimation on core-level with an estimation rate of 10 kHz.

2. Optimizing the FFNN architectures, i.e. number of layers and neurons, with the objective of minimizing estimation error.

3. Additionally, optimizing the FFNN architectures with the two concurrent objectives of minimizing estimation error and minimizing run-time overhead.

4. Showing that relative estimation error decreases by 7.5% compared to a state-of-the-art linear modeling approach and by 5.5% compared to a multivariate polynomial regression model.

5. Proposing a micro-controller as well as a dedicated hardware implementation for run-time inference and discussing its overhead.

First, the FFNN architectures and hyperparameter solution space, a single-objective optimization in Section 4.2.1. Sections 4.2.2 and 4.2.3 introduce a single-objective and

multi-objective neural architecture hyperparameter optimization methodology, respectively. The results of these hyperparameter optimizations, the accuracy of the final power estimating FFNNs as well as their run-time overhead are presented in Section 4.2.4.

Notation and Definitions As discussed previously, power modeling for run-time power dynamic core-level estimation is inherently a regression problem. The desired power information is a dependent variable and the performance counters are the independent variables. First, the single-objective optimization methodology [2] for FFNN-based power modeling is described. The dynamic core-level power information is denoted as $P_{dyn,core}$ for each of the cores of the multi-core processor. The core power is estimated during run-time through the observation of n performance counters PC_i with $0 \leq i \leq n$ per core. With actual dynamic $P_{dyn,core}$ not being observable for each individual core, the following approximation for the model generation step is used when only one of the cores is active at a time:

$$P_{dyn,core} = P_{pack} - P_{idle+uncore}. \tag{4.10}$$

For physical processors, package-level power P_{pack} can be observed through instrumentation of the mainboard or the CPU, e.g. current sensors at power supply pins, and $P_{idle+uncore}$ is the idle power of the processor when the full chip is powered-on, however, where none of the cores is actively computing and the uncore including L3 caches, NoC, I/O interface etc. is also mostly idling, i.e. the uncore power consumption is counted towards $P_{idle+uncore}$. With only P_{pack} being observable, different models (FFNN, polynomial, linear) are generated for P_{pack} and $P_{idle+uncore}$ is subtracted to derive core-level power consumption. Therefore, the PC_i and P_{pack} data used for model generation is filtered to time intervals where only a single core is active at a time, similarly to the ICA-based approach, as the core-level power models are not aiming to capture shared resource power consumption. This allows to mostly capture the power response of that particular active core. However, there is an error term regarding the dynamic uncore power consumption, mostly due to data movements, could be partially counted towards the dynamic core power during model generation. With homogeneous multi-core processors being investigated in this thesis, the power models for the j-th core can be generalized to any core of the system by using the respective performance counters of those cores as model input. The error cost function for generating the subsequent power models is then:

$$P_{pack,error} = |P_{pack,act} - P_{pack,est}| \tag{4.11}$$

where the subscripts $_{est}$ and $_{act}$ indicate estimated power and actual observed power, respectively.

4.2.1 FFNN Architectures and Hyperparameter Solution Space

There exists a multitude of NN architectures, e.g. FFNN, Elman, LSTM, for modeling and estimating nonlinear functions and systems. With most fine-grained power models using linear regression models and the limited computational resources realistically available for run-time power estimation, the following analysis is kept to comparatively

simple NN architectures. Overall, the goal is to achieve higher estimation accuracies than with purely linear modeling techniques while adding as little additional modeling complexity and run-time overhead as possible. For this reason, well-known feedforward networks are chosen which can theoretically model any nonlinear function according to the universal approximation theorem [115]. Similar to previous works, no delays are used on the PC_i inputs, i.e. the generation of AR models is avoided. This is due to the assumption that the performance counter inputs, which effectively are used to approximate the switching factor α, are time-invariant over the scale of a single power estimation. In other words, previous performance counter changes — in contrast to currently changed performance counter values — do not hold additional information about the current power consumption in the current power estimation time step. While linear regression models are at risk of underfitting the underlying dynamic power relationship, FFNNs are at risk of both underfitting and overfitting the power/performance relationship. With a finite amount of training data, FFNNs of sufficient size and thus can fit each data point perfectly, i.e. memorize the data, while not actually learning the underlying relationship. In that case, the PC_i data is overfitted and the estimation errors on P_{core} for untrained/novel PC_i input data could be significant. Therefore, careful consideration has to be taken regarding the chosen hyperparameters of the FFNN which are distinguished between *algorithm* hyperparameters (learning related) and *model* hyperparameters (architecture-related). For the algorithm hyperparameters, a multitude of networks for dynamic power estimation are trained and both the resulting power estimation accuracy as well as the needed training time are compared. This heuristic approach yields the conjugate gradient backpropagation with Polak-Ribière updates providing the best training time/ estimation accuracy trade-off. As stop conditions for training the FFNNs, the following algorithm hyperparameters were set:

- stop after 1000 training epochs OR,
- an MSE below 1% on the training data OR,
- a minimum performance gradient of $1 \cdot 10^{-5}$ OR,
- five subsequent failed validation tests, i.e. additional training leads to higher estimation errors on the validation data.

After sweeping over a set of different activation functions and comparing the resulting estimation accuracy, *tanh* was chosen as activation function for all hidden neurons. The question of how to determine the optimal FFNN model hyperparameters is still an ongoing topic of research, therefore, best practices for hyperparametrization were followed. As a first step, the model hyperparameters have to be confined which is described in detail in the following.

Single-objective Hyperparameter Solution Space The goal of the single-objective optimization methodology is to find FFNN architectures which effectively minimize the power estimation error for untrained performance counter and power data, i.e. are well-suited to avoid underfitting and overfitting. The run-time overhead is then simply a

direct consequence of the architecture chosen for it having the lowest estimation error. The proposed single-objective methodology exhaustively searches through a constrained solution space of FFNN architectures. Therefore, the solution space of FFNN architectures should be both representative and well-constrained such that the necessary training time falls within a given computational limitation for training the FFNN models.

In regard to the number of hidden layers and hidden neurons per layer, the solutions space is aligned with the related work for coarse-grained power models for servers/datacenters. The number of hidden layers and hidden neuron hyperparameters are confined as shown in Figure 4.5.

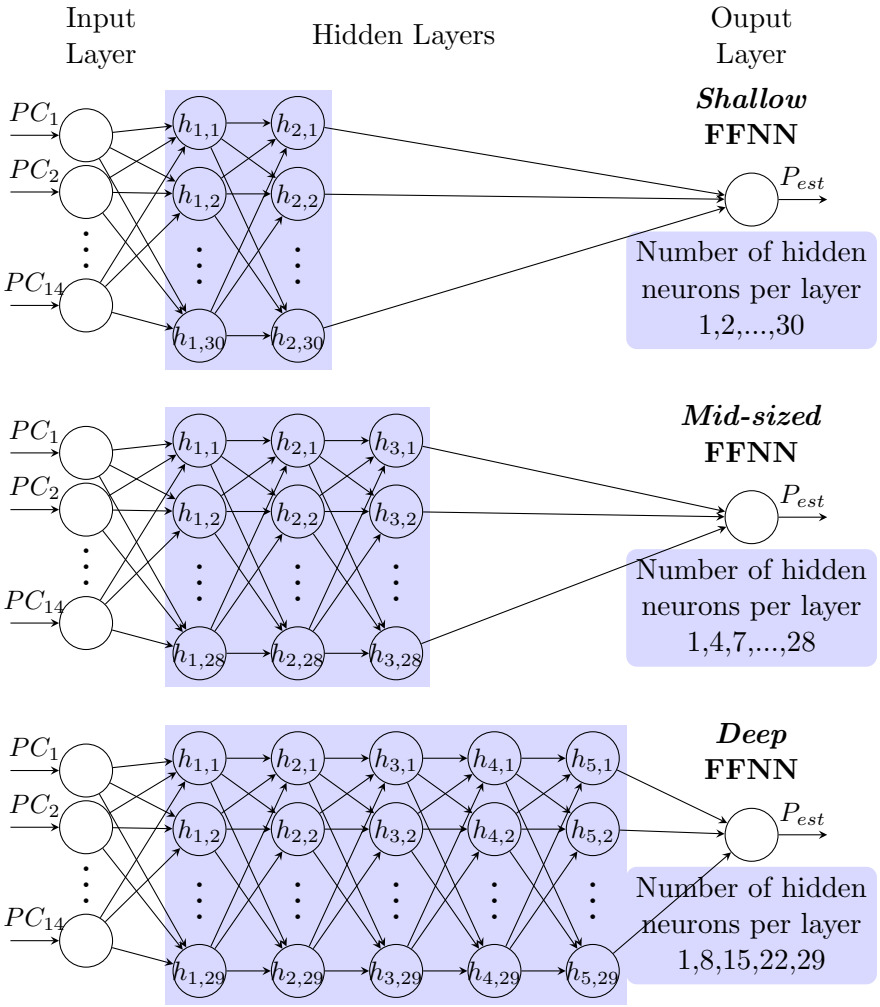


Figure 4.5: Overview of the ANN architectures investigated; the blue shaded rectangles indicate the ability to parameterize the number of hidden neurons per layer, i.e. from at least one hidden neuron per layer up to the given maximum number [2]

Overall, there are three different layer sizes explored during the single-objective optimization. The smallest is a two-layered *shallow* network with 1-30 neurons per hidden

layer. Note, that all possible combinations of the number of hidden neurons per layer are explored, i.e. 900 differently parameterized two-layered FFNNs are the solution space for the two-layered network architectures. Furthermore, a *mid-sized* network with three hidden layers is part of the overall solution space. Here, the number of neurons per layer can be any number of $1, 4, 7, 10, 13, 16, 19, 22, 25, 28$ spanning a solution space of 1000. Finally, a *deep* network with fibe hidden layers is explored where the number of neurons per layer can be $1, 8, 15, 22, 29$.

The number of neurons per layer is constrained through a more coarse-grained stepping of the possible number of neurons per layer for the *mid-sized* and *deep* networks. This is done to keep the amount of training time on a reasonable level. Although adding layers and increasing the number of neurons per layer increases the risk of overfitting, it also decreases the risk of underfitting due to an undersized FFNN.

Overall, the three different layer sizes and possible neurons-per-layer of the FFNN neural architectures constitutes $30^2 + 10^3 + 5^5 = 5025$ different FFNN architectures. These 5025 FFNN architectures are the overall solution space for the single-objective optimization. This solution space is well-constrained in regard to training time due to skipping four as well as higher-layered FFNN architectures and skipping neuron distributions within the three-layered and five-layered architectures. However, the downside is that possible more optimal architectures are missed during the optimization. In the following, the extension of the single-objective optimization methodology is presented, beginning with the extended multi-objective solution space.

Multi-objective Hyperparameter Solution Space The goal for the proposed multi-objective optimization methodology is to find neural architectures minimizing both the power estimation error as well as the run-time overhead. To find solutions simultaneously optimal for both objectives, a larger solution space has to be explored than for the single-objective optimization as the 5025 possible solutions described previously would likely not contain architectures optimal in both objectives. Due to the needed increase in the solution space, an exhaustive optimization of the FFNN architectures is cost-prohibitive in regard to training time. Therefore, a heuristic architectural optimization methodology is proposed and described later in this section.

The solution space is constrained to FFNN architectures with two, three, four or five hidden layers with up to 30 neurons per layer without any intermediate, i.e. skipping, constraints on the number of neurons per layer. Therefore, the solution space grows $30^2+30^3+30^4+30^5 = 25,137,900$ different neural architectures which is four magnitudes larger than for the single-objective optimization. With the multi-objective approach being an extension of the single-objective approach, it was possible to use the results of the single-objective optimization to determine that optimal solutions will likely not lie outside of these constraints.

In the following, both the proposed single-objective as well as the multi-objective methodologies to find well-performing FFNN architectures for run-time power estimation will be described in detail.

4.2.2 Single-Objective Neural Architecture Hyperparameter Optimization

The methodologies for the hyperparametrization of the model parameters and for the final training and computation of expected estimation accuracy are shown in Figure 4.6. At two stages, 10-fold cross validations are used to obtain statistically reliable interme-

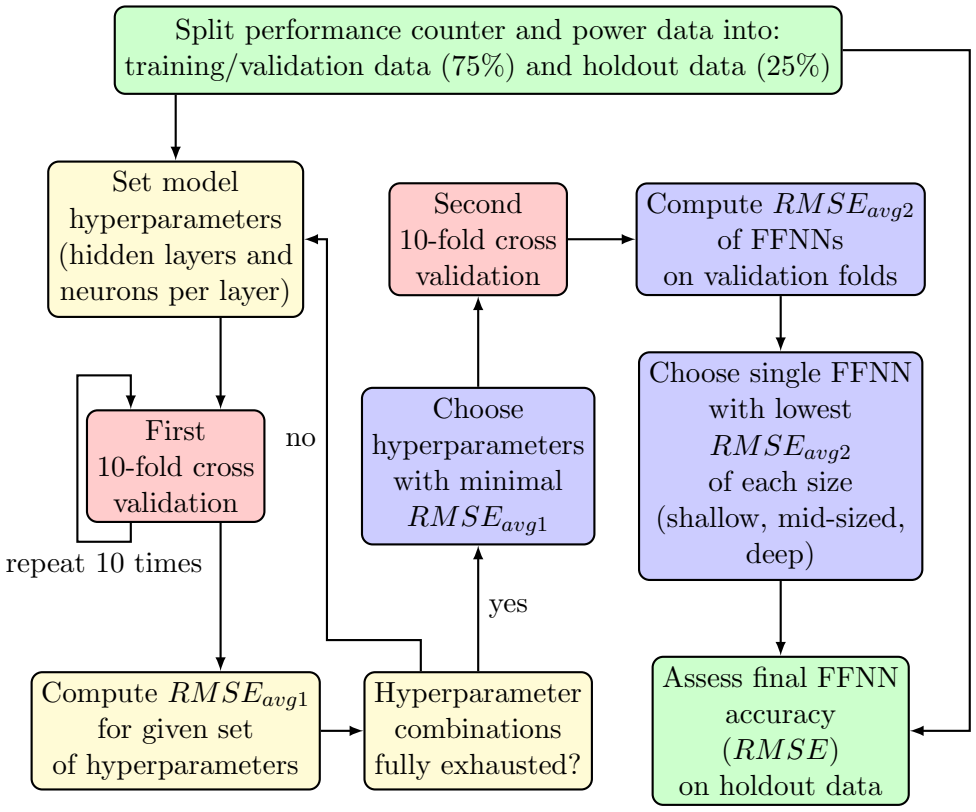


Figure 4.6: Flowchart of model hyperparametrization using a first 10-fold cross validation, final FFNN generation using a second 10-fold cross validation and the final power estimating FFNN estimation accuracy assessment on holdout data [2]

diate accuracy results, i.e. to avoid random accuracy outliers for random architectures to negatively steer the end result of the architecture search. These 10-fold cross validations are used at the model hyperparametrization stage as well as the final training stage where the final FFNNs of each layer-size (*shallow*, *mid-sized*, *deep*) are trained for actual deployment, i.e. in this thesis for holdout data accuracy assessment.

For the overall methodology, at first the traced PC_i and P_{pack} data is partitioned into a training/validation data set (75%) and a holdout data set (25%) such that the holdout data set contains benchmarks from different benchmark suites covering diverse performance / power behaviors. The holdout data set is neither used for optimizing the hyperparameters nor for training the three final FFNNs. It can thus be used to determine the actual performance of these FFNNs for data they have not been directly trained for or indirectly been steered towards during the hyperparametrization.

For the hyperparametrization itself, all possible hidden neuron per layer combinations for each FFNN layer-size (*shallow*, *mid-sized*, *deep*) are iterated over and the first cross validation loop is executed. In this loop, the training data set is further partitioned into 10 folds and each fold is used once for validation with the remaining folds being used as consecutive training data for training the FFNN based on the architecture associated with that specific loop iteration. This step is repeated for each fold ten times to produce statistically significant results and to be able to remove outliers, i.e. diverging FFNNs. Thus, overall a 100 different FFNNs of the same architecture are generated for each possible model hyperparameter combination. After a full hyperparametrization run, the estimation error on the validation data is averaged for each hyperparameter / neural architecture over all folds. Then the hidden neuron parametrizations for each FFNN layer-size with the lowest average RMSE are chosen under the assumption that these neuron parametrizations provide the best general fit for the given performance / power data. The benefit of the repeated 10-fold cross validations lies in the robustness of the average RMSE for the different hyperparameters and thus in choosing with high confidence good hyperparameters for generating the final FFNNs.

The hidden neuron parametrizations are then used in the second 10-fold cross validation step. In this step, an FFNN is trained for each training/validation fold combination, i.e. 10 FFNNs for each FFNN layer-size (*shallow*, *mid-sized* and *deep*) and those FFNNs which performed best on their corresponding validation data are chosen for final evaluation. The second cross validation is used to minimize the risk of selecting an overfitting-FFNN from the first cross validation where 100 different FFNNs were generated for each neuron parametrization. The risk of, the FFNN with the highest accuracy on their respective validation fold, overfitting is higher when 10 such FFNNs are available to choose from, rather than just one. In the final step, the three chosen FFNNs of layer-size *shallow*, *mid-sized* and *deep* are evaluated on the holdout data to assess their potential dynamic power estimation performance in a potential deployment environment.

4.2.3 Multi-Objective Neural Architecture Hyperparameter Optimization

Multi-objective optimizations of neural network architectures based on NSGA-II [116] have been proposed in several works [117, 118, 119], usually for image classification tasks. To the best of this author's knowledge, this work is the first to propose using NSGA-II to optimize FFNN neural architectures for multi-core run-time power estimation models. For the multi-objective optimization of the neural hyperparameters for the proposed power modeling FFNNs, NSGA-II [116] with Simulated Binary Crossover (SBX) [120] and polynomial mutation [121] are adapted. As the basis for the crossover operation SBX [120] was chosen, as it enables gradual changes close to the chosen parent neural architectures. By this, the solution space is searched by the crossover operation with higher probability in the vicinity of existing solutions and with lower probability in larger distances. To ensure diversity of the population and to avoid being stuck in a local minima, polynomial mutation was chosen. In addition, a comparatively large percentage of the offspring populations are generated by this mutation operator.

The goal is to find neural architectures minimizing the modeling error while also minimizing the run-time inference overhead. In addition, the aim is to make our methodology generic in regard to the neural architecture solution space, especially the number of layers. This has the advantage, that neural architectures with different number of layers can be searched in a single optimization run allowing for faster convergence towards more optimal architectures within the same training time. The disadvantage of this approach is that in case of small population numbers, some layer values might be excluded too early from the population without the chance of getting back into the population, i.e. the optimization of the neural architecture gets stuck in a local optimum without the chance of escaping it.

An explicit encoding of the neural architecture is proposed. It contains the number of layers and the number of neurons per layer as illustrated in Table 4.3. This encoding also contains the used performance metrics as shown in the table. The performance metrics are each neural architecture individual's modeling error (RMSE) and its run-time overhead counted in MAC operations. The final dense FFNN layer after the last hidden layer is not explicitly encoded as its specification is fully derived from the last non-zero hidden layer. Thus, it can not be influenced by the optimization and encoding it explicitly is superfluous. In the following, a population size of 50 is used to keep sufficiently large numbers of individual neural architectures in the population while also keeping the necessary compute time for each generational step in a reasonable range. With more computational resources available, a larger population size could be set to increase population diversity throughout the generations. Also an archive of all populations generated over the different generations is kept and cross-referenced with the currently by the crossover and mutation operations generated neural architecture individuals. This is done to avoid computationally costly re-training and re-evaluation of the same neural architecture in the case that it is generated multiple times over the generations by crossover or mutation. An overview of the multi-objective methodology

Table 4.3: Example of the encoding of individual solutions within a population including the objective performance metrics RMSE and run-time overhead and the NSGA-II population-relative performance metrics [3]

Architecture Encoding		Objective Performance		Relative Performance NSGA-II	
Number of Layers	Neuron Distribution	RMSE	Overhead (MACs)	Rank	Crowding Distance
3	6-4-17-0-0	0.55	389	1	Inf
2	25-16-0-0-0	0.45	962	1	0.25
...	...	...	...	...	...

generating and optimizing the neural architecture population is shown in Figure 4.7.

