Fine-Grained Power Modeling of Multicore Processors Using FFNNs

Mark Sagi(✉), Nguyen Anh Vu Doan, Nael Fasfous, Thomas Wild, and Andreas Herkersdorf

Technical University of Munich, Munich, Germany
{mark.sagi,anhvu.doan,nael.fasfous,thomas.wild,herkersdorf}@tum.de

Abstract. To minimize power consumption while maximizing performance, today's multicore processors rely on fine-grained run-time dynamic power information – both in the time domain, e.g. μs to ms, and space domain, e.g. core-level. The state-of-the-art for deriving such power information is mainly based on predetermined power models which use linear modeling techniques to determine the core-performance/core-power relationship. However, with multicore processors becoming ever more complex, linear modeling techniques cannot capture all possible core-performance related power states anymore. Although, artificial neural networks (ANN) have been proposed for coarse-grained power modeling of servers with time resolutions in the range of seconds, no work has yet investigated fine-grained ANN-based power modeling. In this paper, we explore feed-forward neural networks (FFNNs) for core-level power modeling with estimation rates in the range of 10 kHz. To achieve a high estimation accuracy, we determine optimized neural network architectures and train FFNNs on performance counter and power data from a complex-out-of-order processor architecture. We show that, relative power estimation error decreases on average by 7.5% compared to a state-of-the-art linear power modeling approach and decreases by 5.5% compared to a multivariate polynomial regression model. Furthermore, we propose an implementation for run-time inference of the power modeling FFNN and show that the area overhead is negligible.

Keywords: Processor · Multicore · Power · Modeling · Estimation · Core-level · Artificial neural network · ANN · FFNN · Accuracy · Error · Overhead

1 Introduction

To take effective management decisions, both power and thermal (P&T) management of multicores depend on accurate run-time dynamic power consumption information at core-level. Due to the cost-prohibitive nature of actually measuring core power, such run-time power information is usually derived from predetermined power models [14] which use observable performance counters,

A. Orailoglu et al. (Eds.): SAMOS 2020, LNCS 12471, pp. 186–199, 2020.
https://doi.org/10.1007/978-3-030-60939-9_13

operating frequency and voltage as inputs. The performance counters are necessary to model the activity and thus indirectly the power consumption of each core. Apart from the spatial resolution (core-level), such dynamic power information also has to have a high time resolution (μs to ms) to be useful for P&T management of the processor [16,17]. Most dynamic power models with such spatial and time resolution commonly assume a linear relationship between performance counters and dynamic power and a nonlinear relationship between changes of the voltage/frequency state and dynamic power [2,4,6,9,18,21,22].

Previous works have shown that the relationship between performance counters and dynamic core power can also be nonlinear at core-level at least for low time-resolutions (1 s) [13]. Furthermore, for server level energy accounting with low time resolutions (0.5 Hz–1 Hz), multiple works have already proposed using ANNs for power/energy modeling to capture such nonlinear relationships [7,12,20,24]. However, increasing complexity of modern core architectures integrating hundreds of millions of transistors per core and the importance of effective P&T management increasingly necessitate fine-grained high accuracy power models capturing nonlinear performance counter/power relationships. We therefore investigate the use of FFNNs for high rate power estimations on core-level and the associated run-time inference overhead. A motivational example for using FFNNs is given in Fig. 1 showing core power on the y-axis and a time frame of 250 ms on the x-axis. PARSEC raytrace is executed and actual power is shown with the green line; estimated power based on a linear model is shown as the red line and estimated power based on an FFNN model is shown with the blue line. One can see that the FFNN model more accurately estimates the actual power consumption than the linear model thus minimizing estimation error which allows for more effective P&T management.

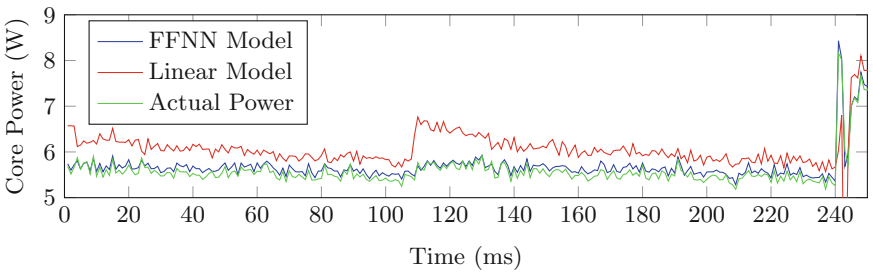

Fig. 1. Actual core power compared to power estimations using an ANN model and a linear model (Color figure online)

With this paper we make the following contributions toward fine-grained run-time power estimation of multicores:

– We explore FFNNs for power estimation on core-level with an estimation rate of 10 kHz.

- We optimize ANN hyperparameters, e.g. number of layers and neurons, with the goal of minimizing estimation error while avoiding overfitting.
- We show 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.
- We propose a micro-controller implementation for run-time inference for power estimation and discuss its negligible area overhead.

2 Related Work

A range of different methodologies for run-time dynamic power estimation on core-level or even on core-component-level for multicore processors have been proposed. Works which focus on a high estimation rates usually rely on linear models to describe the relation between performance counters and dynamic power and can be found for Intel or AMD multicore processors in [2,4,6], for IBM multicore processors in [9] and for embedded ARM multicore processors in [18,21,22].

McCullough et al. first identified non-linear power responses due to changing workloads in multicore power traces which have to be accounted for in power models [13]. They traced power consumption of an Intel Core i7, broke down power consumption on core-level and explored both linear and non-linear modeling techniques (polynomial regression and support vector regression). However, the sampling rate was comparatively low (1 Hz) and their non-linear models did not significantly improve upon linear models.

Further non-linear modeling techniques gained traction in the research area of power/energy modeling for datacenter systems and cloud servers. The first to propose ANNs for power modeling were Cupertino et al. [7]. They determined that a FFNN architecture with 2 hidden layers – 20 neurons in the first layer and 5 neurons in the second layer – to provide accuracy improvements compared to state-of-the-art linear power modeling techniques. However, the very low sampling rate 3 Hz for power and performance counter data limits the applicability to energy accounting and load balancing in datacenters.

In [10], ANNs were used to predict power consumption of applications across different processor architectures with the underlying assumption that the linear power models for each processor architecture are sufficiently accurate. In [20], the so called *additivity* of performance counters in regard to energy modeling of multicores is explored were additivity denotes the robustness of reusing a performance counter as model input for a wide range of applications. To determine the additivity of performance counters for their energy model generation, the use of linear, tree based and ANN models was investigated. However, parametrization and optimization of the neural network models is not discussed and the focus was on full system energy modeling with low estimation rates.

Another work exploring three different ANNs (FFNN, Elman and LSTM) for cloud server power modeling is given in [12] with BP and Elman having a single hidden layer with 25 neurons and the LSTM having 2 hidden layers with

10 neurons each. The resulting power model provides high accuracy at course-grained spatial (system-level) and time (1 Hz) resolution. In contrast to our work, core-level power models with high estimation rates (10 kHz) were not explored. Recurrent Elman neural networks have been proposed for course-grained power modeling of cloud servers where one hidden layer encodes power states and which exploits time series performance information [24]. Finally, for system design, the development of P&T management algorithms and power modeling algorithms, power simulators are often used, e.g. McPAT [11]. Although, the underlying principles of such power simulators are highly accurate, they cannot be used for runtime power estimation due to their large computational overhead.

In contrast to previous work, this paper focuses on generating FFNN-based power models accounting for non-linear effects with fine-grained spatial (core-level) and time (10 kHz) resolution with the resulting power estimations being applicable for run-time P&T management purposes. 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 – in regard to number of hidden layers and neurons – and the overhead for run-time inference necessary.

3 Feed-Forward Neural Networks for Power Modeling

Power modeling for runtime power estimation is inherently a regression problem with the desired power information as dependent variable and the performance counters as independent variables. We focus on dynamic core-level power information $P_{core,j}$ where j denotes the j-th core of the multicore processor. The core power is estimated during run-time through n performance counters PC_i with $0 \leq i \leq n$ all related to their respective cores. With actual $P_{core,j}$ not being observable for each individual core, we use the following approximation for the model generation step when only the j-th core is active at a time:

$$P_{core,j} = P_{pack} - P_{idle}. \tag{1}$$

Package-level power P_{pack} can be observed through instrumentation of the mainboard, i.e. actual power sensors, and P_{idle} is the idle power of the processor when no core is active. With only P_{pack} being actually observable, we generate different models (FFNN, polynomial, linear) for P_{pack} and subtract P_{idle} to derive core-level power consumption. Therefore, the PC_i and P_{pack} data used for model generation is reduced to timeframes where only a single core is active at a time to capture the power response of that particular core. As we only investigate a homogeneous multicore processor in this work, the power models for the j-th core can be generalized to any core of the system by using the performance counters of those cores as model input, respectively. The error (cost) function for generating the subsequent power models is then:

$$P_{pack,error} = |P_{pack,act} - P_{pack,est}| \tag{2}$$

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

3.1 ANN Architecture

There exist a multitude of ANN architectures, e.g. FFNN, Elman, LSTM, for modeling and prediction of non-linear functions and systems. With most fine-grained power models using linear regression models, we keep our analysis to comparatively simple ANN architectures. Our goal is to achieve higher estimation accuracies than with linear modeling techniques while adding as little additional modeling complexity and run-time overhead as possible. For this reason, we choose well-known feedforward networks which can theoretically model any nonlinear function according to the universal approximation theorem [8]. Similar to previous works, we do not use any input delay on the PC_i inputs, i.e. we do not generate any autoregressive models. 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 relationship. With a finite amount of training data, FFNNs of sufficient size 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 PC_i data will be significant. Therefore, careful consideration has to be taken of the chosen hyperparameters of the FFNN which are distinguished between *algorithm* hyperparameters (learning related) and *model* hyperparameters (architecture related). For the algorithm hyperparameters, we train a number of networks for dynamic power estimation and compare both the resulting accuracy as well as training time and find that conjugate gradient backpropagation with Polak-Ribiére updates provide the best training speed/accuracy trade-off. As stop conditions for training the FFNN, we use the following:

- 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,
- 5 subsequent failed validation tests where additional training leads to higher estimation errors on the validation data.

The question of how to determine the optimal FFNN model hyperparameters is still an ongoing topic of research, therefore, we follow best practices for the hyperparametrization. As a first step, the model hyperparameters have to be confined. For the activation function of the hidden neurons, we choose tanh after sweeping over a set of different activation functions and comparing estimation accuracy.

For the number of hidden layers and hidden neurons per layer, we align ourselves with the related work for coarse-grained power models for servers/datacenters and confine the hidden layer and hidden neuron hyperparameters as shown in Fig. 2. We explore one two-layered *shallow* network with 1–30 neurons per hidden layer. Note, that we explore all possible combinations of the number of hidden neurons per layer, i.e. 900 differently parameterized two-layered FFNNs. We further investigate one *mid-sized* network with 3 hidden layers where the number of neurons per layer can be any number of $1, 4, 7, 10, 13, 16, 19, 22, 25, 28$ and one *deep* network with 5 hidden layers where