First, the performance counter data is split the same way as previously into training/-validation data as well as holdout data. Afterward, an initial population is generated with a random number of layers and neurons per layer for each individual constrained

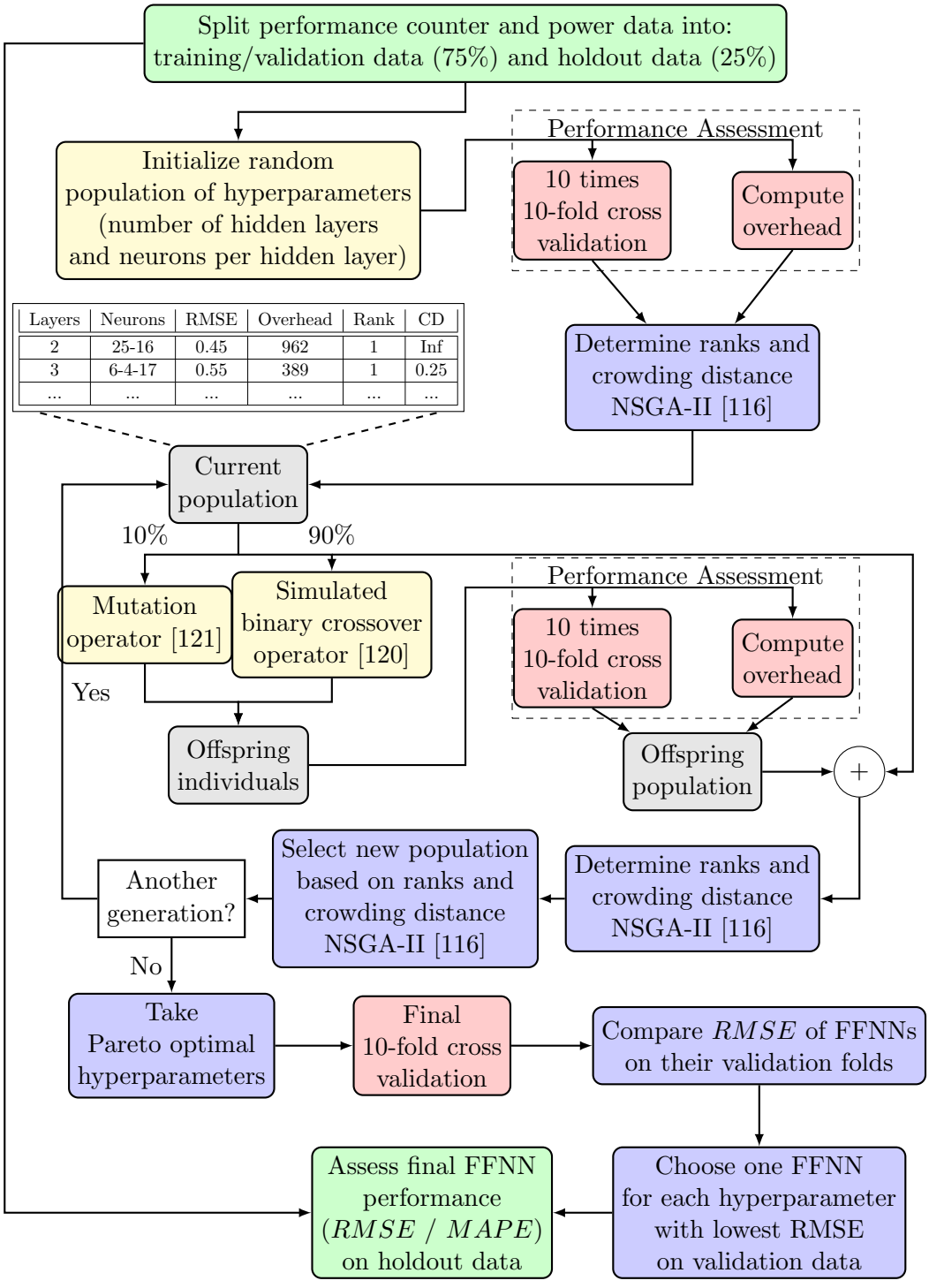


Figure 4.7: Flowchart of the multi-objective FFNN optimization of the hyperparameters using NSGA-II, 10-fold cross validations and a final FFNN power estimation accuracy assessment on the holdout data [3]

within the overall limits on number of layers and neurons per layer. The random population of neural architectures is then assessed with the same repeated 10-fold cross validation as in the previous section to get consistent estimation errors, i.e. RMSE, for each neural architecture dependent mostly on the underlying architecture and not on random deviations. In addition, the run-time overhead is calculated in MAC operations for a single inference of the neural network architecture generating a power estimate. Based on the error and overhead values, the individual ranks and crowding distance is computed using NSGA-II [116]. With this, the initial starting population is complete and functions as the current population.

In the next step, the proposed genetic algorithm optimizes the population over a multitude of generations by applying a mutation and a crossover operation, both discussed in detail in the next paragraph. Overall, 50 new neural architectures, i.e. the offspring population, are generated with 10% (5) neural architectures generated by the mutation operation and 90% (45) generated by the crossover operation. The decision on this distribution of the mutation/crossover likelihood is motivated in trying to ensure sufficiently high randomness in the offspring population throughout the optimization while also having sufficiently high convergence towards improved neural architectures. In case a newly generated neural architecture has already been evaluated, i.e. is in the generational archive, we reapply the crossover or mutation operation to get a novel neural architecture while ensuring the above mutation/crossover distribution.

Crossover and Mutation Operation The crossover operation operating on the neural architectures is shown in Algorithm 4.2.3. Note, that the two parent architectures used for a single crossover operation are chosen by binary tournament from the current population using the crowded distance operator as described in NSGA-II [116]. The parameter regarding the probability distribution is set as 20 resulting in a wide probability distribution for the neuron values.

With the number of layers being an explicit part of the solution space and thus optimization, the crossover operator has to differentiate two cases. In the first case, both parents have the same number of layers allowing to simply use that as the number of layers for the two child architectures to be generated. In the second case, both parents have different number layers and we designate randomly, with equal probability, one of the parents as the *dominant* parent. The dominant parent's layer number is then set for both child architectures. If the dominant parent architecture has *fewer* layers than the other parent architecture, the crossover of the neuron values is only executed up to the dominant (smaller) layer number generating two children architectures. If the dominant parent architecture has *more* layers than the other parent architecture, the crossover of the neuron values is again only executed for the neuron values up to the now non-dominant (smaller) layer number and the dominant parent architecture's neuron values of the remaining layers is copied for the two children architectures. Finally, the real-valued neuron values of the children architectures are rounded to the closest integer values and in case the neuron limits (1 or 30) are underflown or overflown, the neuron values are set to the respective neuron limits.

Algorithm 1: Crossover operation for optimizing the neural architecture hyperparameters [3]

Input : 2 parent architectures p_1 and p_2 chosen by binary tournament
Output: 2 child architectures c_1 and c_2
if *both parents have same number of layers* $n_1 == n_2$ **then**
  foreach *layer* $i = 1 : n_1$ **do**
    Generate real-valued neuron distributions for $c_{1,i}$ and $c_{2,i}$ using [120]
  end
else
  Randomly choose dominant parent from p_1 and p_2 **if** *dominant parent* p_d *has more layers* n_d *than other parent* n_o **then**
    foreach *layer* $i = 1 : n_o$ **do**
      Generate real-valued neuron distributions for $c_{1,i}$ and $c_{2,i}$ using [120]
    end
    foreach *layer* $i = n_o + 1 : n_d$ **do**
      $c_{1,i} = p_{d,i}$ and $c_{2,i} = p_{d,i}$
    end
  else
    foreach *layer* $i = 1 : n_o$ **do**
      Generate real-valued neuron distributions for $c_{1,i}$ and $c_{2,i}$ using [120]
    end
  end
end
Round neuron values of to $c_{1,i}$ and $c_{2,i}$ to closest integer values
if *a neuron value exceeds min/max neuron limits* **then**
  Set neuron values to limit values

For the mutation operation, random individual neural architectures are chosen from the current population and the number of neurons per layer mutated using polynomial mutation with the same probability parameter set to 20 as for the crossover operation [121]. Neuron values are increased and decreased with equal probability and the resulting neuron values rounded to the nearest integer value, with the value saturating in case it exceeds the neuron limit upwards or set to one in case of values below 1.

The performance of the offspring population in regard to modeling error and overhead is then assessed and the individual ranks and crowding distance are determined. After both objectives have been evaluated, the current and the offspring population are combined and their relative performance, i.e. their relative ranks and crowding distance, is computed according to NSGA-II. In the final step of a generation, the best individuals — according to ranks and crowding distance — are selected for the next generation.

In this proposed methodology, the only stopping criteria is a maximum number of generations limit as the computational requirements for training the FFNNs are the crucial

limit for the evaluation in this thesis. However, other stopping criteria could be sensible in case of larger computational resources. For example, a convergence limit when the population is not adding new individuals from the offspring population to the new population generation, might be advantageous to not waste larger computational resources which allow for far larger maximum number of generations. After reaching the maximum number of generations — or other possible stopping criteria — the performance of the individual neural architectures of the final generation in the Pareto front is evaluated on the holdout data. For this last step, the approach is similar to the single-objective methodology with FFNNs generated for the different validation folds and the ones with the lowest RMSE chosen for assessment on the holdout data with the only difference that there can be multiple neural architectures in the Pareto front and thus multiple final FFNNs.

4.2.4 Experimental Evaluation

The results of the proposed single-objective and multi-objective optimizations for generating power estimating FFNNs are shown and discussed in the following. Again, the experimental setup described in detail in Chapter 3 is used to generate the underlying power and performance data. First, the reference power models used for comparison with power estimating FFNNs are introduced. Afterwards, the resulting FFNN architectures from the optimizations and their run-time inference overhead for a micro-controller-based approach and custom logic approach are shown. Finally, the accuracy results on the holdout data set for the different power models are presented and discussed.

Reference Power Models

The estimation accuracy of the final FFNNs are compared with a state-of-the-art linear approach based on a proposal by Bertran et.al. [36] and a polynomial regression model based on proposal by McCullough et al. [30]. For the linear model, a set of microbenchmarks and the PARSEC/SPLASH-2 benchmarks are executed on the multi-core system, P_{pack} and PC_i are traced and the results combined into a core-level linear regression model. Although [30] argues that a polynomial regression model was not able to accurately capture the nonlinear power relationships; possibly due to overfitting. This thesis uses a polynomial model to test the hypotheses that the available performance/power data could potentially be described well through polynomial regression and therefore obviating the need for more complex NN methodologies. To generate the polynomial regression model, a similar methodology as described in Section 4.2.1 for FFNNs is used, and described in more detail in Section 4.3.1. First, repeated 10-fold cross validations are executed to determine the best polynomial order and to then generate the final polynomial regression model through another 10-fold cross validation choosing the polynomial model with the highest accuracy on its respective validation fold. The maximum polynomial orders explored for the independent PC_i inputs were one to six. It was found that a maximum polynomial order of two offered the best average estimation performance on

Table 4.4: FFNN architectures found by the single-objective optimization methodology with average RMSE on the randomized validation data [2]

FFNN	Number of Hidden Layers	Neurons per Hidden Layer	Average RMSE	Relative Error
Shallow	2	14-29	0.31 W	5%
Mid-sized	3	25-28-25	0.36 W	6%
Deep	5	22-08-22-15-15	0.33 W	6%

the validation data. These two reference-based linear and polynomial models are then used in the final accuracy comparison of the power estimating models.

Single-Objective Hidden Neuron Architecture Results

The single-objective optimization solely optimizes the estimation accuracy for the three fixed layer-sizes *shallow*, *mid-sized* and *deep*. In the following, the model hyperparameters of the three hidden layer-sizes minimizing the RMSE on the validation data from the 10-fold cross validations are investigated. For these three different FFNN layer-sizes, the hidden neuron parametrizations with the lowest average RMSE are shown in Table 4.4. In addition, the relative error compared to the average core power is computed as $\frac{RMSE}{P_{avg}}$.

The first observation is that the *shallow* FFNNs with two hidden layers and 14 neurons in the first hidden layer and 29 neurons in the second hidden layer offer on average the best performance on the validation data. Both the *mid-sized* and *deep* FFNN offer worse performance than the *shallow* FFNN on the validation folds. Notably, all FFNNs steer towards the previously set upper limit on the number of neurons per layer of 30 within the available solution space. However, for the *mid-sized* and *deep* FFNNs these higher neuron counts can also be due to the coarse granularity of the solution space for these layer-sizes. Smaller FFNN architectures with specific neuron-distributions might be outside of the solution space but still offer similar or better accuracy for FFNNs with three or four layers. This is also a motivation to avoid course-grained solution spaces for such power estimating FFNNs where run-time inference overhead is critical and any avoidable architectural complexity — which roughly translates to run-time overhead, should be avoided.

Multi-Objective Hidden Neuron Architecture Results

With the proposed optimization being a multi-objective metaheuristic, first its convergence properties towards good FFNN architectures over time are investigated. In Figure 4.8, the average RMSE and overhead values of the neural architectures are shown over 50 generations of genetic optimization. Also, the smallest RMSE and overhead values of the Pareto-optimal (rank 1) solutions in the population are shown. One can observe that the average overhead values are steadily decreasing while the average RMSE

values tend to fluctuate over the generations. After further investigating the population composition from generation to generation, it was found that small numbers of neural architectures with above-average RMSE and very small overheads are introduced and kept in the population on the lower ranks. For example, neural architectures with only one to three neurons on a single layer drastically decrease overhead while simultaneously increasing the modeling error and thus stay in the population as they are still optimal on the overhead objective while underperforming the error objective. This leads to average RMSE values fluctuating with due to such individual solution outliers. However, this behavior is not in itself problematic as an overall increase in population diversity is seen when inspecting manually the populations over the different generations. This higher population diversity can lead to more optimal solutions further down the generational timeline. As was to be expected for the solutions in the actual Pareto front of the population, the minimal values observed for either the RMSE or the overhead are both decreasing steadily. This thesis concludes that our heuristic multi-objective optimization successfully optimizes the initial random neural architectures.

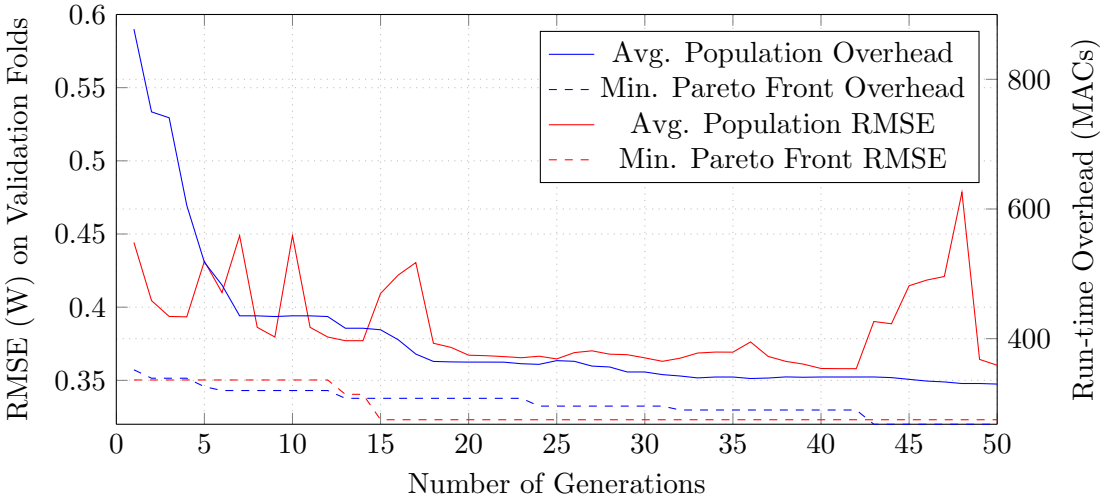


Figure 4.8: Average population performance over 50 generations as well as lowest overhead and RMSE values of the Pareto optimal solutions [3]

The Pareto optimal solutions obtained after 50 generations are given in Table 4.5. In addition, each solutions corresponding RMSE values on the validation folds and their run-time overhead are specified. All four solutions have quite similar neuron architectures with small trade-offs between overhead and modeling error. This is not a large diversity in the Pareto optimal front of the population and could mean that either the overall optimization converged towards a local optimum quite early in the optimization or that these are solutions close to a global optimum for a power estimating FFNN architecture with the given training and validation data. The convergence towards a global optimum cannot be proven due to necessity of sweeping the whole solution space to achieve a provable conclusion. However, within the lower ranks of the final population

only neuron architectures with different layer numbers and significantly different neuron distributions were found. This at least lowers the risk that the arising solutions where only found within a local optimum of the solution space. When comparing these architectures with the results of the single-objective optimization methodology, as shown in Table 4.4, one can see that the overhead values were successfully minimized. However, the power estimation errors are slightly larger for the heuristically optimized architectures. In conclusion, the proposed methodology is at least able to converge towards optima in both the accuracy as well as the run-time inference overhead objective.

Interestingly, the heuristic methodology has not found the same or similar neural architectures in regard to lowest RMSE as the exhaustive single-objective methodology. However, this is not unexpected as the solution space for the heuristic methodology is four magnitudes larger than for the far more constrained exhaustive optimization methodology. In regard to compute/training time for the optimization, the heuristic optimization evaluated 2500 different neural architectures, while the single-objective optimization evaluated 5025 architectures. This means that the multi-objective optimization found solutions with similar accuracy on the training/validation data and far lower run-time overhead within half of the training time.

Table 4.5: FFNN architectures found by the multi-objective optimization methodology with average RMSE on the randomized validation data [3]

FFNN	Number of Hidden Layers	Neurons per Hidden Layer	Average RMSE	Relative Error	Run-time Overhead
Multi-1	3	5-4-8	0.32 W	5%	326 MACs
Multi-2	3	3-4-8	0.34 W	6%	290 MACs
Multi-3	3	3-2-8	0.37 W	6%	268 MACs
Multi-4	3	4-4-8	0.34 W	6%	308 MACs

Run-time Inference Overhead for an on-chip Micro-controller Implementation

In the following, the computational and memory overhead of run-time inference of the FFNNs for producing a single power estimation is assessed. This run-time overhead is critical in regard to the viability of fine-grained FFNN-based power models. Due to the fact that the power models cannot require any substantial amount of on-chip area or computational resources and the estimation rates of 10 kHz, the computational complexity of a single run-time inference has to be marginal to make the models viable.

This thesis assesses run-time overhead always in regard to the necessary number of MAC operations and the required memory; in the case of FFNN-based power models the memory needed to store the 32-bit neuron weights. An overview of both the compute overhead and memory overhead is given in Table 4.6 for all the different FFNN architectures generated through both single-objective and multi-objective optimization. The number of necessary MAC operations is derived from the needed input computations and the computations in the hidden layers and the output layer.

Table 4.6: Necessary computations and memory for a single power estimation, i.e. FFNN power model inference [2, 3]

Model	Number of MAC Operations	Memory in kBit
Single-Objective Optimization		
Shallow FFNN	827	20
Mid-sized FFNN	1971	56
Deep FFNN	1426	39
Multi-Objective Optimization		
Multi-1 FFNN	326	10
Multi-2 FFNN	290	9
Multi-3 FFNN	268	8
Multi-4 FFNN	308	10
Reference-based		
Linear Model [36]	14	0.5
Poly. Model	36	0.8

Compared to the linear regression model, the *shallow* FFNN needs approximately 60 times more MAC operations and 40 times more memory The *multi-x* FFNNs, which were optimized for both accuracy and overhead simultaneously, need on average 21 times more MAC operations and 20 times more memory compared to the linear regression model. Both, the computational and memory overhead, are at least one magnitude higher for any FFNN implementation and, therefore, the feasibility and area overhead of a run-time inference implementation of the different FFNNs is discussed in the following.

At least on IBM multicore processors, small micro-controllers are integrated for both power estimation — so-called power proxies — and for power management purposes [37]. Under the assumption of a similar integrated micro-controllers being available for run-time inference of the *shallow* FFNN for power estimation, an approximation of the transistor overhead is made. As a reference micro-controller the 32-bit ARM Cortex M0 — a well established and very minimalistic micro-controller— is used. It can conservatively be operated at 50 MHz with an implementation using less than 100 k transistors [122]. Besides these transistors needed for the compute logic, additional SRAM is needed to store the weights for the *shallow* FFNN. This will add an additional 120 k transistors to the transistor / area overhead. Each MAC operation and its associated load/store instructions take 6 cycles on the M0 leading to approximately 5 k cycles for a single inference of the *shallow* FFNN. Therefore, at a clock frequency of 50 MHz power estimations could be executed with a periodicity of $100\mu s$. One micro-controller would be needed per core if a power estimation rate of 10 kHz power estimations is required.

Such a micro-controller implementation leads to an overhead of approximately 250 k transistors under conservative assumptions and has to be compared to these power estimations being used for a complex out-of-order core having hundreds of millions of

transistors. The area overhead of— at maximum 0.25% —would decrease the average relative power estimation error by 7.5%, translating to better power and thermal management and thus the possibility of higher compute performance and/or higher energy efficiency. Area overhead could be further decreased by custom logic for FFNN inference which would however remove the programmability of the power estimation model. With a micro-controller implementation, both the neural architecture as well as the neural weights are simply defined as program code and can be adapted through the firmware of the multi-core processor. Depending on the requirements of the power and thermal management algorithms employed in the multi-core processors, however, multiple cores could also share a single micro-controller if the required estimation rate is substantially lower than 10 kHz.

In conclusion, the *multi-x* FFNNs and the *shallow* FFNN can be implemented for run-time power modeling on today's multicore processors. The *multi-x* FFNNs have, due to the multi-objective optimization approach which takes overhead directly into account and steers the architectures towards low run-time overhead, a clear advantage with run-time overhead being on average 60% smaller compared to the *shallow* FFNN which was generated with the single-objective optimization approach.

Run-time Inference Overhead for an Accelerator Implementation

The following accelerator implementation and evaluation as published in [3], was done by Nael Fasfous and should not be seen as a contribution of this thesis' author. The discussion of the accelerator implementation in comparison to the micro-controller implementation itself was done by the thesis' author. In this section, it is kept for the convenience of the reader, to give a fuller picture of how the power estimating FFNN architectures can be effectively implemented as hardware accelerators.

The implementation of the proposed FFNN architectures on hardware is investigated as an alternative to the micro-controller execution in Section 4.2.4. The simplicity of the FFNN architectures facilitates their synthesis as computation graphs on FPGA fabric. This follows a dataflow-style architecture where all the weights remain on-chip while the activations flow through the network. The compute graph is pipelined, therefore successive inputs can be processed in different parts of the graph at any point in time. This leads to large improvements in latency and throughput, but requires more on-chip memory to hold the intermediate results between the layers, as well as the weights of the FFNN. The implemented designs follow the method proposed in [123], but offer a higher numerical precision of 8-bit for weights and activations.

The resulting inference rates and FPGA resource usage numbers are given in Table 4.7. The FPGA 6-input LUTs can be very conservatively translated to a transistor count of 3.8 million transistors per 10,000 LUTs, assuming worst case truth tables and no optimization of the logic functions. Of the different FFNN architectures, the *multi-x* architectures offer significantly higher inference rates and simultaneously lower compute logic and memory usage compared to the *shallow*, *mid-sized*, *deep* architectures.

When comparing the overheads of the possible on-chip micro-controller implementation in Section 4.2.4 with the FFNN accelerator approach, one can observe that the

Table 4.7: FPGA resource usage, latency and maximum inference rates for an accelerator implementation [3]

Model	LUTs	Memory in kBit	Latency in us	Inference Rate in kHz
Single-Objective Optimization				
Shallow FFNN	15,763	88	6.3	158
Mid-sized FFNN	21,937	111	17.8	56
Deep FFNN	32,810	156	12.3	81
Multi-Objective Optimization				
Multi-1 FFNN	18,622	69	1.3	769
Multi-2 FFNN	17,272	49	0.9	1,064
Multi-3 FFNN	17,148	49	0.7	1,389
Multi-4 FFNN	17,151	48	1.1	893

accelerator approach for the *multi-x* architectures offers approximately a 100 times higher inference rate at a cost of approximately 20 times more transistors, under worst case transistor usage assumptions. Due to the very high inference rate of the accelerator, a single accelerator could be used for a 100-core processor computing the power estimations for all cores at an effective per-core inference rate of 10kHz. This would translate to five times lower area overhead compared to the per-core micro-controller implementation. Also, future very large core architectures might require higher power estimation rates than today's multi-core processors, making such an accelerator implementation on a per-core/compute tile/chiplet level even more advantageous.

Estimation Accuracy on Holdout Data

The neural architecture hyperparameters gained from the single-objective optimization and shown in Table 4.4 are used to generate the first three final FFNNs (*shallow*, *mid-sized*, *deep*) using the second round of 10-fold cross validation as presented previously in Section 4.2.2. Four additional FFNNs (*multi-1/2/3/4*) are generated in the same way. They are based on the Pareto optimal neural architecture hyperparameters gained from the multi-objective optimization as presented in Section 4.2.3 and shown in Table 4.5. Note, that for an actual deployment in a multicore processor, one would only generate either the *shallow* FFNN or one of the four *multi-x* FFNNs as they all occupy the same Pareto front with the *shallow* FFNN having lowest error values on the validation data and the other FFNNs having slightly higher error values but also lower overhead.

However, for providing an extensive performance review and analysis of the power estimating FFNNs, additionally the performance of the *mid-sized* and *deep* FFNN is presented. The final estimation accuracy comparison, i.e. what one would reasonably expect in an actual deployment, of all seven resulting FFNNs is determined on the holdout data. This is data that has neither been used for the neural architecture hyperparametrization nor for training the FFNNs. Table 4.8 shows the RMSE and percentage

Table 4.8: Estimation accuracy of FFNNs, linear model, and polynomial model on the holdout data set [2, 3]

Model	**RMSE**	**Relative Error**	**MAPE**
Single-Objective Optimization			
Shallow FFNN	0.26 W	4.5%	5.4%
Mid-sized FFNN	0.50 W	8.4%	8.0%
Deep FFNN	0.40 W	6.8%	7.0%
Multi-Objective Optimization			
Multi-1 FFNN	0.31 W	5.3%	5.9%
Multi-2 FFNN	0.43 W	7.2%	7.7%
Multi-3 FFNN	0.47 W	7.9%	8.2%
Multi-4 FFNN	0.36 W	6.1%	6.0%
Reference-based			
Linear [36]	0.75 W	12%	12%
Poly. Model	0.60 W	10%	11%

errors of the seven power estimating FFNNs as well as the model linear and the multivariate polynomial model as a comparison.

From the FFNNs, the *shallow* one has the best estimation performance with the remaining six FFNNs having worse error results as was to be expected from the previous validation data results. Compared to the linear model based on the state-of-the-art [36], a decrease in the relative error by 7.5% can be observed for the *shallow* FFNN. Such an improvement in estimation accuracy can be significant for both short-term power (density) management and long-term thermal management and energy management. For example, an overestimation of the power consumption of 7.5% over a time range of ten milliseconds can lead to power-inefficient mapping and scheduling of tasks or an early end to frequency boosting by the power management.

The best polynomial model had an RMSE of 0.01 W on the validation data but an RMSE of 0.60 W on the holdout data which is not significantly better than the performance of the linear power model. Compared to the polynomial power model, the relative estimation error of the power estimating *shallow* FFNN still offers a decrease by 5.5%. The result that the best performing FFNNs have two or three layers similar to the best-suited polynomial order being 2, can be interpreted as meaning that the underlying nonlinear performance counter/power relationship in itself is probably not overly complex. The comparatively bad polynomial model performance reconfirms the findings of McCullough et al. [30] that polynomial regression modeling very easily overfits the underlying performance/power data and is not well-suited to capture the nonlinear relationships. The *multi-x* FFNN power models perform between 0.8% to 3.4% worse concerning the relative error compared to the *shallow* FFNN but all perform still significantly better than the linear and polynomial models. However, these *multi-x* FFNN

power models also have far fewer hidden neurons compared to the *shallow* FFNN and thus lower complexity and run-time overhead.

In addition to the RMSE values and relative error values compared to average core power, the MAPE values which are defined as:

$$\text{MAPE} = 100\% \cdot E\left(\left|\frac{P_{forecast} - P_{actual}}{P_{actual}}\right|\right), \tag{4.12}$$

are also provided in the final comparison shown in Table 4.8. Compared to the relative error, MAPE penalizes underestimations of the core power consumption stronger than overestimations of core power which could be advantageous for power management algorithms operating on conservative assumptions. Although no significant qualitative differences between the relative error values and the MAPE values can be observed, the MAPE values have been included for sake of completeness.

On the final choice of which of these power models — FFNN-based, linear or polynomial — to use in a multicore processor, all come with a trade-off between modeling accuracy and complexity which directly translates to run-time overhead. If high accuracy is the most important objective during the design process of the multi-core system, the *shallow* FFNN would be a natural choice. If a slight degradation of 0.8% in accuracy is acceptable, the *multi-1* FFNN would allow for 61% lower run-time overhead compared to the *shallow* FFNN. If run-time overhead is the most important objective, i.e. if the cores are less complex or power consumption and power management is not a driving concern, the linear model is a straight-forward choice with its minimal complexity and run-time overhead.

Summary In this section, the usage of FFNN-based models were investigated for the same fine-grained run-time power estimation as aimed for in this thesis, i.e. on core-level with an estimation rate of 10 kHz. Suitable hyperparametrizations of the number of hidden neurons per layer for three different FFNN sizes (two, three and five hidden layers) were explored through a single-objective optimization of the neural architecture. In addition, a multi-objective optimization was proposed to optimize the hyperparameters for both high estimation accuracy and low run-time overhead for FFNNs with two to six hidden layers using an adapted NSGA-II methodology. To avoid underfitting nonlinear relations between performance counters and dynamic power, and to avoid overfitting the training data, 10-fold cross validations were applied throughout the two optimization methodologies. The *shallow* FFNN with two hidden layers proved to be the most accurate on the validation data and decreased relative power estimation errors on the holdout data — data which was not used for training the FFNN — by 7.5% compared to a state-of-the-art linear model and by 5.5% compared to a multivariate polynomial regression model. In comparison, the FFNNs derived from the multi-objective optimization fared slightly worse in regard to estimation accuracy than the *shallow* FFNN, however, they offer a decrease in run-time overhead of a factor of more than two compared to the *shallow* FFNN. Furthermore, a micro-controller-based implementation was again investigated, as previously in Section 4.1 which allows the FFNN inference / power estimation

to be executed at 10 kHz for each core with a maximum area overhead of 0.25% for large out-of-order cores. In conclusion, it was shown that FFNN-based power modeling is a viable approach for capturing nonlinear relations between performance counters and dynamic power, providing significant improvements in accuracy for fine-grained, high-rate power estimations. However, the FFNN-based approach is a black-box approach which does not allow for a straight-forward interpretation on where the nonlinear relations occur and what could be the underlying micro-architectural reasons for them. The next section will present a methodology for generating more transparent nonlinear power models which allow such micro-architectural investigations.

4.3 Nonlinear Transformation-based Power Model

This section presents and evaluates the methodology for nonlinear transformation-based power modeling [4].

Motivation and Contributions While Section 4.2 and Section 4.1 showed an ICA-based and an FFNN-based power modeling methodology, this section will present a nonlinear function-based power modeling methodology. The goal is again to develop fine-grained, highly accurate power models using performance counters as input while having low run-time overhead. As discussed previously in Section 4.2, one promising avenue to increase modeling accuracy is to effectively — in regard to overfitting and underfitting — and efficiently — in regard to run-time overhead — account for nonlinear effects in the underlying performance / power relationship. These nonlinear performance / power relations had been consistently shown for large processor architectures at power estimation rates of 1 Hz [30].

This section will propose an explicit and lightweight nonlinear power modeling approach for core-level power estimation at high estimation rates of 10 kHz; and demonstrate the advantages thereof. To account for nonlinear performance counter / core power relationships, nonlinear functions for performance counter input transformation are determined which maximize the linear correlation between transformed performance counters and power. Afterwards, a power model consisting of a nonlinear transformation block and a linear estimation block is generated which achieves higher power estimation accuracy compared to the state-of-the-art while still maintaining an overall low run-time overhead. To assess the run-time overhead, two different approaches of implementing the power model are investigated in this section: either periodically executing the power estimations on the available cores of the processor or a micro-controller implementation which lead to 0.11% computational overhead or 0.1% area overhead, respectively. Furthermore, this proposed nonlinear transformation methodology allows an investigation into the causes for nonlinear power responses. The following contributions are made in this section:

- Proposing a novel and lightweight methodology for power modeling which transforms performance counters with nonlinear functions, such that linear correlation with core power is maximized.
- Using LARS to make a DSE of the performance counter inputs with regard to the accuracy of the resulting model, i.e. select the most important performance counters.
- Showing that the generated nonlinear model reduces the relative error by up to 7% and 10% over a state-of-the-art linear model and an input-optimized polynomial model.
- Demonstrating how the proposed methodology can reveal the underlying microarchitectural causes for nonlinear effects.

First, the nonlinear modeling methodology, the usage of LARS for the DSE and the parametrization for the polynomial model are described in Section 4.3.1. The results in regard to the significance of the different performance counter inputs, the best polynomial order and the accuracy of the different power models are shown in Section 4.3.2. In Section 4.3.3, the nonlinear effects and their possible origin from the microarchitectural design of the multi-core system are discussed.

4.3.1 Power Model Generation and Lightweight Run-time Usage

The main basis for this work is that when complex workloads are executed on a multi-core processor, the relationship between specific performance counters $PC_{i,t}$ and switching activity $\boldsymbol{X}_t$ of all related logic gates and resulting dynamic power is nonlinear:

$$PC_{i,t} \sim f(\boldsymbol{X}_t), \tag{4.13}$$

with f being a nonlinear function. As the actual core-level switching activity $\boldsymbol{X}_t$ cannot be observed on-chip, this thesis proposes to use nonlinear transformations f_{trans} to reduce such nonlinear dependencies for more accurate power modeling:

$$P_{dyn,t} = \boldsymbol{\beta} \cdot f_{trans}(\boldsymbol{PC}_t) \cdot f_t \cdot V_t^x, \tag{4.14}$$

where β are linear power weights. Such an approach enables the generation of a very lightweight, yet more accurate model compared to a purely linear modeling approach. Suitable nonlinear transformations f_{trans} are obtained by observing the performance counters and the related power consumption response of the multi-core system. An overview of the flow in regard to the nonlinear power modeling and run-time estimation methodology is shown in Figure 4.9. The first fundamental step of data acquisition is the same as for ICA-based and FFNN-based power models as in Sections 4.1 and 4.2. First, a set of workloads is executed on the multi-core system at a static voltage and frequency. In parallel, package power is traced in a $1 \times n$ vector $\boldsymbol{P_{pack}}$ with n discrete points in time. Processor activity with m different performance counters, i.e. activity signals, are traced in a $m \times n$ matrix $\boldsymbol{PC}$.

At the next step, the methodology becomes different from the previous power modeling methodologies in this thesis. The pairwise Pearson correlation coefficient ρ_i between power $\boldsymbol{P}$ and each performance counter $\boldsymbol{PC}_i$ with $1 \leq i \leq m$ is computed as:

$$\rho_i(\boldsymbol{P}, \boldsymbol{PC}_i) = \frac{\sum_{t=1}^{n}(P_t - \overline{\boldsymbol{P}})(PC_{i,t} - \overline{\boldsymbol{PC}}_i)}{\sqrt{\sum_{t=1}^{n}(P_t - \overline{\boldsymbol{P}})^2 \sum_{t=1}^{n}(PC_{i,t} - \overline{\boldsymbol{PC}}_i)^2}} \tag{4.15}$$

where $\overline{\boldsymbol{X}}$ denotes the arithmetic mean of a vector $\boldsymbol{X}$ over time t. The resulting correlation coefficients $\rho_i(P, PC_i)$ are used to first compute — between a performance counter and the overall power consumption response — the baseline level of linear correlation. This baseline level of linear correlation is later used to assess if a nonlinear transformation is suitable to improve the modeling capability of that performance counter input. Many performance counters have a nonlinear dependency with power and therefore, show a low

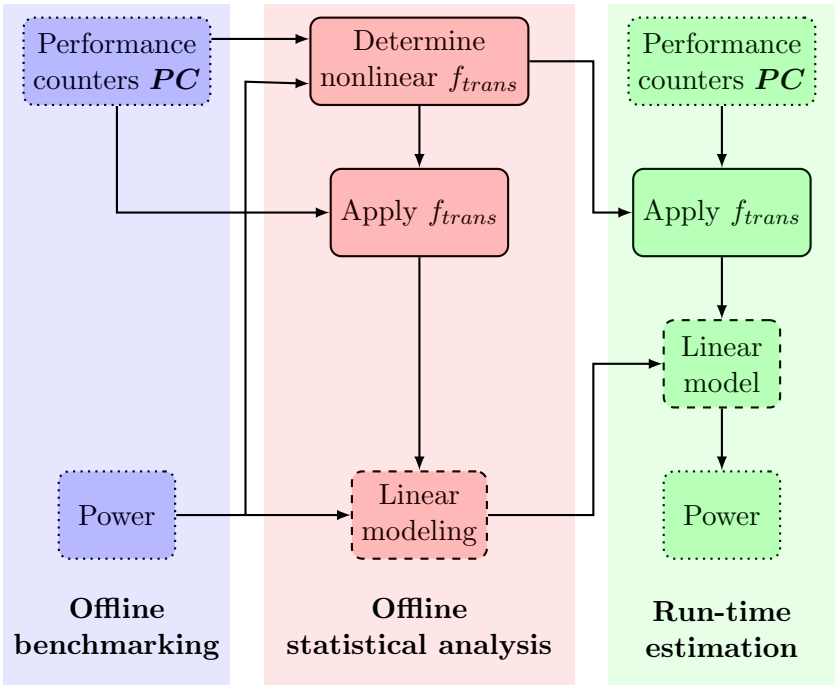


Figure 4.9: Power modeling and estimation flow (contributions of this work denoted with solid rectangles) [4]

(but not zero) linear correlation with power. Equation 4.15 is also used in the proposed algorithm determining which nonlinear functions f_{trans} are best suited to transform a nonlinear relationship between performance counter $\boldsymbol{PC}_i$ and power $\boldsymbol{P}$ into a linear relationship.

The *pseudocode* of the proposed algorithm to find such nonlinear transformations is shown in Algorithm 4.3.1. The algorithm iterates over a set F of nonlinear functions to transform the performance counters, compute the resulting correlation coefficients and keep the most effective f_{trans} which maximizes correlation for each transformed performance counter. These functions will afterwards be used to transform the performance counter data for both, regression modeling and run-time power estimation.

The set F of nonlinear functions as given in Table 4.9 are investigated to determine if these functions increase the linear correlation between a transformed performance counter and the observed power consumption. The exponential, root and logarithmic transformations can uncover nonlinear relationships where, e.g. small performance counter value variations in certain ranges lead to disproportional (nonlinear) dynamic power responses. In addition, trigonometric functions are also investigated to achieve a high coverage of possible nonlinear transformations although the expectation is that the subset of periodic trigonometric functions will not lead to any improvement in the linear correlation metric. The *stepsize* denotes for each applicable function how fine-

Algorithm 2: Selecting nonlinear transformations [4]

Input : Activity information $\boldsymbol{PC}$, power $\boldsymbol{P}$ and a set F of nonlinear functions
Output: Suitable nonlinear functions maximizing $\boldsymbol{\rho}(P, PC_i)$
foreach $\boldsymbol{PC}_i \in \boldsymbol{PC}$ **do**
  Compute $\rho_{i,base}(\boldsymbol{P}, \boldsymbol{PC}_i)$, ; `// Baseline`
end
foreach $f(.) \in F$ **do**
  foreach $\boldsymbol{PC}_i \in \boldsymbol{PC}$ **do**
    $\boldsymbol{PC}_{i,trans} \leftarrow f(\boldsymbol{PC}_i)$; `// Transformation`
    Compute $\rho_{i,trans}(\boldsymbol{P}, \boldsymbol{PC}_{i,trans})$
    if $|\rho_{i,trans}| > |\rho_{i,base}|$ **then**
      Store $f(.)$ for $\boldsymbol{PC}_i$ as $f_{trans,i}(.)$
      $\rho_{i,base} \leftarrow \rho_{i,trans}$
  end
end

Table 4.9: Set of nonlinear transformations investigated in this section [4]

Function Type	Function	Range	Stepsize
Exponents	PC^j	$1 < j < 10$	0.1
j-th root	$\sqrt[j]{PC}$	$-10 < j < 10$	0.1
j-th logarithm	$\log_j(PC)$	$1.1 < j < 20$	0.1
Exponential	$\exp^{PC}$	-	
Trigonometric	$\sin(PC), \sinh(PC), \arcsin(PC), \operatorname{arcsinh}(PC)$,		
	$\cos(PC), \cosh(PC), \arccos(PC), \operatorname{arccosh}(PC)$		
	$\tan(PC), \tanh(PC), \arctan(PC), \operatorname{arctanh}(PC)$		
	$\operatorname{cotan}(PC), \operatorname{cotanh}(PC), \operatorname{arccotan}(PC), \operatorname{arccotanh}(PC)$		

Figure 4.10: Run-time power estimation accounting for nonlinear performance counter / dynamic power relationship [4]

grained the value of j is varied within its given constrained range. This range constraint was determined roughly through heuristic variation of possible maximum values for the specific functions where it is applicable. Each variation of a function by the stepsize can also be interpreted as a distinct function $f(.) \in F$. It is important to note that for each performance counter a different transformation may be selected by the algorithm. The performance counter input data is also normalized and offset by one to have stable results for the logarithm and specific trigonometric functions, e.g. for arcsin.

After determining the most effective functions for maximizing linear correlation, these functions are used to transform the individual performance counters PC_i into transformed input data $PC_{trans,i}$ for $1 \leq i \leq m$. The vector containing all transformed performance counter for a point in time t is then denoted as $\boldsymbol{PC}_{trans,t}$. To have a complete power model for run-time estimation, another modeling step is needed which generates a regression model based on the transformed performance counters $\boldsymbol{PC}_{trans}$ with power $\boldsymbol{P}$. In this nonlinear transformation-based power modeling approach, an OLS regression is used — similarly to other state-of-the-art works as well as the ICA-based power modeling approach — and static power is subtracted from the package power to complete the dynamic power model as:

$$P_{dyn,t} = \boldsymbol{PC}_{trans,t} \cdot \boldsymbol{\beta} \cdot f_t \cdot V_t^2. \tag{4.16}$$

where $\boldsymbol{\beta}$ denotes the OLS regression coefficients. The nonlinear dynamic power model is then fully described by:

$$P_{dyn,t} = \sum_{i=1}^{m} (f_{trans,i}(PC_{i,t}) \cdot \beta_i) \cdot f_t \cdot V_t^2. \tag{4.17}$$

Such a nonlinear model is also called *Hammerstein* model [124] with both the nonlinear transformation block at the beginning and the corresponding correlation coefficients for run-time power estimation of the linear block afterwards being shown in Figure 4.10. The non-linear block introduces additional run-time overhead, however, processors usually execute the power estimation on dedicated on-chip logic [22]. The actual run-time overhead when then non-linear block is generated through function approximations — which incur a small area or computational overhead — will be discussed in more detail in Section 4.3.2. The application of the proposed methodology is shown in Section 4.3.2 where a nonlinear model of the core dynamic power is generated and run-time power is estimated based on the nonlinear model.

Least Angle Regression

LARS is a computationally efficient regression model selection algorithm [125] which allows to iteratively generate regression models based on the statistically most important inputs. In this thesis, it is also used to analyze the effect of the proposed nonlinear input transformations. The advantage of building power models with less than the maximum available performance counter inputs is both lower run-time overhead and the power model being more descriptive in regard to the underlying performance counter / power relations. With performance counters PC_i with $0 \leq i \leq m$ as independent variables and $\boldsymbol{P}_{dyn}$ as dependent variable, LARS in its first step sets all power weights β_i to zero:

$$\boldsymbol{P}_{dyn} = 0 + 0 \cdot \boldsymbol{PC}_1 + ... + 0 \cdot \boldsymbol{PC}_i + ... + 0 \cdot \boldsymbol{PC}_m \tag{4.18}$$

LARS then identifies the performance counter $\boldsymbol{PC}_i$ most correlated to P_{dyn} over all available measurements with $1 \leq t \leq t_{max}$, computes corresponding β_0 and β_i and adds $\boldsymbol{PC}_i$ to the *active set* of independent variables, i.e. the power model's inputs. The first such power model with a single performance counter as input is then defined as:

$$\hat{\boldsymbol{P}}_{dyn} = \beta_0 + \beta_i \cdot \boldsymbol{PC}_i, \tag{4.19}$$

where $\hat{\boldsymbol{P}}_{dyn}$ denotes the power *estimate* compared to the *actual* power $\boldsymbol{P}_{dyn}$ over time. Afterwards, the power weight β_i is increased in direction of $sgn(\rho_i(\boldsymbol{P}_{dyn}, \boldsymbol{PC}_i))$ until another performance counter $\boldsymbol{PC}_k$ has the same linear correlation with the residual error $r = |\hat{\boldsymbol{P}}_{dyn} - \boldsymbol{P}_{dyn}|^2$, i.e. until $\rho_{r,\boldsymbol{PC}_i} = \rho_{r,\boldsymbol{PC}_k}$. The new performance counter is then added to the active set of independent variables and the resulting model is as follows:

$$\hat{\boldsymbol{P}}_{dyn} = \beta_0 + \beta_i \cdot \boldsymbol{PC}_i + \beta_k \cdot \boldsymbol{PC}_k, \tag{4.20}$$

Both β_i and β_k are then increased in their joint least squared direction to find the next performance counter input β_l with the same correlation with the residual error. This is repeated until all performance counters have been added to the regression model which is an OLS model with all performance counters as input:

$$\hat{\boldsymbol{P}}_{dyn} = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot \boldsymbol{PC}_i. \tag{4.21}$$

The main advantage of LARS for modeling purposes is that it provides a linear solution path, i.e. different linear regression models, along the increasing number of performance counters as input.

In addition to the presented linear approach, LARS is also used to generate nonlinear power models with transformed performance counter $\boldsymbol{PC}_{trans,t}$ as input with the final model being

$$\hat{\boldsymbol{P}}_{dyn} = \sum_{i=1}^{m} \beta_i \cdot \boldsymbol{PC}_{trans,i} \cdot f \cdot V^2, \tag{4.22}$$

where β_0 has been subtracted under the assumption that it closely resembles static power. These additional nonlinear LARS generated power models provide insights, into

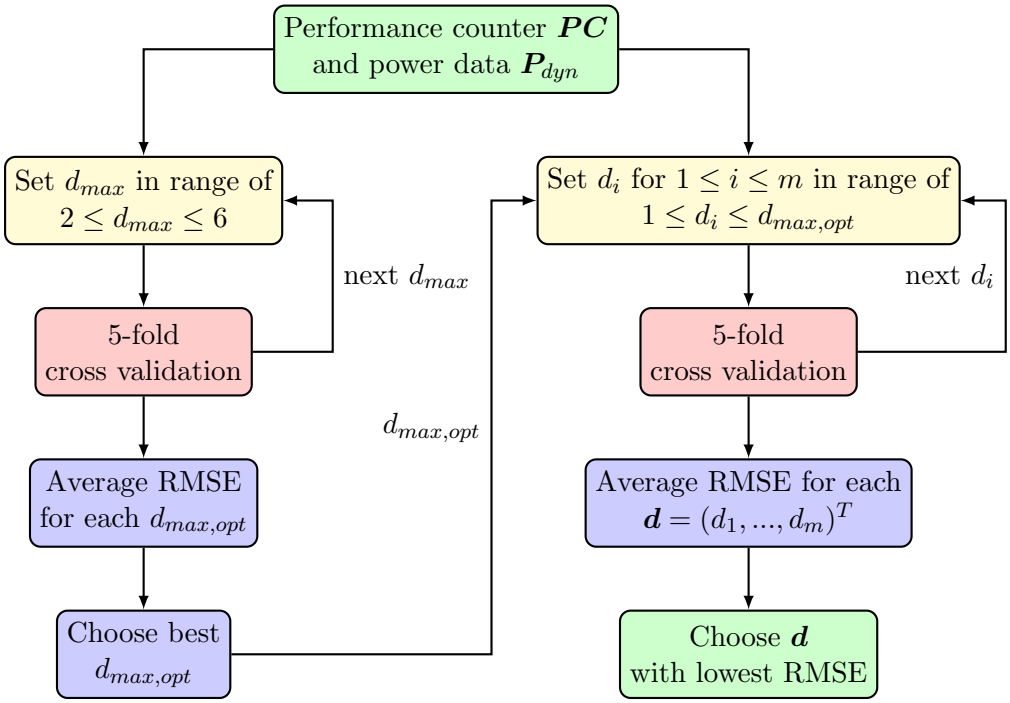


Figure 4.11: Multivariate polynomial modeling approach with variable degrees per input performance counters [4]

the power model behavior as well as the performance counter / power relations when the solution path of the purely linear input models is compared to the solution path of the proposed nonlinear transformation input models. In addition, LARS gives an intuitive ranking of the performance counter inputs which is used to generate differently sized polynomial models as comparison to the linear and nonlinear transformation models.

Multivariate Polynomial Regression

To compare the proposed nonlinear transformation approach with another comparatively lightweight nonlinear modeling approach, additional multivariate regression models are generated. Previous work [30] has also explored polynomial regression for power modeling, however, the generated nonlinear models did not consistently and significantly improve power modeling accuracy. As this thesis is mostly focused on the relation of specific performance counters and dynamic power consumption; and to be able to improve modeling accuracy while decreasing the risk of overfitting, an extended polynomial modeling approach is used in the following. Figure 4.11 shows the proposed multivariate polynomial modeling approach.

In contrast to [30], this approach is a two-staged heuristic to avoid overfitting the performance counter / power relations. This is achieved by determining specific polynomial orders for each performance counter input which try to optimally fit the linear or nonlinear relations. In the first stage, the maximum polynomial order d_{max} is set to

increasing values within the range of $2 \leq d_{max} \leq 6$ and polynomial models are generated using 5-fold cross validation. Based on this stage, the resulting polynomial models have the following form:

$$P_{dyn,t} = (\sum_{i=1}^{m} \sum_{d=1}^{d_{max}} \beta_{i,d} \cdot PC_i^d) \cdot f_t \cdot V_t^2. \tag{4.23}$$

Afterwards, for each d_{max} bracket the RMSE values are averaged over all validation folds and a $d_{max,opt}$ value is chosen for the second stage based on which d_{max} value offered the lowest average RMSE. In the second stage, the maximum orders for each performance counter input $\boldsymbol{PC}_i$ are differentiated using $d_{max,i}$ as each performance counter's specific polynomial order with $1 \leq d_{max,i} \leq d_{max,opt}$. This means that each performance counter input can be a either a linear input with $d_{i,max} = 1$ or any polynomial input with orders $2 \leq d_{max,i} \leq d_{max,opt}$. This exhaustive search for optimal specific input orders leads to $(d_{max,opt})^m$ different polynomial model orders which are encoded in an input order vector $\boldsymbol{d} = (d_1, ..., d_i, ..., d_m)^T$. Furthermore, for each possible model order again a 5-fold cross validation is used to generate statistically reliable model performance data, thus generating $5 \cdot (d_{max,opt})^m$ different polynomial power models. The exhaustive search for specific polynomial orders is computationally intensive. This computational intensity is the main reason for introducing the first stage of this methodology which limits the necessary computations in case of $d_{max,opt} < 6$.

After generating all power models, the RMSE values are averaged over the 5 different validation folds for each specific input order $\boldsymbol{d}$. Based on these average RMSE values, $\boldsymbol{d}$ with the lowest average RMSE is chosen as the final polynomial model order. The resulting multivariate polynomial model is then:

$$P_{dyn,t} = (\sum_{i=1}^{m} \sum_{d=1}^{d_i} \beta_{i,d} \cdot PC_i^d) \cdot f_t \cdot V_t^2. \tag{4.24}$$

While [30] had the model order fixed at $d_{max} = 3$, the proposed polynomial approach in this section allows us to differentiate the polynomial order for each specific performance counter input. Although being far more computationally intensive, this approach minimizes the risk of overfitting the underlying performance counter / power consumption data during model generation. In addition, this approach allows the comparison of the polynomial order of each performance counter input with the nonlinear transformations found to be maximizing the linear correlation of the performance counter inputs with power consumption; enabling further interpretation of the performance counter / power relations.

4.3.2 Experimental Evaluation

In this section, the results of the experimental evaluation are shown and the possible reasons for the effectiveness of nonlinear power modeling discussed. Again, the experimental setup described in detail in Chapter 3 is used to generate the underlying power and performance data. First, the results of the nonlinear transformation search is

shown. Afterwards, the linear solution path obtained from LARS is presented for linear and nonlinearly transformed inputs and the best-suited multivariate polynomial orders are shown. Finally, the accuracy of the different power models under different operating conditions and their relative software and hardware overhead are discussed.

Nonlinear Transformations

The experimental evaluation first gathers power and performance counter data by executing the 20 different benchmarks on the simulated multicore system as described in detail in Chapter 3 at a frequency of 3.0 GHz. Based on this data, the level of linear correlation with between individual performance counters and dynamic core power is computed. All cache related performance counters (LxLI, LxSI, LxLM, LxSM, LxCLK) showed low linear correlation $|\rho| \leq 0.25$ with core power. Medium level of correlation $0.25 < |\rho| \leq 0.75$ was observed for BPU, FP, and C0. The only performance counter showing high correlation $0.75 < |\rho|$ was IPC. These linear correlation results are the baseline for optimizing the power model by transforming performance counters via suitable nonlinear functions.

After determining this baseline, the most effective nonlinear functions for maximizing linear correlation between each performance counter and power are searched according to Algorithm 4.3.1. The most suitable functions to transform the performance counters for power modeling are shown in Table 4.10. The remaining performance counters, e.g. IPC and C0, are best kept as a linear input in the power model with $f_{trans,i}(\boldsymbol{PC_i}) = \boldsymbol{PC_i}$ as no nonlinear function increased the level of linear correlation with core power.

As was to be expected, periodic trigonometric functions like sin and cos did not yield any positive results in regard to increasing linear correlation. This is likely due to the performance counter / power relation being time-invariant. In contrast, linear correlation with power is improved when using tanh and arcsinh for cache related performance counters. Note, that all nonlinear functions to be found to improve linear correlation are monotonically increasing and that both $\text{arcsinh}(PC)$ and $\sqrt{PC}$ are unbounded functions while tanh saturates with increasing values of its input.

Table 4.10: Transformations maximizing linear correlation of specific performance counters with dynamic power [4]

Function	**Performance Counter** PC with (j)
$\tanh(PC)$	L2LI, L2LM, L3LI, L3LM
$\text{arcsinh}(PC)$	L2SI, L2SM, L3SI, L3SM
$\sqrt[j]{PC}$	BPU(2), L2CLK(8), L3CLK(4)

The correlation coefficients for all performance counters before and after applying the respective nonlinear transformation maximizing the linear correlation are shown in Table 4.11. Aside of L3LM and L2CLK, one can observe increases in the absolute linear correlation coefficients computed as the Pearson correlation coefficient. Although, no performance counter experiences an increase in the linear correlation such that its linear

Table 4.11: Correlation coefficients for each performance counter after applying the nonlinear transformation [4]

Performance Counter	**Correlation coefficients**	
PC	$\lvert\rho(\boldsymbol{PC}, P_{dyn})\rvert$	$\lvert\rho(\boldsymbol{PC}_{trans}, P_{dyn})\rvert$
L2LI	0.06	0.40
L2LM	0.04	0.31
L3LI	0.01	0.24
L3LM	0.05	0.09
L2SI	0.13	0.43
L2SM	0.16	0.42
L3SI	0.05	0.41
L3SM	0.15	0.38
BPU	0.74	0.80
L2CLK	0.03	0.14
L3CLK	0.14	0.29

correlation coefficient values $|\rho(\boldsymbol{PC}_{trans}, P_{dyn})|$ reach the absolute values of IPC or BPU, the increases are still very substantial compared to raw $|\rho(\boldsymbol{PC}, P_{dyn})|$ values. These increases can be exploited in the linear regression model which is afterwards generated using such nonlinearly transformed performance counter inputs.

Results of Using LARS

LARS is used to generate OLS linear regression models using either the performance counters $\boldsymbol{PC}$ as independent variables or the transformed performance counters $\boldsymbol{PC}_{trans}$ as independent variables. Both $\boldsymbol{PC}$ and $\boldsymbol{PC}_{trans}$ were normalized before using them as inputs in LARS. The dependent power variable P was centered beforehand. The selection order of the two different variants of inputs $\boldsymbol{PC}$ and $\boldsymbol{PC}_{trans}$ is given in Table 4.12.

IPC is chosen by LARS as most significant power model input both for the linear power model and the nonlinear transformation power model. Afterwards, one observe that LARS gives higher modeling priority to the cache relevant performance counters in case that the performance counters have been transformed with the linear correlation maximizing nonlinear functions. This implies that these performance counters hold significant modeling information but are not as usable for linear modeling compared to the C0, BPU and FP performance counters. The benefit of transforming specific performance counters is further supported when comparing the AIC for both the linear and the nonlinear transformation power models. The AIC is computed as:

$$AIC = 2m - 2ln(\hat{L}), \tag{4.25}$$

with $\hat{L}$ as the maximum of the likelihood function for the underlying model. The AIC is a means to compare different models in regard to their estimation ability for unknown

Table 4.12: Selection order of the performance counters by the LARS algorithm [4]

Order of Selection	**Performance Counter *PC***	**Transformed Performance Counter PC_{trans}**
1.	IPC	IPC
2.	C0	L2SM
3.	BPU	L3SI
4.	L2SM	L3SM
5.	FP	L3LM
6.	L2LI	L2LM
7.	L3LM	L3LI
8.	L2SI	L2SI
9.	L2CLK	L2CLK
10.	L3SM	L2LI
11.	L3SI	BPU
12.	L2LM	FP
13.	L3LI	L3CLK
14.	L3CLK	C0

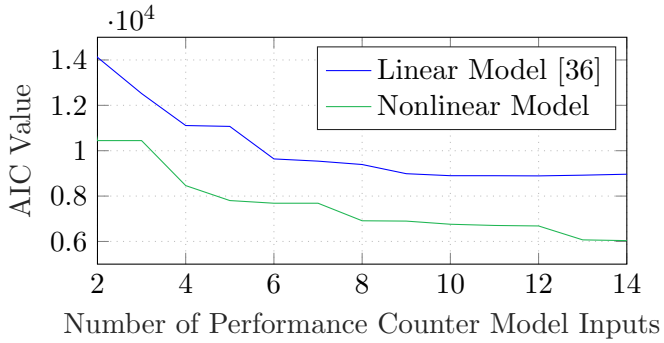


Figure 4.12: Comparison of AIC values for linear model and nonlinear transformation model with increasing number of performance counter inputs [4]

data and computes the trade-off between simple models, i.e. lower number of inputs, and the goodness-of-fit of the models. A lower AIC value signifies a better model in regard of both the risk of overfitting and the risk of underfitting the underlying data.

Figure 4.12 shows the AIC values for increasing number of performance counter inputs for the respective power models beginning at two inputs – due to IPC being the same optimal first input for the linear and the nonlinear transformation models – and finishing at the full complement of 14 performance counter inputs. Note, that the dependent variable P_{dyn} is not transformed by a nonlinear function, only centered for both modeling approaches. Therefore, the AIC values of the linear and nonlinear transformation model are directly comparable.

One can observe that the AIC is consistently lower for the nonlinear transformation power model at the sample complexity level compared to the linear model. The linear modeling quality improves until a total number of nine performance counters are used as model inputs and stagnates afterwards. In contrast, the nonlinear model improves until all available transformed performance counters are included in the power model as independent input variables. Based on this, a comparison of the accuracy of linear and nonlinear transformation models when all performance counters are used as inputs will be shown. In addition, in Section 4.3.2, the linear and nonlinear models will be compared when the number of performance counter inputs is reduced, e.g. for lowering modeling complexity and thus run-time overhead.

Multivariate Polynomial Model Orders

As described in detail in Section 4.3.1, the proposed methodology to generate multivariate polynomial power models consists of two steps. First, the general polynomial order minimizing modeling error was determined and it was found that $d_{max,opt} = 2$ produces polynomial power models with lowest average RMSE. For the specific polynomial order $\boldsymbol{d}$, the best polynomial power models were generated when the performance counters IPC, FP and L2LI are set to a maximum order of $d = 1$, i.e. these performance counters are used as linear inputs for the polynomial models. For the remaining performance counters, the polynomial models performed best with $d = 2$, and are thus being used as both linear as well as quadratic model input. When comparing these results with the nonlinear transformations, partial overlaps in regard to the inputs being used as polynomial input or being transformed can be observed. While both the nonlinear transformation approach and the optimized, multivariate polynomial model keep IPC and FP as a purely linear input, the proposed transformation approach transforms L2LI with a monotonically increasing function while keeping C0 as a purely linear input.

To have a baseline comparison to the approach in [30] where the polynomial order was the same for all power model inputs, a polynomial model with a fixed $d_{max} = 3$ for all performance counter inputs were generated and is used as an additional reference polynomial model. This complements the final power model comparison with a state-of-the-art linear model, the proposed nonlinear transformation model and the proposed multivariate polynomial model.

Accuracy Analysis

Modeling Accuracy with Known Core Power For assessing the different power modeling methodologies in regard to modeling accuracy, two different scenarios are investigated. The first scenario, described in the following, is a synthetic scenario where actual core power consumption is known / measurable during model generation which likely is not applicable a real-world deployement, however, still gives insights into the capacity of the different methodologies to capture nonlinear performance counter / power relations. Note that this scenario with known core power is only possible due to using simulator-based performance counter and power data. The other more realistic scenario of unknown core power for model generation, will be shown in the next paragraph.

Starting with known core power for generating the power model, a 5-fold cross-validation is used for model generation to get statistically reliable results. The performance and power data of each executed benchmark is distributed into one of five groups; each group containing the performance and power data of four benchmarks. Each group is used once as validation data set with the remaining groups constituting the model generation data set. The performance counters are transformed according to the functions shown in Table 4.10 for generating the nonlinear model with its nonlinear transformation block and linear regression block. For comparison, linear regression models with the original performance counters as input are also generated.

During the model generation step, a linear model, the reference polynomial model [30] (Poly-3), the proposed multivariate polynomial model (Poly-d) and the proposed nonlinear transformation model are generated. Afterwards, during the evaluation step, all four models are used to estimate power based on the performance counter data from the current validation group. To determine estimation errors, this estimated power derived from the models is compared with the actual power values from the validation group. The 5-fold cross-validation is repeated 6000 times with benchmarks chosen pseudorandomly such that no identical combination is repeated out of the $\binom{20}{5}$ possible combinations. The resulting average estimation error over all validation groups and the worst case estimation error are given in Table 4.13.

Two error metrics are distinguished: total RMSE and relative RMSE compared to average core power for that validation group. The thus gained results show that for the proposed nonlinear model, the estimation error decreases significantly for the mean, median and worst case error compared to the linear model and compared to both polynomial models. While the proposed Poly-d model, with performance counter specific polynomial degrees, performs better than the linear model and the Poly-3 model, the nonlinear transformation model is still significantly better on average and also in the worst case than the Poly-d model. Interestingly, the worst case performance of the Poly-3 model is slightly worse than the worst case of the linear model; this is probably due to overfitting in specific scenarios with disadvantageous validation groupings. As mentioned in the beginning, however, for an actual multicore system, one cannot usually use measured or simulated core power information to train a core power model.

Table 4.13: Power Estimation error as absolute RMSE and relative RMSE for the four power modeling approaches [4]

Estimation Error when Core Power is Known for Model Generation				
Total RMSE	**Linear [36]**	**Poly-3 [30]**	**Poly-d**	**Nonlinear**
Mean	0.57 W	0.56 W	0.52 W	0.44 W
Median	0.58 W	0.53 W	0.49 W	0.43 W
Worst Case	1.4 W	1.6 W	1.4 W	0.9 W
Relative RMSE				
Mean	8.9%	8.7%	8.1%	6.8%
Median	9.0%	8.2%	7.6%	6.7%
Worst Case	21%	25%	21%	13%

Estimation Error when Core Power is Unknown for Model Generation				
Total RMSE	**Linear [36]**	**Poly-3 [30]**	**Poly-d**	**Nonlinear**
Mean	0.84 W	0.80 W	0.68	0.58 W
Median	0.77 W	0.82 W	0.70	0.52 W
Worst Case	1.9 W	2.1 W	1.8 W	1.5 W
Relative RMSE				
Mean	13%	12%	10%	9.0%
Median	12%	13%	11%	8.1%
Worst Case	30%	33%	28%	23%

Modeling Accuracy with Unknown Core Power To model core power consumption without needing core power as modeling input, one usually uses package or system level power information and identifies correlations with performance counter data. As shown in Section 2.2, most power estimation approaches use some form of linear model generation to correlate performance with power consumption and build a regression power model. To generate power models using only package power, the methodology as proposed in [36] is applied in this thesis. In addition to the previously used (generic) benchmarks, i.e. PARSEC and SPLASH-2, synthetic workloads were executed on the multicore processor to isolate particular microarchitectural units of the cores, their power consumption and corresponding power weights $\boldsymbol{\beta}$. The processor's baseline idle power including uncore power is determined and then subtracted from the modeled package power to derive the dynamic core power as shown in the following equation:

$$P_{core-0,t} = P_{pack,t} - P_{idle+uncore}, \tag{4.26}$$

where $P_{core-0,t}$ describes the indirectly derived power of core-0 to generate the power models, $P_{pack,t}$ is the power consumed by the processor on package-level and $P_{idle+uncore}$ is the idle power when core-0 is halted but not yet power gated nor in a sleep state.

For the nonlinear transformation model and the polynomial models, the same approach was used and the performance counter inputs $\boldsymbol{PC}$ were exchanged with the transformed performance counter inputs $\boldsymbol{PC}_{trans}$ for model generation and run-time estimation. The generic benchmarks were again distributed into groups of 4 benchmarks and each group used either for model generation or evaluation/validation through a 5-fold cross validation which was again repeated 6000 times.

For this scenario, where core power is unavailable for model generation, the estimation errors for the linear and nonlinear model are also given in Table 4.13 below the estimation errors with known core power during model generation. Overall, estimation accuracy decreases for all models — as was to be expected — however, the nonlinear transformation model still provides higher accuracy compared to the state-of-the-art linear power model, the state-of-the-art polynomial power model and the polynomial Poly-d power model.

Although the decreases in relative RMSE of 4% / 3% on average and 7% / 10% in worst case scenarios compared to the reference linear/polynomial power model might seem small, these can have significant impacts on the effectiveness of run-time power and thermal management. For example, a continuously overestimated power consumption on a core can lead to unnecessary remapping of tasks or a premature stop of frequency boosting of the afflicted core.

Reduced Number of Performance Counter Inputs

Decreasing the number of performance counters as input, decreases the model complexity and run-time estimation overhead but also increases the risk that the power model cannot accurately describe the run-time power behavior. When looking at the information criterion AIC values in Figure 4.12, distinct improvements / jumps can be seen for the linear power model when the number of performance counter inputs is increased to four inputs, then to six inputs, and then to nine inputs. To have a fair comparison with the nonlinear model, number of inputs were chosen as 4 and as 9 for the accuracy investigations of the other power models with reduced number of performance counter inputs, i.e. the expectation is that the linear model will perform best at these input sizes making these input sizes conservative assumptions for the other models. Due to time and space limitations, the power model input size of six performance counters was not further investigated; also due to input sizes four and nine covering roughly 1/3 and 2/3 of the maximum input size of 14. For the linear model and polynomial models, the selection order was the same as previously determined by LARS for the linear power model; while using the selection order determined for transformed performance counters for the nonlinear model as given in Table 4.12. To get statistically reliable model performance data, the same procedure of 5-fold cross validation with 6000 runs and model generation based on unknown core power is repeated as in the previous paragraph. The resulting estimation errors for power models with reduced performance counter inputs are shown in Table 4.14.

As was to be expected, all power models perform on average worse when less performance counter inputs are allowed for model generation, i.e. the power models have

Table 4.14: Relative power estimation error for power models with reduced performance counter inputs [4]

Relative RMSE	**Linear [36]**	**Poly-3 [30]**	**Poly-d**	**Nonlinear**
4 Performance Counters as Input				
Mean	19%	16%	16%	12%
Worst Case	31%	29%	27%	26%
9 Performance Counters as Input				
Mean	14%	15%	15%	10%
Worst Case	30%	31%	30%	23%

less information to compute a power estimate leading to worse accuracy. However, the nonlinear transformation model shows the smallest degradation in accuracy of all power models. Interestingly, the worst case estimation error for the polynomial models actually decreases when only four performance counters are used as input; compared to the full set of performance counters. This is probably due to a lower risk of overfitting when reducing its model complexity by decreasing its inputs. Overall, this analysis shows that the nonlinear transformations are robust in regard to reduced performance counter sizes and that the performance counter transformation functions likely do not lead to overfitting of the underlying data. If such overfitting had occurred, one would expect the nonlinear transformation model to show erratic behavior with lower input sizes and thus higher accuracy degradation.

Estimation Accuracy for the Different Benchmarks

In the following, the estimation accuracy for the different benchmarks is investigated. For this, the relative errors for each benchmark were averaged over each benchmark's occurrence in the validation fold. This includes the worst case scenarios in which the power models performed the worst. The resulting benchmark-dependent errors for the linear power model and the nonlinear power model are shown in Figure 4.13.

Notably, the worst case scenario for both power modeling methodologies happens when trying to estimate the core-level power consumption of PARSEC freqmine, a data mining benchmark. The linear power model outperforms the nonlinear model for PARSEC canneal and for SPLASH-2 radix by small margins. Comparing the PARSEC and SPLASH-2 suites, the overall estimation errors are more variable for the PARSEC benchmarks compared to the SPLASH-2 benchmarks for both linear and nonlinear power models while estimation accuracy is more consistent for the SPLASH-2 suite for both approaches. Also, the nonlinear power model provides consistently larger accuracy improvements over the linear power model for the SPLASH-2 suite compared to the PARSEC suite.

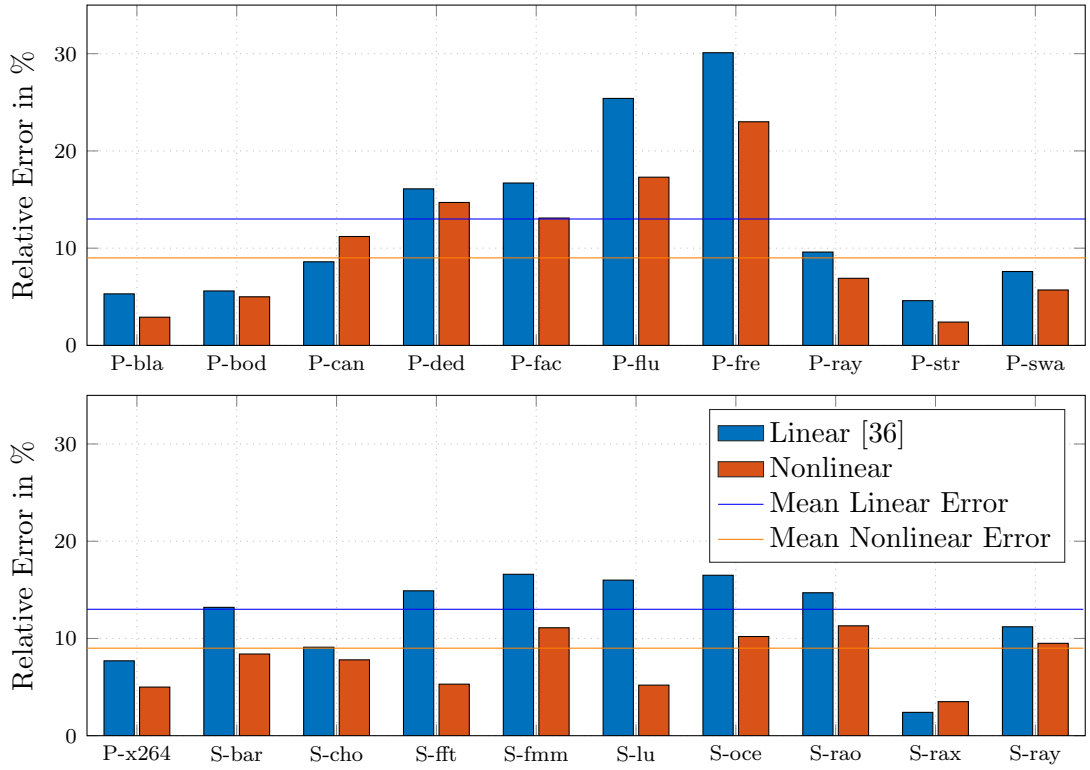


Figure 4.13: Relative error values for power models generated with unknown core power for each PARSEC (P) and SPLASH-2 (S) benchmarks with benchmark names abbreviated with first three letters except SPLASH-2 radiosity (S-rao) and radix (S-rax) [4]

Nonlinear Models at Different Operating Frequencies

The previous experiments were all done at a static frequency of 3 GHz. However, two questions arise in regard to frequency changes when the nonlinear power models are used in a multi-core system:

- Are the nonlinear transformations which optimize linear correlation the same at different voltage/frequency states?
- Do the regression coefficients β_i have to be adapted for different frequencies due to the applied nonlinear transformations?

To assess the impact of changes in the operating frequencies, the benchmarks listed in Table 3.3 are executed at 1 GHz and 2 GHz. The nonlinear transformations maximizing the linear correlation coefficients are now again determined according to Algorithm 4.3.1. The first observation is that the IPC, C0 and FP performance counters are not improved through any of the nonlinear functions and further observe that the nonlinear functions almost all stay the same as previously shown in Table 4.10 for 3 GHz. The only exception is the BPU performance counter. The detailed impact on the correlation coefficients

for the nonlinear transformations is shown in Table 4.15 where $|\rho| = |\rho(\boldsymbol{PC}_{trans}, P_{dyn})|$ and Δ denotes $|\rho(\boldsymbol{PC}_{trans}, P_{dyn})| - |\rho(\boldsymbol{PC}, P_{dyn})|$, i.e. the difference in correlation coefficients between performance counters and transformed performance counters. With

Table 4.15: Correlation coefficients after applying the nonlinear transformation at different frequencies and delta to the correlation coefficients of the regular performance counters [4]

Performance Counter PC	$\|\rho\|$ at 3 Ghz	Δ	$\|\rho\|$ at 2 GHz	Δ	$\|\rho\|$ at 1 GHz	Δ
L2LI	0.40	0.34	0.22	0.19	0.10	0.05
L2LM	0.31	0.27	0.13	0.10	0.12	0.07
L3LI	0.24	0.23	0.14	0.11	0.12	0.03
L3LM	0.09	0.04	0.10	0.09	0.13	0.03
L2SI	0.43	0.30	0.56	0.32	0.59	0.27
L2SM	0.42	0.26	0.54	0.26	0.57	0.31
L3SI	0.41	0.36	0.55	0.29	0.55	0.24
L3SM	0.38	0.23	0.56	0.29	0.60	0.30
BPU	0.80	0.06	0.74	*-0.04*	0.76	*-0.05*
L2CLK	0.14	0.11	0.05	0.04	0.04	0.03
L3CLK	0.29	0.15	0.38	0.14	0.40	0.15

decreasing frequencies, the linear correlation values decrease for cache load instructions and increases for cache store instructions. In regard to the consistency of reusing the same nonlinear transformations for the same performance counter inputs at different operating frequencies, BPU is the sole exception for which the linear correlation values actually decreases when applying its predetermined nonlinear transformation function at 1 GHz or 2 GHz. Although, the decrease in linear correlation is rather small, a negative impact on the power estimation accuracy is still expected from this behavior.

However, ideally one would want to keep the same transformations over the full range of frequencies to be able to reuse the power weights $\boldsymbol{\beta}$ at all operating frequencies. Therefore, the same transformations as specified in Table 4.10 are reused for estimating power at both 1 GHz and 2 GHz. To be able to reuse the power models which were generated in the previous sections, the performance counter inputs are normalized to the operating frequency according to:

$$PC_{i,norm} = \frac{PC_i}{f_{GHz}}, \quad \forall i (i \neq IPC); \tag{4.27}$$

where f_{GHz} is the current operating frequency in GHz; this avoids unnecessarily small floating point values. The IPC performance counter is not normalized as it is already normalized to the cycle length and thus to the operating frequency. The regression coefficients — aside of the coefficient(s) associated with IPC — of the power models at 3 GHz have to then be multiplied with 3 (GHz) to be frequency neutral. With this, the

Table 4.16: Relative RMSE at different operating frequencies and unknown core power for power estimation model generation [4]

Total	Linear [36]	Poly-3 [30]	Poly-d	Nonlinear
Operating Frequency 3 GHz				
Mean	13%	12%	10%	9.0%
Worst Case	30%	33%	28%	23%
Operating Frequency 2 GHz				
Mean	13%	15%	14%	9.2%
Worst Case	29%	34%	32%	22%
Operating Frequency 1 GHz				
Mean	13%	16%	16%	9.1%
Worst Case	30%	35%	36%	22%

regression coefficients $\boldsymbol{\beta}$ previously determined at 3 GHz can be applied on performance counter and power data generated at 1 GHz and 2 GHz to estimate run-time core power. Due to time limitations, the investigation on the frequency-stability of the power models was restricted to the power models generated with *unknown core power*, i.e. the real deployment scenario 5-fold cross validations is employed again while keeping the benchmarks divisions from previous experiments, i.e. the power models do not estimate power of benchmarks used for generating the power models.

The relative RMSE results are shown in Table 4.16 with the previously obtained 3 GHz results added as comparison. The estimation accuracies for the linear and nonlinear transformation model change only marginally. The nonlinear transformation model performs slightly worse at 1 GHz and 2 GHz compared to it operating at 3 GHz, possibly due to keeping the BPU transformation which was shown to be not optimal at these lower operating frequencies. However, keeping the nonlinear transformations the same over a wide frequency range allows for a simplified implementation of the run-time nonlinear power estimation model. In contrast, both polynomial models perform significantly worse, probably due to overfitting at 3 GHz which is only observable when applying the polynomial model at different operating frequencies than it was generated at. Overall, one can conclude that both linear power models and the proposed nonlinear transformation models are well-suited for application at different operating frequencies. Thus with all performance counter inputs normalized to the frequency, the regression coefficients $\boldsymbol{\beta}$ can be kept the same over different Vf states.

Run-time Power Estimation Overhead

To determine the run-time overhead for either a software or hardware implementation, the same approximation strategy as in Section 4.1 and Section 4.2 is used with first calculating the necessary number of MAC operations to execute a single power estimation for each different power model. To approximate the compute overhead for the necessary nonlinear transformations functions, a series approximation of the nonlinear functions

Table 4.17: Computational and memory overhead for a single power estimation [4]

Model	Number of MAC Operations	Memory in kBit
Linear	14	0.5
Poly-3	112	1.5
Poly-d	36	0.8
Nonlinear	377	1.4

is used. An overview of the necessary MAC operations and memory — limited to the power models with 14 performance counters as input — is given in Table 4.17. The required MAC operations for computing a single power estimation and the memory needed to retain the power model parameters are reported on core-level. The system-wide overhead depends on the overall number of cores of the multi-/many-core system for which run-time power estimations are needed and scales linearly with the number of cores for all power models.

The pure compute overhead — in terms of needed MAC operations — of using the nonlinear transformation model increases 27 times compared to the linear model and 3.4 times compared to the reference polynomial model. The memory overhead in kBit is 3× larger for the nonlinear transformation model compared to the linear model and similar to the reference polynomial model. Overall, these are significant increases in raw compute and memory overhead. The question arises if these overheads contribute towards a significant run-time overhead when the nonlinear transformation model is implemented on a multi-core system; which will be discussed in the following.

Overhead Software Implementation The power models can be periodically executed during run-time on their respective cores. Including DRAM access times and disregarding context switches, the computation time for one power estimation of the linear model is approximately 70 ns at 3 GHz on the multi-core system described in Chapter 3, i.e. this is the total time to fetch the data from memory and to execute the necessary MAC operations. The computation time for the proposed nonlinear power model is 190 ns for one estimation and thus 2.7 × larger than for the purely linear power model. Note, that the memory accesses add significant computational latency for both, the linear power models and the nonlinear power models and thus the run-time increase in run-time overhead is a magnitude smaller than what would have been expected from the pure MAC operations comparison. At an estimation rate of 10 kHz, the required computation times translate to computational overheads of 0.07% and 0.19%, respectively. For such a software implementation, the expected power overhead of executing the power models during run-time is similar to the computational overhead.

Overhead Hardware Implementation As mentioned in previous sections, some multi-core processors have integrated micro-controllers for run-time power estimation as well

as for running power and thermal management algorithms [37]. Such micro-controller implementations seem to be a good trade-off between area, power and programmability. Therefore, a possible micro-controller implementation is investigated in the following for run-time power estimation. An 32-bit ARM Cortex-M0 micro-controller running at 50 MHz can be integrated with a small area footprint of less than 100 k transistors [122] and needs six cycles to compute a single MAC operation. For the 32-bit ARM Cortex-M0, this would translate to 2.3 k cycles for a single power estimation of the nonlinear power model and would allow a power estimation rate of approximately 21 kHz which is double the target estimation rate in this thesis of 10 kHz. In contrast to the FFNN neuron weights needed in the FFNN-based methodology presented in Section 4.2, there is negligible memory overhead for using nonlinear approximations functions in addition to the linear regression part and thus will be neglected in the following overhead analysis. In comparison, the linear model would only require 84 cycles for a single power estimation. However, such a power estimating micro-controller adds are overhead which has to be considered for the viability of the different power modeling approaches.

Today's complex out-of-order cores integrate 100s of millions of transistors, the area overhead of 100 k transistors per core for the power estimating micro-controller would lead to an area overhead of less than 0.1%. With the micro-controller operating at a slow 50 MHz, one can assume that the dynamic power overhead of executing power estimations is significantly less than 0.1% while static power overhead will be around 0.1% of overall static power consumed by the core itself. In total, the power overhead for run-time power estimation for the nonlinear power model should still be distinctly below 0.1% for such a proposed micro-controller implementation. In regard to both implementations and assuming that core power has to be estimated in some form, the incurred run-time overheads of the proposed nonlinear power model over a linear power model are comparatively low.

4.3.3 Discussion of Nonlinear Performance Counter/Power Relationship

With the nonlinear transformation power model being transparent — in contrast to the proposed FFNN-based power model — in regard to how it minimizes nonlinear performance counter / power relations, it can be used to investigate these relations. In the following, hypotheses are provided for why the model accuracy is improved by the nonlinear modeling approach; with the underlying nonlinear performance counter / power relations being referred to in the following as *nonlinear effects*. First, the basic microarchitecture of the out-of-order processor is shown in Figure 4.14. The core *front end* is responsible for instruction fetch, decoding and branch prediction (BPU). In the *execution engine*, the instructions are reordered, scheduled and executed. The *memory subsystem*, loads and stores instructions and data from the *uncore* which is not part of the core itself but shown for easier understanding of the memory subsystem and its interactions with the whole multi-core system. Taking into account the nonlinear functions and the qualitative performance counter/power relationship for the generated nonlinear power models as shown in Table 4.18, the following observations are made:

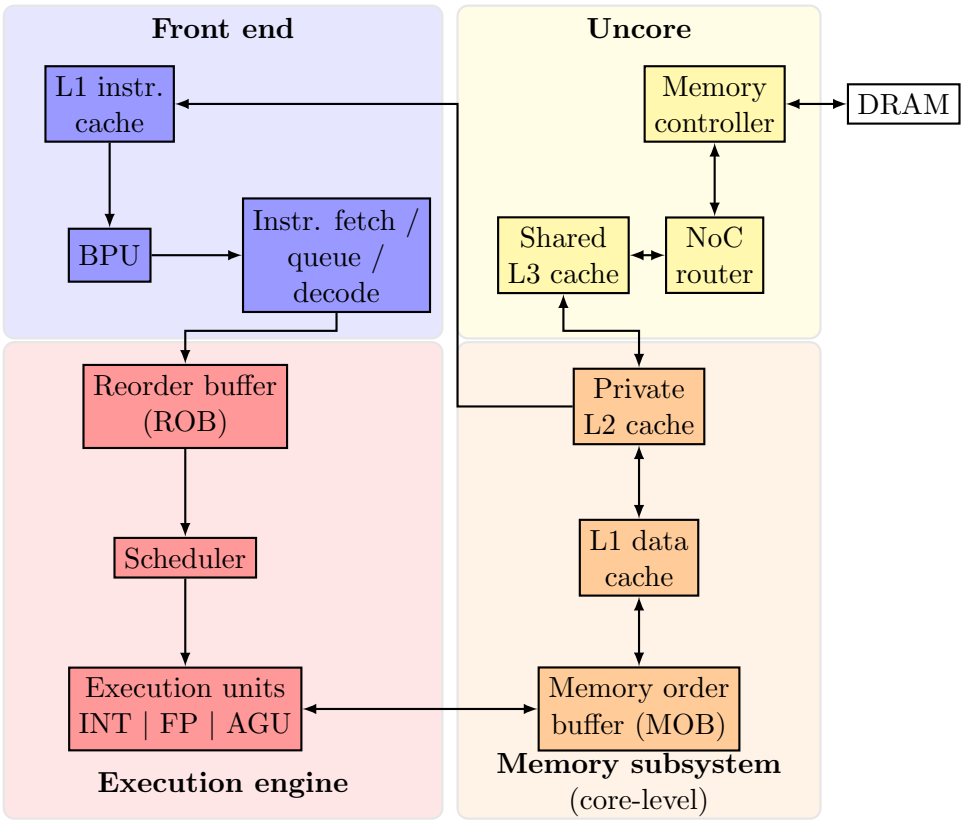
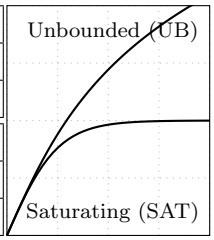


Figure 4.14: Overview of the core microarchitecture composed of front end, execution engine as well as memory subsystem including the private L1 and L2 caches as well as the core's connection with the processor uncore [4]

Table 4.18: Observed relationship between performance counter and core power for power modeling in regard to pos./neg. correlation sgn(β_i) and shape of the monotonically increasing/decreasing transformation functions [4]

A_i	**BPU**	**L2LI**	**L2LM**	**L2SI**	**L2SM**	/
sgn(β_i)	-	+	+	+	+	/
Slope	UB	SAT	SAT	UB	UB	/
A_i	**L3LI**	**L3LM**	**L3SI**	**L3SM**	**L2CLK**	**L3CLK**
sgn(β_i)	-	-	+	+	-	+
Slope	SAT	SAT	UB	UB	UB	UB



Front End (BPU): Increasing number of branch instructions contribute negatively to overall core power and this effect is nonlinear and unbounded. The core has to spend more time refilling the front end and the rate of subsequent stalls in the execution engine, seems to increase nonlinearly. Even adding additional BPU related performance counters will likely not change the nonlinear relationship.

Memory Subsystem on core-level (L2LI, L2SI, L2LM, L2SI, L2CLK): An increase of L2 load activity leads to saturating core power contributions. This is intuitive because L2 load misses can be masked at low L2 load rates but not anymore at high L2 load rates. Note, that L2 load rate and load miss rate are correlated, thus explaining the similar power behavior for both performance counters. The increasing L2 load rate and load miss rate lead to execution engine stalls and decreasing execution engine power and a saturating core power contribution. For L2 stores, the effect is still nonlinear but unbounded and can be explained by the Gainestown's store buffer being (only) double the size of the load buffer in the MOB unit. Most of the benchmarks show on average a load instruction rate which is more than twice the rate of store instructions, meaning that execution engine stalls do not saturate due to L2 store misses. For L2CLK, the activity of the front end and the execution engine saturates and starts to decrease while the memory subsystem activity keeps increasing with the percentage of lost cycles due to L2 load and store misses.

Uncore (L3LI, L3SI, L3LM, L3SI, L3CLK): Although the uncore switching activity does not contribute to core-level power consumption in the scope of this thesis, the uncore performance counters still correlate with core power and improve accuracy of the core power model. The nonlinear effect stays the same for L3 loads, but the indication for the core power contribution becomes negative as the likelihood of stalls increases significantly with L3 load misses. Overall, this does not mean that overall package power — including cores and the uncore — goes down with increasing L3 load misses, however, the core power itself decreases allowing resource management algorithms to adapt the resource mapping and scheduling. For L3 stores, the nonlinear effect is the same as for L2 stores. For L3CLK, the nonlinear behavior is the same as with L2CLK, however, the power response is negative as the possibility for masking the cache miss penalty is comparatively lower.

Summary This section introduces a nonlinear modeling approach based on nonlinear transformation functions to account for nonlinear performance counter / power relations in a model transparent way, in contrast to the more black box FFNN models of Section 4.2. By identifying and applying nonlinear transformations for specific performance counters, a nonlinear model is generated which estimates run-time power more accurately than the reference-based linear and polynomial approaches. Although the run-time overhead of this approach is higher compared to other lightweight models, i.e. linear and polynomial, it is sufficiently low to allow estimation rates of 10 kHz with relatively low area or computational overhead similar to the proposed FFNN-based methodology. Furthermore, the presented methodology does not rely on knowledge of microarchitectural implementation details. Finally, the generated nonlinear power models are fully

transparent, providing interpretability in regard to the causes of nonlinear performance counter / dynamic power relationships.

4.4 Comparison and Discussion of the Power Estimation Methodologies

This thesis proposes three different approaches for improving on fine-grained run-time power estimation models for multi-core systems:

- an ICA-based approach in Section 4.1,
- an FFNN-based approach in Section 4.2,
- a nonlinear transformation-based approach in Section 4.3.

These three approaches have been compared — in the respective section — to linear power models and polynomial power models which were based on state-of-the-art related works while also discussing their associated trade-offs. In this section, an overall comparison and discussion of the three different approaches will be given in regard to their estimation accuracy, their run-time overhead as well as other model properties, e.g. employed workloads for model generation and model transparency.

An overview of the relative RMSE errors of the different proposed power estimation models as well as two reference-based power models, one linear and one polynomial, are shown in Table 4.19. As described in Section 3.4, different distributions of the generic workloads (SPLASH-2 and PARSEC benchmarks) were used for final evaluation of the different power estimation models. These different distributions also apply for the reference-based power models, e.g. in Section 4.1 the synthetic workload-based linear power model was evaluated over all workloads while in Section 4.2 it was evaluated on the holdout data set. Therefore, there are slight deviations in the relative and absolute error performances of the different power models. To make the accuracy comparison as conservative as possible, the lowest relative error values of the reference-based power models are chosen for comparison in Table 4.19. Note, that in the table only those power model versions are added which were generated with unknown core-level power, i.e. derived from package power, and with a full complement of performance counter inputs as described in Section 3.2. Alos, no versions of the power models with reduced number of performance counter are included in this comparison nor those power models which were generated with known core-level power as model generation input. Thus the table provides a baseline comparison of all novel power estimation models presented in this thesis. First, one can observe that the accuracy of the ICA-based methodology to generate linear run-time power models is not significantly (2%) better than for linear or polynomial approaches from the related works. Also, the ICA-based methodology has the same accuracy as the proposed polynomial approach with variable degrees per performance counter input from Section 4.3. The aim for the ICA-based methodology was to minimize multicollinearity in an automated methodology, without reliance on synthetic workloads or manual performance counter transformations, while the estimation accuracy was only a secondary aim, i.e. keeping it similar to other approaches was deemed sufficient. However, the FFNN-based power estimation models were also purely trained on generic workloads and achieve significantly higher estimation accuracy than

Table 4.19: Average RMSE estimation errors of different run-time dynamic power models and their run-time computational overhead per model inference, e.g. for 10 kHz estimation rates every 0.1 ms, as well as the total memory to be stored on the processor for each model

Power Model	Relative Error	MAC OPs per Execution	Total Memory in kBit
ICA-based (Sec. 4.1)	10%	196	6.5
Shallow FFNN (Sec. 4.2)	**4.5%**	827	20
Multi-1 FFNN (Sec. 4.2)	5.3%	326	10
Multi-2 FFNN (Sec. 4.2)	7.2%	290	9.3
Multi-3 FFNN (Sec. 4.2)	7.9%	268	8.6
Multi-4 FFNN (Sec. 4.2)	6.1%	308	10
Nonlinear trans. (Sec. 4.3)	9.0%	377	1.4
Reference-based			
Linear (syn. workloads) [36]	12%	**14**	**0.5**
Polynomial degree-3 [30]	12%	112	1.5
Polynomial var. degree (Sec. 4.3)	10%	36	0.8

the ICA-based methodology which is in line with some coarse-grained NN works discussed in the related work which also did not use any specific methodologies to minimize multicollinearity. Overall, in regard to difficulties from multicollinearity, these results indicate that FFNN are a very powerful tool to inherently minimize the modeling errors and problems due to multicollinearity without the need for any additional methodologies like ICA transformations or synthetic workloads.

The FFNN-based approach as well as the nonlinear transformation-based approach both aimed to increase run-time estimation accuracy by accounting for potential nonlinear performance counter / power consumption relations. One can observe that the five final FFNN power estimation models, with different neural architectures, significantly outperform the nonlinear transformation approach in regard to the relative RMSE of the power estimations, i.e. 1.1% - 4.5% lower relative error than the nonlinear transformation model, 4.1% - 7.5% lower relative error than a purely linear power model and standard polynomial models of degree 3. However, the first optimized *shallow* neural architecture which was determined by single-objective optimization still required more than double the MAC operations for a single power estimation compared to the nonlinear transformation approach and more than 14 times as much memory to encode all neuron weights compared to encoding the nonlinear coefficients. The multi-objective optimization, which minimized both run-time overhead and power estimation error in parallel, of the neural architectures later yielded the *multi-x* FFNNs. These *multi-x* FFNNs all require slightly *less* MAC operations than the nonlinear transformation model while still requiring six to seven times more memory. For both approaches, memory requirements could be reduced in future works by carefully decreasing the bit-width of the coefficients

while trying to keep the estimation accuracy from degrading too much. Overall, the FFNN-based approach is likely the most suitable approach for many run-time power estimation applications for multi-core/many-core processors with complex core architectures. The only downside compared to the nonlinear transformation approach is the higher memory requirement as well as the non-transparency of the resulting power model. Where the nonlinear transformation model easily allows for investigation of nonlinear performance counter / power relations, the same investigation is more difficult for the trained FFNNs due to the multi-layered concept with sporadic activations of contributing estimation pathways in the FFNNs power models.

Finally, one can observe clear trade-offs between model complexity and estimation accuracy in the investigated power modeling approaches. In regard to fine-grained, run-time power estimation, there is no "free lunch" which allows for higher estimation accuracy while keeping run-time overhead static. The proposed power estimation models have shown that significantly higher model accuracy can be achieved for high estimation rates of 10 kHz while still having small area/power/computational overheads compared to large out-of-order core architectures. However, they do lead to higher run-time overhead for the power estimations and if/when to spend such overhead during the multi-/many-core design process is up to the processor designers and the integrated power and thermal management algorithms, i.e. how well these algorithms perform under partially incorrect run-time power information inputs. As discussed in Section 2.1.3, many of the state-of-the-art power and thermal management algorithms which use run-time power as an input to their decision process, assume perfectly accurate, fine-grained power information. That assumption is likely incorrect for most processor systems and future works could investigate the impact of power information uncertainty onto the correctness of power and thermal management decision making and thus the performance impact of inaccurate, run-time power information.

5 Novel and Accurate Power Forecasting Model based on LSTM RNNs

This section presents and evaluates the methodology for LSTM-based power forecasting [5].

Motivation and Problem Formulation As discussed in Section 2.1.4, forecasting future power states of a multi-/many-core system can enable more effective, *proactive* power and thermal management algorithms and power forecasting fundamentally belonging to the problem class of time-series forecasting. There are many challenges involved in the time-series forecasting of power on core-level. Although some AR-based and history table-based power forecasting methodologies were proposed in the related work, see Section 2.3, these have some fundamental limitations in regard to: either their capability to recognize power-relevant application patterns to be able to forecast power phase changes or to be sufficiently fine-grained in the time-domain.

In general, AR models — or variants thereof — are often-used simple time-dependent linear models. However, the recognition of patterns in the performance and power data, which enables the forecasting of power phase changes, can require both nonlinear modeling and the ability to dynamically discard or forget previous forecast values, e.g. one-time shocks. Another key drawback of AR-based models is the limited history encoding that prevents learning long-term dependencies and larger amounts of training data. The power and thermal management policy may change the Vf-levels at any time which imposes another challenge for power forecasting because power and performance counter traces may be observed at different Vf-levels. However, compared to similar problems, not only is the past power trace available as a feature for forecasts but the observed past performance counter data, which could allow for additional insights into the application behavior, i.e. the application pattern over time.

There are a very wide range of works including *performance forecasting* on multi-core processors where LSTM RNNs [126] have been successfully employed for time-series forecasting. These LSTM RNNs models overcome the limitations of AR-like models because:

- they are inherently able to learn nonlinear dependencies,
- they can learn/encode long-term dependencies with the help of an internal memory.

Therefore, this section presents a power forecasting methodology using LSTM RNNs which allows for fine-grained — in the time-domain as well as the spatial domain —

power forecasts and has a high capacity for power-relevant pattern recognition and thus can accurately forecast power phase changes.

The challenge of changing Vf-levels and its potential impact on model stability of the LSTM RNNs is tackled by normalizing both inputs and outputs of the models to the Vf-levels; allowing the use of a single model for all Vf-levels. However, there are several remaining challenges when employing LSTM RNNs for power forecasting. To achieve a high forecasting accuracy on untrained/unknown applications, i.e. generalizing the behavior from trained applications and their power behavior towards unknown applications, it is crucial that a single model is created that is usable for all applications and does not rely on prior application knowledge nor on inherent task periodicity. In addition, the time interval of forecasts, e.g. forecasting the next 1 ms or the next 10 ms, depends on the power or thermal management algorithm that makes use of the forecasts. For example, Vf-scaling operates much faster than task migration which results in very different requirements for the forecast time intervals. A forecasting technique should be easily adaptable to different commonly used power and thermal management decision periodicities. With 1 ms and 10 ms being common decision periodicities, these two values are chosen for investigation as forecasting time intervals. To summarize, the following novel contributions are made in this section:

- Forecasting future workload-dependent core-power consumption at run-time, considering large patterns of power and performance counters, with high accuracy.
- Full-compatibility with DVFS, comprising feature traces and forecasting outputs at different Vf-levels with a single model.
- Demonstrating that the LSTM RNN model generalizes to unseen applications with high accuracy.

In the following, a short overview of the problem formulation and notation specific to the power forecasting contributions is given. The near-future power consumption — which is to be forecast — is defined as the average power consumption within short future time intervals, i.e. in this thesis time intervals of 1 ms and 10 ms. The accuracy is measured with both the MAPE of the forecasts as well as the instantaneous Worst Case (WC) error over all individual forecasting time steps. The complexity of the LSTM RNN, e.g. number of layers and number of neurons per layer, is used as a proxy variable for the run-time overhead. The inputs to the forecasting technique are as throughout this thesis the observations of the current system state, i.e. power consumption values and performance counter readings. These power and performance counter observations can be obtained at a shorter period than the forecasting time intervals, i.e. observations every 10 us while the forecasting time intervals are 1 ms and 10 ms.

The following summarizes the problem formulation for this section:
Target: Forecast average core-level power consumption of the future time period τ.
Optimization Goal: Find a suitable trade-off between forecasting error (MAPE, WC error) and LSTM RNN complexity (number of layers and number of neurons per layer).
Available Inputs:

- Core-level power consumption values at observation period $\tau_M \leq \tau$
- Performance counter PC readings at observation period τ_M
- Employed Vf-levels at observation period τ_M
- Target Vf-level for forecasting

The remainder of this section is structured as follows. First, an overview of the methodology to generate power forecasting LSTM RNNs is given in Section 5.1 and the data preprocessing and hyperparameter optimization are presented in detail. Afterwards, the results in regard to the optimized neural architecture hyperparameters, the general forecasting accuracy compared to other state-of-the-art approaches and the accuracy in forecasting power phase changes are presented in Section 5.2.

5.1 Methodology for Generating LSTM RNN Power Model

First, an overview of the proposed methodology for generating LSTM RNNs for multi-core power forecasting is shown in Figure 5.1. The methodology is constituted of two distinct phases:

1. At design time, the power and performance counter traces are aggregated for a multitude of training benchmarks, the LSTM RNN architectures are optimized and the final LSTM RNN for power forecasting is trained.
2. At run-time, the LSTM RNN forecasts power for periods of τ.

The multi-core system, aggregation of power and performance counter traces and the executed benchmarks in this section are as specified in Chapter 3. In the following, all the remaining individual steps of the methodology will be discussed in detail.

Data Preprocessing

Due to the inherently large run-time overhead in regard to memory of LSTM RNNs of even medium complexity, it is highly desirable to obtain reusable power forecasting LSTM RNNs, i.e. the same LSTM RNN with the same neural architecture and the same neural weights needs to accurately forecast power at different Vf-levels. This means that the final LSTM RNN itself needs to be invariant in regard to voltage and frequency. Therefore, all performance counter inputs PC are — where necessary — normalized to the operating frequency of the current training and or evaluation data trace. Some

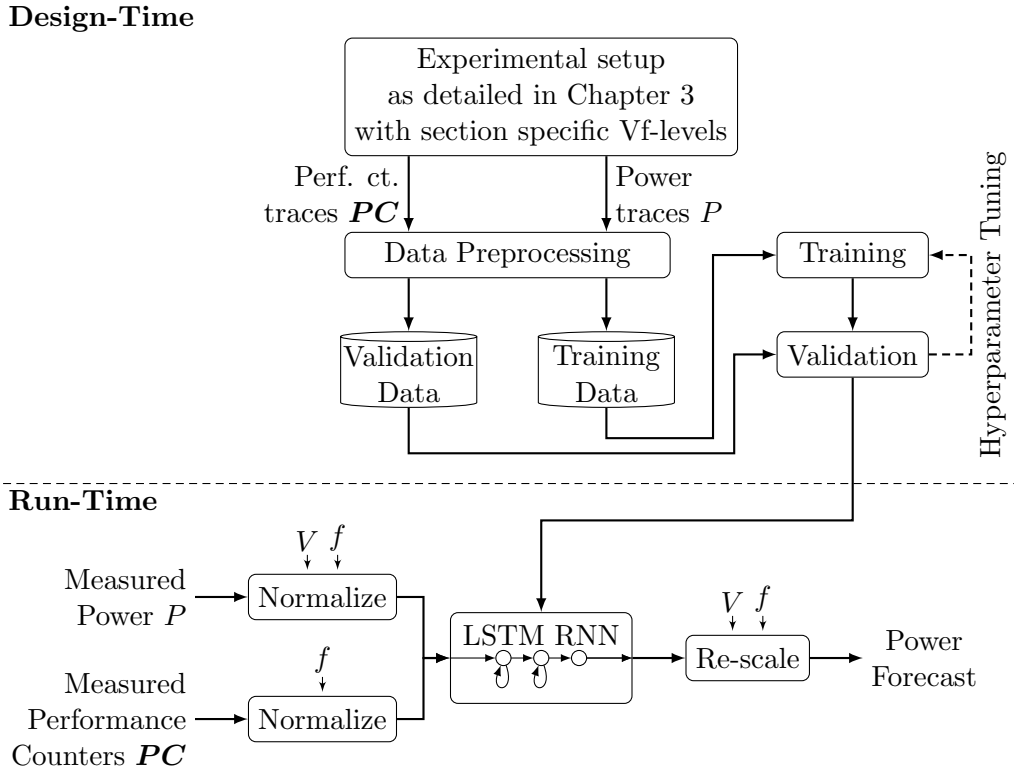


Figure 5.1: Overview of the proposed methodology to create and optimize the LSTM RNN at design-time and employ it at run-time [5]

performance counters like IPC are already normalized to the cycle length and thus normalized to the operating frequency. To avoid arithmetic underflows, the frequency normalization is done to the operating frequency in *GHz* and not in *Hz* for each discrete point k in time as follows:

$$PC_{inv}[k] = \frac{PC[k]}{f_{GHz}[k]}. \tag{5.1}$$

As discussed in Section 2.1.1, the dynamic power consumption values consist of a workload-dependent part and the Vf-dependent part. With this forecasting methodology aiming to forecast workload-dependent power changes, the Vf-dependent power contributions have to be removed from the power traces. This is also sensible as Vf-levels are set by power and thermal management algorithms which should use the forecasts as input. Therefore, by keeping Vf as contributor to the power consumption traces would train the LSTM RNNs to forecast decisions of the power and thermal management algorithms, thus creating an unintended control loop. The power traces are therefore normalized to the current operational Vf-level as

$$P_{inv}[k] = \frac{P_{dyn}[k]}{f_{GHz}[k] \cdot V^2[k]}. \tag{5.2}$$

Note that all power values are per default core-level power values. The underlying assumption for this is that a different, high accuracy run-time power estimation model is available to obtain core-level power consumption information as input for this power forecasting methodology.

In addition, the training data input for the power forecasts, i.e. the output variable the LSTM RNNs should generate during run-time, is generated in form of average power consumption over τ points in time as:

$$\bar{P}_{inv}[k+\tau] = \frac{1}{\tau}\sum_{i=k}^{k+\tau} P_{inv}[i]. \tag{5.3}$$

Afterwards, PC_{inv}, $\bar{P}_{inv}$ and P_{inv} are scaled to the range of $(0, 1)$ for more stable RNN weight updates.

Finally, the data traces of the benchmarks are pseudorandomly distributed into a training data set, a validation data set and a holdout data set. The pseudorandom distribution is used to avoid a clustering of similar benchmarks in the holdout data set, e.g. including both *fft* benchmarks from the SPLASH-2 and PARSEC benchmarks suites. Power and performance counter traces from an individual benchmark are only used for exactly one of the training / validation / holdout data sets, and never mixed. Thereby, the independence between the sets is fully ensured and there is no cross-contamination between data sets which would allow for the holdout data set to be known by the LSTM RNN before final evaluation. The training data set and the validation data set are used for hyperparameter tuning and for training the final power forecasting LSTM RNN. The holdout data set is solely used to assess the forecasting performance of the parameter-optimized and trained network in Section 5.2.

Hyperparameter Optimization

The goal of optimizing the neural hyperparameters, i.e. number of layers and number of neurons per layer, is to find LSTM RNN architectures which achieve high forecasting accuracy with the lowest possible model complexity. Aiming for low model complexity has the advantage of reducing the risk of having an architecture which is inclined to overfit the underlying training data, thus achieving lower accuracy on the holdout data. Also, low model complexity indirectly translates to low run-time overhead which is an added benefit of this strategy.

The input dimensionality of the LSTM RNN is the number of performance counters while the output dimensionality is the number of neurons in the last LSTM layer. These outputs from the last LSTM layer are fed into a dense layer with an output dimensionality of one which corresponds to the current power forecast. During training, the LSTM RNN is fed with batches of $PC_{inv}[k, k-1, ..., k-m]$ and $P_{inv}[k, k-1, ..., k-m]$ going back m discrete time points and with $\bar{P}_{inv}[k+\tau]$ as a target function. To achieve high forecasting accuracy while reducing run-time overhead, the hyperparameters have to be optimized through a DSE. For training the LSTM RNN, first the algorithmic hyperparameters — see also Section 4.2.1 for an overview of such NN hyperparameters,

are empirically determined by generating a set of LSTM RNNs with identical architecture and only varying a single algorithmic hyperparameter for training. The *Adam* optimizer [127] showed the highest forecasting accuracy compared to Stochastic Gradient Descent (SGD) and Nesterov Accelerated Gradient (NAG) optimizers. The maximum number of epochs were set to 10 for τ=1 and 25 for τ=10 where no accuracy improvement on the training data was observable for even very large network architectures with a hundred layers and a hundred neurons per layer. Mean-Square Error (MSE) was chosen as loss function due to its stability when training RNNs. In addition, early stopping is used in combination with model checkpointing to further avoid overfitting the training data. The stop patience is set at 5 epochs as a trade-off between potentially missing a higher accuracy LSTM RNN and overall training time. Thus, the training is stopped if the validation loss does not improve for 5 continuous training epochs. In such a case, the training output is the previous training instance of the LSTM RNN model with smallest loss. As activation function `ReLU` was chosen because it achieves the highest training accuracy and a negligible overhead over a set of different activation functions (`ReLU`, `tanh`, `sigmoid`).

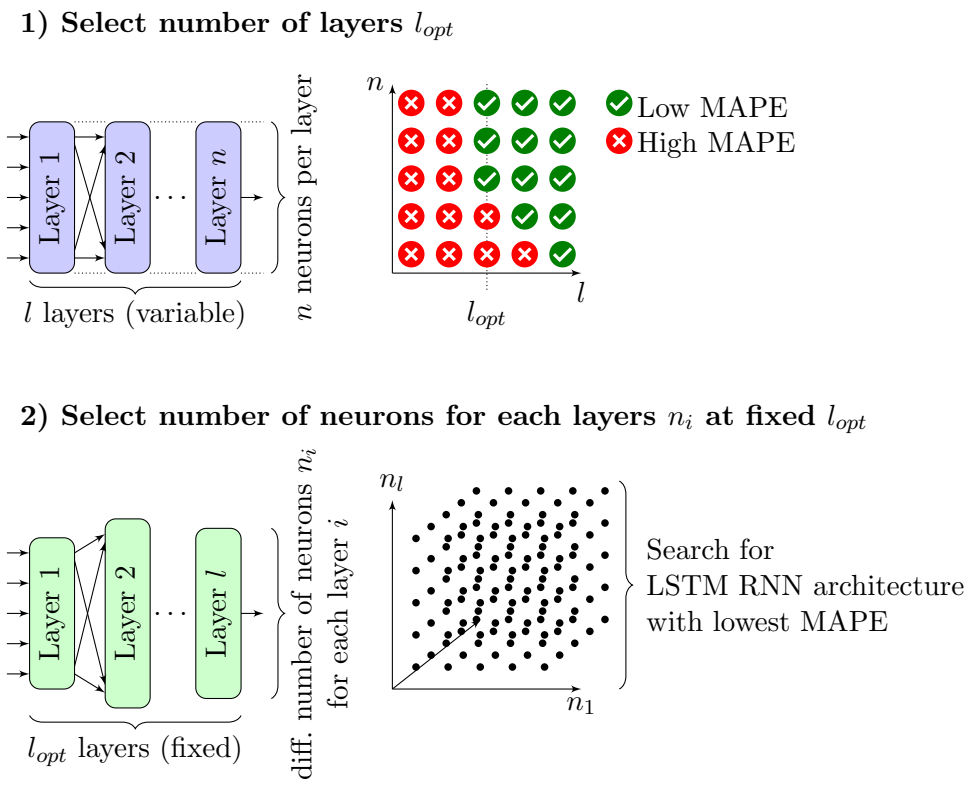


Figure 5.2: Selection of the final LSTM RNN neural hyperparameter architecture in a two-stage DSE. First the depth of the RNN (number of layers) l_{opt} is selected under the simplifications of having the same number of neurons per layer (in blue); then the number of neurons for each layer n_i is optimized for the selected l_{opt} (in green) [5].

The remaining model hyperparameters, i.e. number of layers and number of neurons per each layer, are determined through two successive heuristic grid-searches. A heuristic approach is proposed as an exhaustive parameter search over all possible combinations of layers and neurons per individual layer within normal constraint boundaries, would be computationally infeasible. As mentioned, the model hyperparameters have not only a significant impact on run-time overhead, but also the risk of overfitting. Therefore, the methodology tries to find hyperparameter values which are minimal while still generating networks with acceptable MAPE$\leq$10% and MAPE$\leq$30% values for the 1 ms and 10 ms forecasts, respectively. The overall methodology of the two grid-searches and their respective optimization goals are shown in Figure 5.2.

In the first-grid search, LSTM RNNs with $l \in \{1, 2, \ldots, 12\}$ number of layers and $n \in \{1, 2, \ldots, 20, 25, \ldots, 100\}$ neurons per layer are generated. The goal of this first grid-search is to determine a minimal viable layer count and minimal viable number of neurons per layer. To find these minimal viable neuron architecture numbers, a large set of LSTM RNN are generated along a two-dimensional grid with number of layers in one dimension and number of neurons in the other dimension. The resulting network architectures are always rectangular, i.e. they have always the same number of neurons per layer. To compare the forecasting accuracy of different networks, MAPE is used on the de-normalized power forecasts in the following as this metric is independent of the underlying average power levels, for example for different Vf-levels. In contrast, MSE is chosen as the loss function because the dependence on the power level is not relevant when training particular LSTM RNNs and MSE does not suffer from small denominators in like a MAPE loss function. The optimal layer count l_{opt} is chosen based on the MAPE distributions for the different number n neurons per layer. The smallest layer count for which an n_{min} exists where the MAPE values fall within the acceptable MAPE range for $n \geq n_{min}$ is then the fixed layer count l_{opt} for the second grid-search.

In addition, minimal viable neuron value n_{min} is used to constrain the minimum number of neurons per layer in the second grid-search. An additional parameter n_{max}, constraining the maximum number of neurons, is also determined based on the n values for which further increases yield no statistically significant improvements in the MAPE. A reasonable expectation is that different forecasting time-frames τ — in this proposed forecasting methodology: 1 ms and 10 ms — require different LSTM RNNs architectures. Therefore, a separate first-grid search is executed for each τ value to determine possibly different values of l_{opt}, n_{min} and n_{max}.

In the second grid-search, the aim is to find good neuron distributions for the fixed number of layers l_{opt}. Thus, the LSTM RNN architectures are *not* rectangular anymore. For each individual solution of architectural hyperparameters, 10 LSTM RNNs are generated to be able to average the error metrics and remove outliers for diverging models which allows to have more statistically reliable results for each investigated solution of hyperparameters. The second grid-search is again separately executed for each τ value. The median MAPE value is then computed over the generated LSTM RNN power forecasting models and the architecture hyperparameters with lowest median MAPE is chosen as final, optimized architecture for the run-time power forecasting LSTM RNN.

The final LSTM RNN for run-time deployment generates the dynamic power forecasts $\hat{P}_{dyn}$ as follows:

$$\hat{P}_{dyn}[k+\tau] = \texttt{LSTM RNN}(P_{inv}[k,...,k-l], \boldsymbol{PC}_{inv}[k,...,k-l]) \cdot f_{GHz}[k] \cdot V^2[k]. \quad (5.4)$$

The total core-level power forecast is obtained by adding the static / idle power which only depends on the Vf-level and current temperature $T[k]$ — all measurable during regular multi-core operation; and does not depend on the executed workload:

$$\hat{P}_{total}[k+\tau] = \hat{P}_{dyn}[k+\tau] + P_{static}(V[k], f[k], T[k]). \quad (5.5)$$

This concludes the power forecasting methodology and in the following the results of the experimental evaluation of this are presented.

5.2 Experimental Evaluation

The aim of this section is to assess the accurac of the proposed LSTM-based power forecasting model. Again, the experimental setup described in detail in Chapter 3 is used to generate the underlying power and performance data. First, the results in regard to the optimized neural architecture hyperparameters, i.e. l_{opt}, n_{min} and n_{max} as well as the final neural architecture, are shown. Afterwards, the general forecasting accuracy of the final power forecasting LSTM RNN is compared to two approaches which were based on the state-of-the-art. Finally, the forecasting behavior in regard to rapid power phase changes for the different forecasting models is discussed.

Hyperparameter Optimization Results

For a forecasting time interval $\tau = 1\,\text{ms}$, the first grid-search found an optimal layer count of $l_{opt} = 5$. The forecasting accuracy showed significant improvements starting at a number $n = 10$ of neurons per layer and diminishing improvements starting at around $n = 30$ which leveled off at $n = 60$. Based on this, in the second grid-search it was decided to investigate 5-layered architectures with number of neurons per layer $n_i \in \{10, 30, 60\}$. The reasoning for including $n_3 = 60$ is that a single larger layer or multiple larger layers might lead to overall smaller neuron numbers on the remaining layers, i.e. layers with $n = 10$ neurons. With this, $3^5 = 243$ different neural architectures are investigated in the second grid-search and overall 2430 different LSTM RNNs are generated. Of these 243 different neural architectures, a `30-60-30-30-10` neuron distribution showed the smallest median MAPE.

For a forecasting time-interval $\tau = 10\,\text{ms}$, an $l_{opt} = 3$ was determined and a minimal neuron per layer number of $n = 10$, however, the accuracy improvement for larger neuron numbers was far less distinct compared to the $\tau = 1\,\text{ms}$ time interval results. This means that there is far lower confidence at which maximum neuron per layer number n_{max} the improvements in modeling accuracy would level off and thus, were to cut the solution

space to avoid high model complexity. Therefore, a compromise was chosen were the allowed number of neurons per layer is a set $n_i \in \{10, 20, 40, 60, 80, 100\}$ were it can be reasonably assumed that the model complexity is sufficiently large. However, as discussed previously this incurs the risk of overfitting and lower run-time accuracy and simultaneously unnecessarily high run-time overhead. This approach leads to $6^3 = 216$ different neural architectures and overall 2160 generated different LSTM RNNs. The neural architecture with the smallest median MAPE of this solution space was `20-80-40`. Note that these are the LSTM layers, and that for all LSTM RNNs a final dense layer is implicitly included in the neural architecture / the power forecasting model.

From these results, one can see that for the magnitude larger forecasting time interval, the neural architecture tends towards fewer layers with more neurons per layer. This could imply that less patterns are observable over the time-domain while encoding patterns of the performance counter inputs of a single input time-frame. How this translates into forecasting accuracy will be shown in the following section.

For the reference of the reader: the training time for one LSTM is around one minute and overall training time for both grid-searches and the final evaluation is around 90 hours when using a Nvidia Titan RTX for accelerating the LSTM RNN training.

General Forecasting Accuracy

To assess the forecasting accuracy of the optimized LSTM RNN architectures, a 100 LSTM RNNs of the 1 ms and 10 ms architectures were trained using the training and validation data at 3 GHz operating frequency. This repeated generation of a 100 different models based on the same hyperparameters is the same approach as taken in Section 4.2 for the FFNN-based power estimation models. The goal is again to obtain statistically more robust, final accuracy assessments. The forecasting accuracy is determined by running inference on the holdout performance counter data and comparing the forecast power values $\hat{P}_{dyn}[k+\tau]$ with the actual average power values $\bar{P}_{dyn}[k+\tau]$. The forecasting accuracy is then compared to the following baseline techniques:

- history table forecasting based on [79]
- ARs forecasting based on [6]

For the AR forecasting methodology, the last four average core power values were used as input as those showed the highest forecasting accuracy when parameterizing the AR model for 1 ms and 10 ms forecast durations. The forecasting average and instantaneous worst-case errors for the power forecasting LSTM RNNs approach and the reference-based power forecasting models are given in Table 5.1. The error values are computed for all techniques on the same holdout benchmarks, i.e. data which was not used for training the LSTM RNNs. Note that the history table-based forecasting methodology [79] has to have observed the workloads and changing power phases at least once to be able to make a power phase forecast. Therefore, for this comparison and the accuracy evaluation, the history table-based model observes the holdout data once to generate power phase patterns such that it is able to forecast power.

For a forecasting time interval $\tau = 1\,\text{ms}$, the history table-based approach has significant intra-phase errors due to its identified power phase being of longer than 1 ms durations. The AR based approach shows lower MAPE than the proposed LSTM RNN methodology but higher instantaneous worst-case error. This higher instantaneous worst-case error could indicate worse forecasting capabilities in regard to power phase changes and will, therefore, be investigated in the next section.

For a forecasting time interval $\tau = 10\,\text{ms}$, all approaches have significant MAPE errors with the power forecasting LSTM RNN having smaller instantaneous worst-case errors. These worse forecasting accuracies for the longer forecasting time interval were to be expected as the workload-dependent power behavior naturally becomes more unpredictable for longer time intervals, i.e. forecasting uncertainty increases with the further one forecasts into the future.

Table 5.1: Forecasting MAPE values and instantaneous worst case-error values for the LSTM RNN approach and reference-based approaches [5]

Error	**History Table [79]**	**Autoregressive Model [6]**	**LSTM**
1ms Forecasting Period			
MAPE	30 %	8 %	10 %
Inst. WC	5.3 W	3.5 W	2.0 W
10ms Forecasting Period			
MAPE	31 %	24 %	25 %
Inst. WC	5.4 W	5.8 W	3.6 W

Overall, the history table-based approach seems to perform substantially worse than the AR-based and the LSTM RNN-based approaches; for the latter two approaches further in-depth investigations are needed to identify their specific advantages and disadvantages.

Accuracy in Forecasting Power Phase Changes

To highlight the differing forecasting capabilities of the AR-based and the LSTM RNN-based approaches, an example of power forecasts for $\tau = 1\,\text{ms}$ is given in Figure 5.3 for the PARSEC *facesim* benchmark which is part of the holdout data. The history table approach [79] is left out of the following investigation; although the approach correctly recognizes benchmarks and their power phase, however, due to its phase definition and core activity never reaching 0% within the benchmark, the forecast power level as an average over the whole power phase is very coarse-grained and any comparison towards the other two approaches rather trivial.

The first observation from Figure 5.3 is that the AR approach [6] distinctively lags behind the actual core power consumption. This is especially visible when the power consumption shows sudden changes, e.g. Phases I and II. However, it also effectively

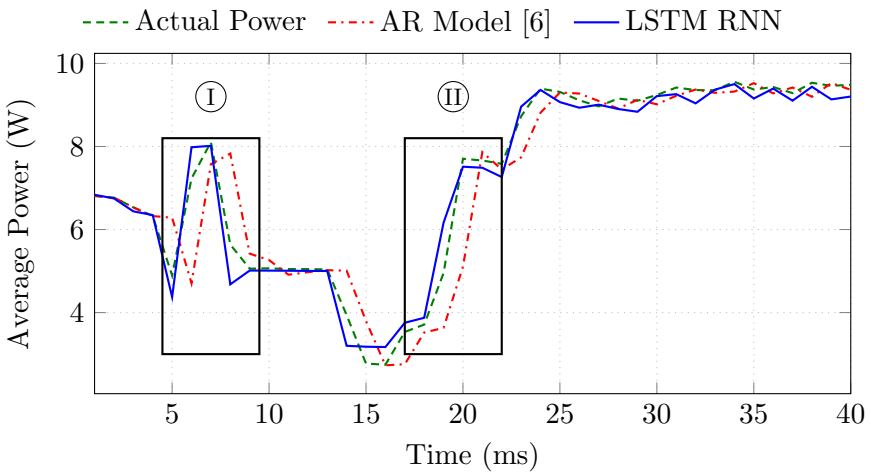


Figure 5.3: Power forecasts over time-frames of $\tau = 1\,\text{ms}$ by reference AR model [6] and the proposed LSTM RNN for the unseen PARSEC *facesim* benchmark [5].

adapts to constant power levels which is the main reason for the AR approach showing lower MAPE values compared to the proposed LSTM RNN approach.

Further investigation of the forecasting capabilities of the two different techniques is done by calculating the cross-correlation between the power trace and the forecast traces for the full traces of all five holdout benchmarks. Figure 5.4 shows the resulting cross-correlation values as well as a hypothetical oracle forecasting model that perfectly forecasts the power trace. The oracle model shows a peak at $\Delta t = 0$ samples as is to be expected. The correlation decreases slowly towards $\Delta t = \pm\infty$ samples because most of the time power values only change slightly. The proposed technique based on the LSTM RNN closely follows this trend, and — most importantly — also has a peak at $\Delta t = 0$ samples. This demonstrates that the LSTM RNN-based technique forecasts the next (future) power value and does *not* simply follow the measured (past) power trace. In contrast, the AR-based model shows a peak at $\Delta t = -1$ samples, i.e. the forecast value correlates most strongly with the last observed power value, and *not* with the actual value. This means that the AR-based model is very accurate for delayed tracking of the power values but limited in forecasting power phase changes.

Frequency-stability of the LSTM RNNs and Run-time Inference Overhead

Finally, the forecasting ability of the previously generated LSTM RNNs is investigated when used at 2 GHz and 1 GHz to assess how stable the LSTM RNNs are when they have been trained at a different frequency than they are currently making power forecasts. This is important because needing different LSTM RNNs for different operating frequencies would definitely be infeasible in regard to run-time overhead. The resulting MAPE instantaneous worst-case error values are given in Table 5.2 and show that the same LSTM RNNs can produce accurate power forecasts over a wide range of frequencies.

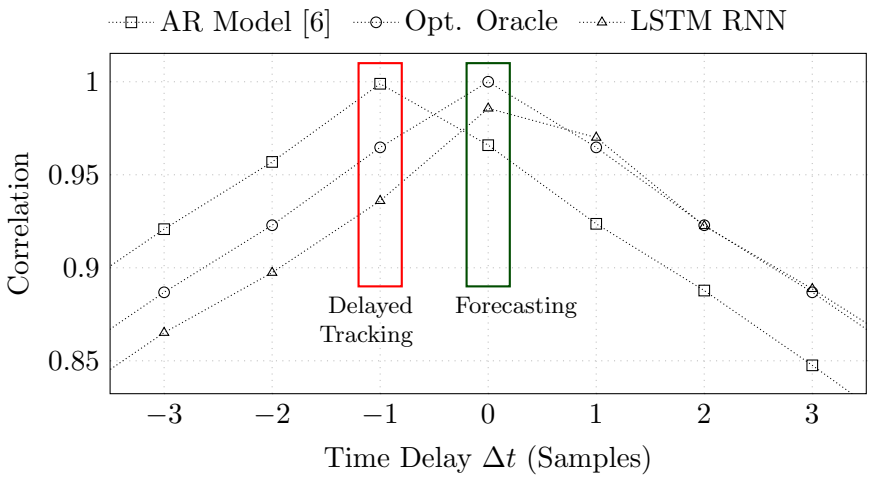


Figure 5.4: Cross-correlation of forecast and actual power trace demonstrating that the LSTM RNN model actually forecasts future power [5]. An optimal oracle shows a peak at time delay 0. The power forecasting LSTM RNN closely matches this, demonstrating that this model actually forecasts future power consumption. In contrast, the AR model shows a peak at time delay −1, indicating that the AR model mostly follows the measured power with a delay of one sample.

Table 5.2: Forecasting error for the LSTM RNNs trained at 3 GHz operating frequency and and used to forecast power at operating frequencies of 2 GHz and 1 GHz [5]

Error	**2 GHz**	**1 GHz**	**2 GHz**	**1 GHz**
	1 ms Forecasting Period		**10 ms Forecasting Period**	
MAPE	12 %	8.9 %	28 %	20 %
Inst. WC	1.9 W	0.4 W	2.3 W	0.7 W

The actual run-time inference overhead of the two different power forecasting LSTM RNNs architectures for time intervals of 1 ms and 10 ms was also investigated. Unfortunately, the required amount of MAC operations per inference were prohibitive for a potential micro-controller implementation at the target forecasting rates of 1 kHz. However, future solutions of dedicated NN on-chip accelerators or the trend towards far larger core architectures might lead to the run-time overhead becoming comparatively manageable.

Summary This section presents LSTM RNNs as a promising solution for the power forecasting problem, which is indispensable to enable proactive power and thermal management for on-chip systems. The proposed technique is able to accurately forecast the power consumption for unknown applications while minimizing LSTM RNN complexity, i.e. number of layers and number of neurons per layer. The experimental evaluation has shown the advantages of LSTM RNNs in regard to accurately forecasting power

phase changes. Nevertheless, there is still an open research question in regard to further run-time overhead minimization to make an actual integration of the LSTM RNN-based power forecasting approach feasible.

6 Conclusion and Outlook

The aims of this thesis were to propose methodologies to generate fine-grained, i.e. with least core-level spatial resolution and high estimation rates of 10 kHz, power estimation models with reasonable run-time estimation overhead and to investigate methodologies to automatically reduce input multicollinearity. In addition, power forecasting methodologies generating forecasts of future workload-dependent power consumption values with forecast intervals of 1 ms and 10 ms. Model outputs and evaluations of the power estimates / forecasts were constrained to the highly variable — due to its direct relation and dependence on highly variable user workloads — dynamic power portion of the total power consumption. As inputs to the estimation and forecasting models, a set of performance counters were used. For reducing input multicollinearity, a power model using ICA transformations was proposed which showed very good results in decreasing multicollinearity with slight improvements in estimation accuracy. To further increase estimation accuracy, FFNN-based power models as well as a methodology using nonlinear functions for input transformation were proposed with the aim to reduce and capture potential nonlinear relations between performance counter inputs and the dynamic power response.

The evaluation of these nonlinear power models shows decreases in relative estimation error between 3.0% - 7.5% compared to a purely linear reference-based power model and with the FFNN power models showing the highest decreases in estimation error. All generated ICA-based and nonlinear power models have sufficiently low additional run-time overheads to be used as run-time power estimators for large core architectures with the FFNN power models having similar computational overhead to the nonlinear transformation model, but also significantly higher memory demands. Overall, these results highlight an existing trade-off between power model complexity / run-time overhead and the power model estimation accuracy for which processor designers have to determine what level of complexity and accuracy offers the optimal solution for a given processor architecture and its use-cases. The power forecasting LSTM RNNs showed high accuracy for power phase changes, but lower average forecast accuracy compared to an AR-based reference model for both time intervals and is thus more applicable for reactive power and thermal management algorithms where high phase change accuracy is needed, e.g. to avoid thermal emergencies.

The limitations of this thesis are the usage of a simulation framework for generation and evaluation of the power estimating and forecasting models which could impact the results if the simulated data is distorting the different power models in different ways, i.e. if the simulation data unknowingly favors one type of power model over another type of power model. Another limitation is in the power forecasting LSTM RNNs due to the underlying complexity of the final, optimized neural architectures which still

require significant run-time overheads in relation to current core architectures. Modeling the *uncore* power consumption of the processor was not scope of this thesis, however, this introduces modeling inaccuracy for the core-level power models themselves. When generating power models with simulated core power as input, the percentage error of the core-level power models decreased by 2% which indicates that some of the dynamic uncore power was incorrectly attributed towards the cores. Finally, the all power models focused on the workload-dependency, i.e. instruction flow, of the power consumption and did not account for the processed data-content — due to data-content performance counters not being routinely available in processors. This likely explains some of the remaining power modeling inaccuracy.

In conclusion, the proposed power estimation models can enable more accurate decision making of reactive power and thermal management algorithms in general, and the proposed power forecasting model can enable specific proactive power and thermal management algorithms which require high phase change accuracy.

Future Work There are open research directions which could build upon the contributions of this thesis. For one, many power and thermal management algorithms proposed in the last decade have an implicit assumption of perfectly accurate, fine-grained run-time power information. Defining an explicit stochastic error variance for the inputs of such management algorithms, would enable research into the robustness of the algorithms' decision making under uncertainty. It would also allow a quantification of the potential performance losses due to inaccurate power information inputs and would give processor designers indications where the optimal trade-off between power model complexity and the associated overhead, and estimation accuracy is to be found. To further improve power modeling accuracy, an investigation into explicit uncore power consumption modeling as well as novel data-content toggle rate performance counters seems promising. Also, methodologies to accurately determine fine-grained, i.e. core-level, power consumption of existing off-the-shelf multi-/many-core processors — even through invasive instrumentation — are lacking. Any such novel methodology allowing for core-level instrumentation would also allow for an accurate reevaluation of the proposed methodologies in this thesis on physical hardware. Finally, further optimizations of the neural architectures for the power forecasting LSTM RNNs are needed to reduce the run-time inference overhead. With the inference and training costs of NNs becoming an ever bigger concern, new and promising methodologies like binarization / ternarization of neuron weights have been proposed in the wider deep-learning community which could be investigated for run-time, low-overhead neural architectures for power forecasting on multi-/many-core processors.

Bibliography

[1] M. Sagi, N. A. V. Doan, T. Wild, and A. Herkersdorf. Multicore power estimation using independent component analysis based modeling. In *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 38–45, 2019. doi:10.1109/MCSoC.2019.00013.

[2] M. Sagi, N. A. V. Doan, N. Fasfous, T. Wild, and A. Herkersdorf. Fine-grained power modeling of multicore processors using FFNNs. In A. Orailoglu, M. Jung, and M. Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation - 20th International Conference, SAMOS 2020, Samos, Greece, July 5-9, 2020, Proceedings*, volume 12471 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-60939-9_13, doi:10.1007/978-3-030-60939-9_13.

[3] M. Sagi, N. A. V. Doan, F. Nael, T. Wild, and A. Herkersdorf. Fine-grained power modeling of multicore processors using FFNNs. *International Journal of Parallel Programming*, 50(tbd):243–266, 2022. doi:10.1007/s10766-022-00730-9.

[4] M. Sagi, N. A. V. Doan, M. Rapp, T. Wild, J. Henkel, and A. Herkersdorf. A lightweight nonlinear methodology to accurately model multicore processor power. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3152–3164, 2020. doi:10.1109/TCAD.2020.3013062.

[5] M. Sagi, M. Rapp, H. Khdr, Y. Zhang, N. Fasfous, N. A. Vu Doan, T. Wild, J. Henkel, and A. Herkersdorf. Long short-term memory neural network-based power forecasting of multi-core processors. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1685–1690, 2021. doi:10.23919/DATE51398.2021.9474028.

[6] P. D. Saj Manoj, A. Jantsch, and M. Shafique. SmartDPM: Machine Learning-Based Dynamic Power Management for Multi-Core Microprocessors. *Jrnl. of Low Power Electronics (JOLPE)*, 14(4), 2019.

[7] G. E. Moore. Cramming more components onto integrated circuits (Reprinted from Electronics, pg 114-117, April 19, 1965). Technical Report 1, 1965. URL: papers3://publication/uuid/8E5EB7C8-681C-447D-9361-E68D1932997D, doi:10.1109/N-SSC.2006.4785860.

[8] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of Ion-Implanted MOSFET's With Very Small Physi-

cal Dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974. doi:10.1109/JSSC.1974.1050511.

[9] L. J. Flynn. Intel halts development of 2 new microprocessors, 2004. URL: https://www.nytimes.com/2004/05/08/business/intel-halts-development-of-2-new-microprocessors.html.

[10] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011.

[11] J. Henkel, H. Khdr, S. Pagani, and M. Shafique. New trends in dark silicon. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015. doi:10.1145/2744769.2747938.

[12] M. Belwal and T. S. B. Sudarshan. A survey on design space exploration for heterogeneous multi-core. In *2014 International Conference on Embedded Systems (ICES)*, pages 80–85, 2014. doi:10.1109/EmbeddedSys.2014.6953095.

[13] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondi, G. Mariani, H. Posadas, E. Villar, C. Wu, F. Dongrui, Z. Hao, and T. Shibin. Multicube: Multi-objective design space exploration of multi-core architectures. In N. Voros, A. Mukherjee, N. Sklavos, K. Masselos, and M. Huebner, editors, *VLSI 2010 Annual Symposium*, pages 47–63, Dordrecht, 2011. Springer Netherlands.

[14] S. Pagani, S. M. PD, A. Jantsch, and J. Henkel. Machine Learning for Power, Energy, and Thermal Management on Multi-Core Processors: A Survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2018. doi:10.1109/TCAD.2018.2878168.

[15] M. Rapp, H. Amrouch, M. C. Wolf, and J. Henkel. Machine Learning Techniques to Support Many-Core Resource Management: Challenges and Opportunities. In *Workshop on Machine Learning for CAD (MLCAD)*. ACM/IEEE, 2019.

[16] A. K. Singh, S. Dey, K. R. Basireddy, K. McDonald-Maier, G. V. Merrett, and B. M. Al-Hashimi. Dynamic Energy and Thermal Management of Multi-Core Mobile Platforms: A Survey. *IEEE Design & Test*, 2020.

[17] P. Michaud. Exploiting thermal transients with deterministic turbo clock frequency. *IEEE Computer Architecture Letters*, 19(1):43–46, 2020. doi:10.1109/LCA.2020.2983920.

[18] D. Lo and C. Kozyrakis. Dynamic management of TurboMode in modern multi-core chips. *Proceedings - International Symposium on High-Performance Computer Architecture*, pages 1–11, 2014. doi:10.1109/HPCA.2014.6835969.

[19] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-management architecture of the intel microarchitecture code-named Sandy Bridge. *IEEE Micro*, 32(2):20–27, 2012. doi:10.1109/MM.2012.12.

[20] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel. TSP: Thermal Safe Power - Efficient power budgeting for Many-Core Systems in Dark Silicon. *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis - CODES '14*, (April):1–10, 2014. URL: http://dl.acm.org/citation.cfm?doid=2656075.2656103, doi:10.1145/2656075.2656103.

[21] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(11):1677–1690, 2011. doi:10.1109/TCAD.2011.2161308.

[22] M. Floyd, M. Allen-Ware, K. Rajamani, B. Brock, C. Lefurgy, A. J. Drake, L. Pesantez, T. Gloekler, J. A. Tierno, P. Bose, and A. Buyuktosunoglu. Introducing the adaptive energy management features of the power7 chip. *IEEE Micro*, 31(2):60–75, 2011. doi:10.1109/MM.2011.29.

[23] A. Pathania and J. Henkel. HotSniper: Sniper-Based Toolchain for Many-Core Thermal Simulations in Open Systems. *IEEE Embedded Systems Letters (ESL)*, 2018. doi:10.1109/LES.2018.2866594.

[24] S. Bhagavatula and B. Jung. A power sensor with 80ns response time for power management in microprocessors. In *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, pages 1–4, 2013. doi:10.1109/CICC.2013.6658487.

[25] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An Energy Efficiency Feature Survey of the Intel Haswell Processor. *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015*, pages 896–904, 2015. doi:10.1109/IPDPSW.2015.70.

[26] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg. Energy efficiency aspects of the amd zen 2 architecture. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 562–571, 2021. doi:10.1109/Cluster48925.2021.00087.

[27] M. Chadha, T. Ilsche, M. Bielert, and W. E. Nagel. A statistical approach to power estimation for x86 processors. In *IEEE Proc. IPDPSW*, 2017. doi:10.1109/IPDPSW.2017.98.

[28] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett. Accurate and stable run-time power modeling for mobile and embedded CPUs. *IEEE Trans. CAD*, 2017. doi:10.1109/TCAD.2016.2562920.

[29] C. Isci and M. Martonosi. Identifying program power phase behavior using power vectors. *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, pages 108–118, 2003. doi:10.1109/WWC.2003.1249062.

[30] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, page 12, USA, 2011. USENIX Association.

[31] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. Invasive manycore architectures. In *17th Asia and South Pacific Design Automation Conference*, pages 193–200, 2012. doi:10.1109/ASPDAC.2012.6164944.

[32] N. Anantharajaiah, T. Asfour, M. Bader, L. Bauer, J. Becker, S. Bischof, M. Brand, H.-J. Bungartz, C. Eichler, K. Esper, J. Falk, N. Fasfous, F. Freiling, A. Fried, M. Gerndt, M. Glaß, J. Gonzalez, F. Hannig, C. Heidorn, J. Henkel, A. Herkersdorf, B. Herzog, J. John, T. Hönig, F. Hundhausen, H. Khdr, T. Langer, O. Lenke, F. Lesniak, A. Lindermayr, A. Listl, S. Maier, N. Megow, M. Mettler, D. Müller-Gritschneder, H. Nassar, F. Paus, A. Pöppl, B. Pourmohseni, J. Rabenstein, P. Raffeck, M. Rapp, S. N. Rivas, M. Sagi, F. Schirrmacher, U. Schlichtmann, F. Schmaus, W. Schröder-Preikschat, T. Schwarzer, M. B. Sikal, B. Simon, G. Snelting, J. Spieck, A. Srivatsa, W. Stechele, J. Teich, F. Turan, I. A. C. Ureña, I. Verbauwhede, D. Walter, T. Wild, S. Wildermann, M. Wille, M. Witterauf, and L. Zhang. *Invasive Computing*. FAU University Press, 2022. doi:10.25593/978-3-96147-571-1.

[33] M. Rapp, M. Sagi, A. Pathania, A. Herkersdorf, and J. Henkel. Power- and cache-aware task mapping with dynamic power budgeting for many-cores. *IEEE Transactions on Computers*, 69(1):1–13, 2020. doi:10.1109/TC.2019.2935446.

[34] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[35] C. Möbius, W. Dargie, and A. Schill. Power consumption estimation models for processors, virtual machines, and servers. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1600–1614, 2014. doi:10.1109/TPDS.2013.183.

[36] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. A systematic methodology to generate decomposable and responsive power models for cmps. *IEEE Transactions on Computers*, 62(7):1289–1302, 2013. doi:10.1109/TC.2012.97.

[37] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock. Accurate fine-grained processor power proxies. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 224–234, 2012. doi:10.1109/MICRO.2012.29.

[38] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter. Architecting for power management: The IBM® POWER7™ approach. *HPCA 16 2010 The Sixteenth International Symposium on HighPerformance Computer Architecture*, pages 1–11, 2010. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5416627`, `doi:10.1109/HPCA.2010.5416627`.

[39] R. Kalla, Balaram Shnharoy, W. J. Starke, and M. Floyd. POWER 7 : IBM ' S Next-Generation Server Processor. *IEEE Micro*, 2010.

[40] S. Bhagavatula and B. Jung. A low power real-time on-chip power sensor in 45-nm soi. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(7):1577–1587, 2012. `doi:10.1109/TCSI.2011.2177129`.

[41] J. Henkel, H. Khdr, and M. Rapp. Smart Thermal Management for Heterogeneous Multicores. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 132–137. IEEE, 2019.

[42] E. Cai and D. Marculescu. Tei-turbo: temperature effect inversion-aware turbo boost for finfet-based multi-core systems. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 500–507, 2015. `doi:10.1109/ICCAD.2015.7372611`.

[43] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. *Int. Symp. on Computer Architecture (ISCA)*, 23(2):24–36, 1995.

[44] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha. A heuristic machine learning-based algorithm for power and thermal management of heterogeneous mpsocs. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 291–296, 2015. `doi:10.1109/ISLPED.2015.7273529`.

[45] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton. Haswell: The fourth-generation intel core processor. *IEEE Micro*, 34(2):6–20, 2014. `doi:10.1109/MM.2014.10`.

[46] Z. Chen and D. Marculescu. Distributed Reinforcement Learning for Power Limited Many-Core System Performance Optimization. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1521–1526. IEEE, 2015.

[47] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors : Methodology and Empirical Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, number December, 2003.

[48] C. Isci and M. Martonpsi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings - International Symposium on High-Performance Computer Architecture*, 2006. doi:10.1109/HPCA.2006.1598119.

[49] W. Bircher and J. Law. Effective use of performance monitoring counters for run-time prediction of power. *University of Texas at ...*, Tech Repor, 2004. URL: http://lca.ece.utexas.edu/pubs/UT{_}LCA{_}TR{_}041104-01.pdf.

[50] W. Bircher, M. Valluri, J. Law, and L. John. Runtime identification of microprocessor energy saving opportunities. *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, pages 275–280, 2005. doi:10.1109/LPE.2005.195527.

[51] W. L. Bircher and L. K. John. Complete system power estimation: A trickle-down approach based on performance events. *ISPASS 2007: IEEE International Symposium on Performance Analysis of Systems and Software*, pages 158–168, 2007. doi:10.1109/ISPASS.2007.363746.

[52] W. L. Bircher and L. K. John. Complete system power estimation using processor performance events. *IEEE Trans. Comput.*, 2012. doi:10.1109/TC.2011.47.

[53] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and Responsive Power Models for Multicore Processors using Performance Counters Categories and Subject Descriptors. *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 147–158, 2010. doi:10.1145/1810085.1810108.

[54] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang. PPEP: Online performance, power, and energy prediction framework and DVFS space exploration. *IEEE/ACM Proc. MICRO*, 2014. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7011408, doi:10.1109/MICRO.2014.17.

[55] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. *2010 International Conference on Green Computing, Green Comp 2010*, pages 135–146, 2010. doi:10.1109/GREENCOMP.2010.5598313.

[56] B. Goel and S. A. McKee. A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 273–282, 2016.

[57] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, L. Yang, L. Zhangt, R. P. Dickt, Z. Qiant, Z. M. Maot, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power

model generation for smartphones. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 105–114, 2010. URL: `http://dl.acm.org/citation.cfm?id=1878982`, `doi:10.1145/1878961.1878982`.

[58] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman. System-level power estimation tool for embedded processor based platforms. In *ACM Proc. RAPIDO*, 2014. `doi:10.1145/2555486.2555491`.

[59] M. J. Walker, A. K. Das, G. V. Merrett, and B. M. Al-hashimi. Run-time Power Estimation for Mobile and Embedded Asymmetric Multi-Core CPUs. In *HIPEAC Workshop on Energy Efficiency with Heterogenous Computing, Amsterdam, NL, 19 - 21 Jan 2015*, 2015.

[60] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, and U. Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Trans. TODAES*, 2020.

[61] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. Power-performance modeling on asymmetric multi-cores. In *IEEE Proc. CASES*, 2013. `doi:10.1109/CASES.2013.6662519`.

[62] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 189–194, 2010. `doi:10.1145/1840845.1840883`.

[63] I. Corporation. Intel® 64 and ia-32 architectures software developer's manual. Technical report, 2016.

[64] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring energy consumption for short code paths using rapl. *SIGMETRICS Perform. Eval. Rev.*, 40(3):13–17, jan 2012. URL: `https://doi.org/10.1145/2425248.2425252`, `doi:10.1145/2425248.2425252`.

[65] I. Advanced Micro Devices. Bios and kernel developer's guide (bkdg) for amd family 16h models 30h-3fh processors. Technical report, 2016.

[66] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204, 2013. `doi:10.1109/ISPASS.2013.6557170`.

[67] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das. Apollo: An automated power modeling framework for runtime power introspection in high-volume commercial microproces-

sors. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 1–14, New York, NY, USA, 2021. Association for Computing Machinery. URL: https://doi.org/10.1145/3466752.3480064, doi:10.1145/3466752.3480064.

[68] C. H. Hsu and S. W. Poole. Power signature analysis of the SPECpower-ssj2008 benchmark. In *IEEE Proc. ISPASS*, 2011. doi:10.1109/ISPASS.2011.5762739.

[69] G. Dhiman, K. Mihic, and T. Rosing. A system for online power prediction in virtualized environments using Gaussian mixture models. In *IEEE Proc. DAC*, 2010. doi:10.1145/1837274.1837478.

[70] W. Dargie. A stochastic model for estimating the power consumption of a processor. *IEEE Transactions on Computers*, 64(5):1311–1322, 2015. doi:10.1109/TC.2014.2315629.

[71] L. F. Cupertino, G. Da Costa, and J. M. Pierson. Towards a generic power estimator. *Computer Science - Research and Development*, 2014.

[72] L. Cupertino, G. Da Costa, A. Oleksiak, W. Piątek, J. M. Pierson, J. Salom, L. Sisó, P. Stolf, H. Sun, and T. Zilio. Energy-efficient, thermal-aware modeling and simulation of data centers: The CoolEmAll approach and evaluation results. *Ad Hoc Networks*, 25(PB):535–553, 2015. doi:10.1016/j.adhoc.2014.11.002.

[73] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky. Improving the accuracy of energy predictive models for multicore cpus using additivity of performance monitoring counters. In *Parallel Computing Technologies*, 2019.

[74] W. Lin, G. Wu, X. Wang, and K. Li. An Artificial Neural Network Approach to Power Consumption Model Construction for Servers in Cloud Data Centers. *IEEE Transactions on Sustainable Computing*, 2019.

[75] W. Wu, W. Lin, L. He, G. Wu, and C.-H. Hsu. A Power Consumption Model for Cloud Servers Based on Elman Neural Network. *IEEE Transactions on Cloud Computing*, 2019.

[76] X. Chen, C. Xu, R. P. Dick, and Z. M. Mao. Performance and power modeling in a multi-programmed multi-core environment. In *Proceedings of the 47th Design Automation Conference*, DAC '10, page 813–818, New York, NY, USA, 2010. Association for Computing Machinery. URL: https://doi.org/10.1145/1837274.1837479, doi:10.1145/1837274.1837479.

[77] B. Dutta, V. Adhinarayanan, and W.-c. Feng. GPU Power Prediction via Ensemble Machine Learning for DVFS Space Exploration. In *International Conference on Computing Frontiers (CF)*, pages 240–243, 2018.

[78] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2013. doi:10.1145/2445572.2445577.

[79] W. L. Bircher and L. John. Predictive Power Management for Multi-Core Processors. In *International Conference on Computer Architecture (ISCA)*, page 243–255. ACM, 2010.

[80] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 359–370, 2006.

[81] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. Dynamic GPGPU Power Management using Adaptive Model Predictive Control. In *High Performance Computer Architecture (HPCA)*, pages 613–624. IEEE, 2017.

[82] A. K. Coskun, T. S. Rosing, and K. C. Gross. Utilizing predictors for efficient thermal management in multiprocessor socs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1503–1516, 2009. doi:10.1109/TCAD.2009.2026357.

[83] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload Prediction using ARIMA Model and its Impact on Cloud Applications' QoS. *Transactions on Cloud Computing (TCC)*, 3(4):449–458, 2014.

[84] S. J. Tarsa, A. P. Kumar, and H. Kung. Workload Prediction for Adaptive Power Scaling using Deep Learning. In *International Conference on IC Design & Technology (ICICDT)*, pages 1–5. IEEE, 2014.

[85] M. G. Moghaddam, W. Guan, and C. Ababei. Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, 2017. doi:10.1109/IGCC.2017.8323597.

[86] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, S. Gumussoy, and U. Y. Ogras. An Online Learning Methodology for Performance Modeling of Graphics Processors. *IEEE Transactions on Computers (TC)*, 67(12):1677–1691, 2018.

[87] M. Rapp, A. Pathania, T. Mitra, and J. Henkel. Neural Network-based Performance Prediction for Task Migration on S-NUCA Many-Cores. *IEEE Transactions on Computers*, 2020.

[88] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing. P^4: Phase-Based Power/Performance Prediction of Heterogeneous Systems via Neural Networks. In

International Conference on Computer-Aided Design (ICCAD), pages 683–690. IEEE, 2017.

[89] C. V. Li, V. Petrucci, and D. Mossé. Predicting Thread Profiles Across Core Types via Machine Learning on Heterogeneous Multiprocessors. In *Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 56–62. IEEE, 2016.

[90] C. V. Li, V. Petrucci, and D. Mossé. Exploring machine learning for thread characterization on heterogeneous multiprocessors. *SIGOPS Oper. Syst. Rev.*, 51(1):113–123, sep 2017. URL: `https://doi.org/10.1145/3139645.3139664`, `doi:10.1145/3139645.3139664`.

[91] A. Iranfar, W. S. De Souza, M. Zapater, K. Olcoz, S. X. de Souza, and D. Atienza. A Machine Learning-Based Framework for Throughput Estimation of Time-Varying Applications in Multi-Core Servers. In *International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 211–216. IEEE, 2019.

[92] J. Kong, S. W. Chung, and K. Skadron. Recent thermal management techniques for microprocessors. *ACM Comput. Surv.*, 44(3), jun 2012. URL: `https://doi.org/10.1145/2187671.2187675`, `doi:10.1145/2187671.2187675`.

[93] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras. STAFF: Online Learning with Stabilized Adaptive Forgetting Factor and Feature Selection Algorithm. In *Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[94] D. De Sensi. Predicting performance and power consumption of parallel applications. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 200–207, 2016.

[95] C. Ababei and M. G. Moghaddam. A survey of prediction and classification techniques in multicore processor systems. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1184–1200, 2019. `doi:10.1109/TPDS.2018.2878699`.

[96] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2008. `doi:10.1145/1454115.1454128`.

[97] Y. Samei and R. Dömer. Automated estimation of power consumption for rapid system level design. In *IEEE IPCCC*, 2014.

[98] A. Asad, A. Dorostkar, and F. Mohammadi. A novel power model for future heterogeneous 3d chip-multiprocessors in the dark silicon age. *EURASIP Journal on Embedded Systems*, 2018(1):1–16, 2018.

[99] F. Oboril, J. Ewert, and M. B. Tahoori. High-resolution online power monitoring for modern microprocessors. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 265–268, 2015. `doi:10.7873/DATE.2015.0208`.

[100] Y. Li and P. Zhou. An outlier detection method and its application to multicore-chip power estimation. In *2017 IEEE 12th International Conference on ASIC (ASICON)*, pages 460–463, 2017. doi:10.1109/ASICON.2017.8252513.

[101] S. Van den Steen, S. De Pestel, M. Mechri, S. Eyerman, T. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout. Micro-architecture independent analytical processor performance and power modeling. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 32–41, 2015. doi:10.1109/ISPASS.2015.7095782.

[102] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, page 52. ACM, 2011. doi:10.1145/2063384.2063454.

[103] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan. HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design. *Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006. doi:10.1109/TVLSI.2006.876103.

[104] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. M. D. Hill, D. A. D. A. Wood, B. Beckmann, G. Black, S. K. S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, A. Basil, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. M. D. Hill, and D. A. D. A. Wood. The gem5 Simulator. *Computer Architecture News*, 39(2):1, 2011. URL: http://www.engineeringvillage.com/blog/document.url?mid=inspec{_}ef5502133370e5bf3M6a022061377553{&}database=ins{%}5Cnhttps://www.engineeringvillage.com/blog/document.url?mid=inspec{_}ef5502133370e5bf3M6a022061377553{&}database=ins{%}5Cnhttp://dl.acm.org/citation.cfm?doi, doi:10.1145/2024716.2024718.

[105] D. Genbrugge, S. Eyerman, and L. Eeckhout. Interval simulation: Raising the level of abstraction in architectural simulation. *Hpca*, pages 1–12, 2010. doi:10.1109/HPCA.2010.5416636.

[106] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. An Evaluation of High-Level Mechanistic Core Models. *ACM TACO*, 2014.

[107] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks. Quantifying sources of error in mcpat and potential impacts on architectural studies. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 577–589, 2015. doi:10.1109/HPCA.2015.7056064.

[108] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu. Mcpat-calib: A microarchitecture power modeling framework for modern cpus. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021. doi:10.1109/ICCAD51958.2021.9643508.

[109] M. LeBeane, J. H. Ryoo, R. Panda, and L. K. John. Watt watcher: Fine-grained power estimation for emerging workloads. In *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 106–113, 2015. doi:10.1109/SBAC-PAD.2015.26.

[110] W. Lee, Y. Kim, J. H. Ryoo, D. Sunwoo, A. Gerstlauer, and L. K. John. Powertrain: A learning-based calibration of mcpat power models. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 189–194, 2015. doi:10.1109/ISLPED.2015.7273512.

[111] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros. Splash-3: A properly synchronized benchmark suite for contemporary research. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 101–111, 2016. doi:10.1109/ISPASS.2016.7482078.

[112] A. Hyvärinen. *ICA by Minimization of Mutual Information*, chapter 10, pages 221–227. John Wiley & Sons, Ltd, 2001. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471221317.ch10, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471221317.ch10, doi:https://doi.org/10.1002/0471221317.ch10.

[113] K. Esbensen and P. Geladi. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(Issues 1-3):37–52, 1987. arXiv:arXiv:1011.1669v3, doi:10.1016/0169-7439(87)80084-9.

[114] D. A. Belsley. A Guide to using the collinearity diagnostics. *Computer Science in Economics and Management*, 4(1):33–50, 1991. doi:10.1007/BF00426854.

[115] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *Trans. Neur. Netw.*, 17(4):879–892, July 2006. URL: https://doi.org/10.1109/TNN.2006.875977, doi:10.1109/TNN.2006.875977.

[116] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi:10.1109/4235.996017.

[117] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf. Nsga-net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 419–427, New York, NY, USA, 2019. Association for Computing Machinery. URL: https://doi.org/10.1145/3321707.3321729, doi:10.1145/3321707.3321729.

[118] X. Chu, B. Zhang, and R. Xu. Multi-objective reinforced evolution in mobile neural architecture search. In A. Bartoli and A. Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 99–113, Cham, 2020. Springer International Publishing.

[119] P. Vidnerová and R. Neruda. Multi-objective evolution for deep neural network architecture search. In H. Yang, K. Pasupa, A. C.-S. Leung, J. T. Kwok, J. H. Chan, and I. King, editors, *Neural Information Processing*, pages 270–281, Cham, 2020. Springer International Publishing.

[120] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. Technical report, 1994.

[121] K. Deb and S. Agrawal. A niched-penalty approach for constraint handling in genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*, pages 235–243, Vienna, 1999. Springer Vienna.

[122] ARM Limited. Cortex-M0 technical reference manual. Technical report, 2009.

[123] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 65–74. ACM, 2017.

[124] K. S. Narendra and P. G. Gallman. An Iterative Method for the Identification of Nonlinear Systems Using a Hammerstein Model. *IEEE Trans. Autom. Control*, 1966. doi:10.1109/TAC.1966.1098387.

[125] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *IMS Trans. Ann. Statist.*, 2004.

[126] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to Forget: Continual Prediction with LSTM. 1999.

[127] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2014.