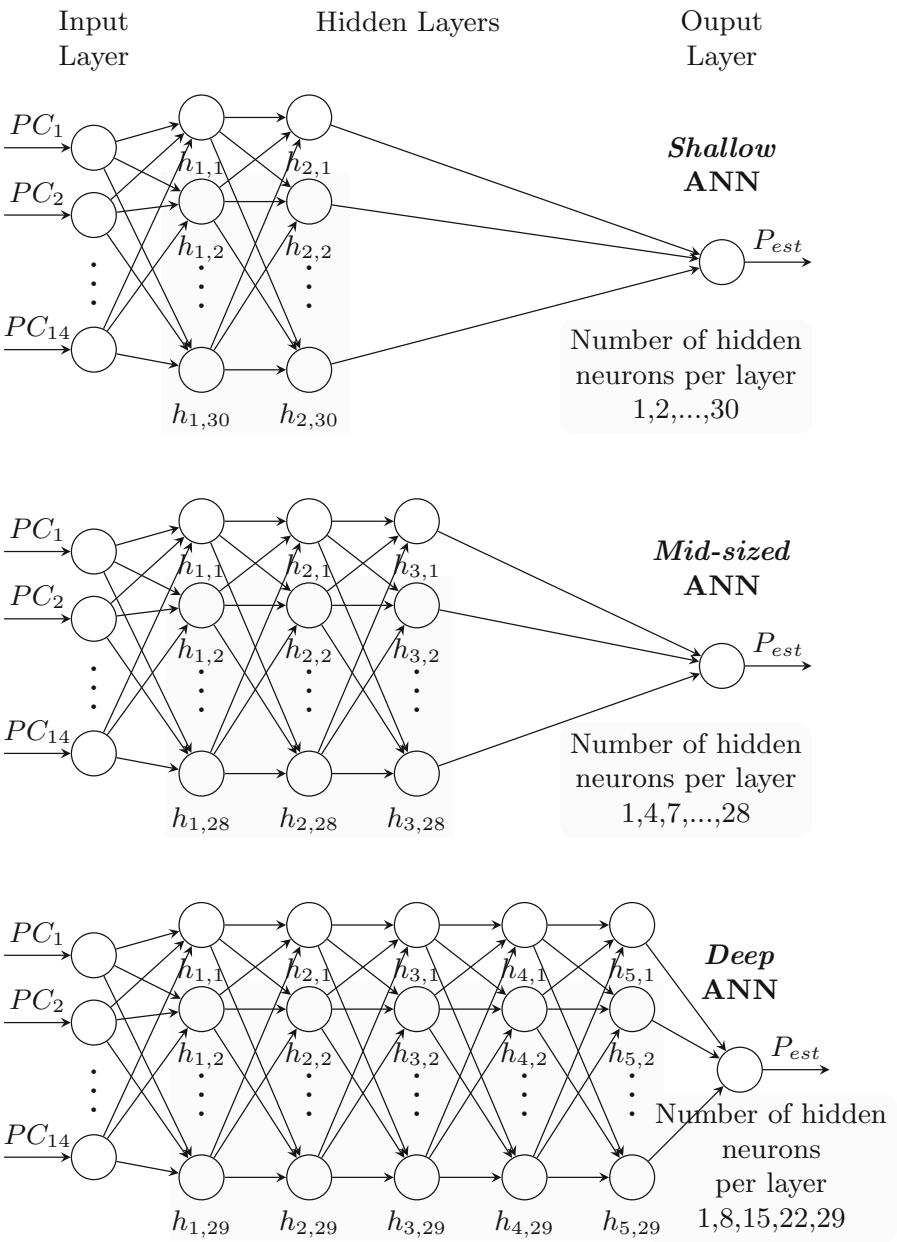

Fig. 2. Overview of the ANN architectures investigated; the yellow shaded rectangles indicate the ability to parameterize the number of hidden neurons per layer, i.e. from at least one hidden neuron per layer up to the given maximum number

the number of neurons per layer can be $1, 8, 15, 22, 29$. The number of neurons per layer is constrained for the mid-sized and deep networks to keep the amount of training time on a reasonable level.

Although, adding layers and number of neurons per layer increases the risk of overfitting, it also decreases the risk of underfitting due to an undersized FFNN. In the following, we show our methodology to find hidden neuron parametrizations with the most consistent estimation performance for each of the three different network sizes.

3.2 Hyperparameter Optimization and FFNN Training

Our methodology for the hyperparametrization of the model parameters and for the final training and computation of expected estimation accuracy are shown in Fig. 3. We use 10-fold cross validations for both, model hyperparametrization and to train a final FFNN of each size (shallow, mid-sized and deep), i.e. for actual deployment.

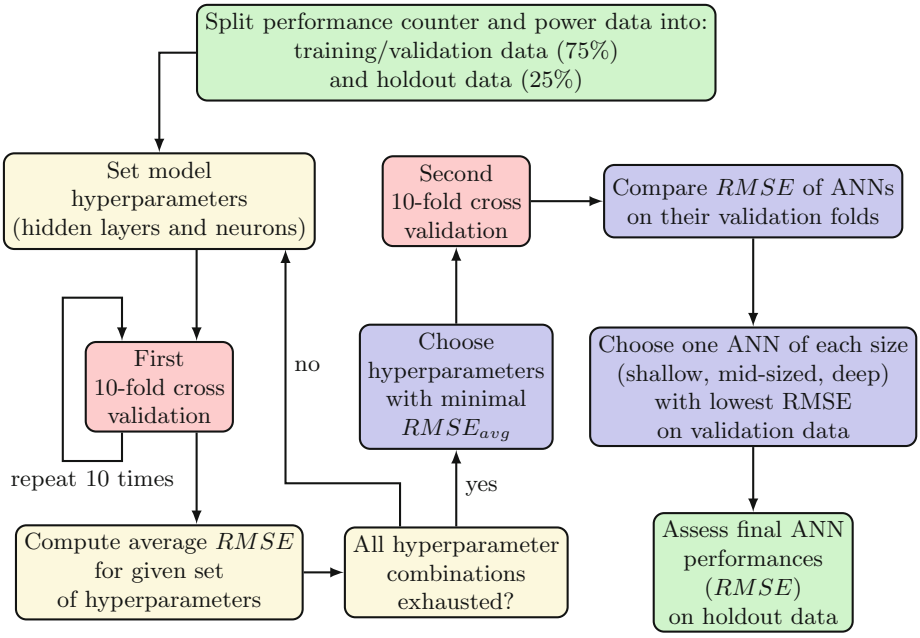

Fig. 3. Flowchart of model hyperparametrization using first 10-fold cross validation, final ANN generation using second 10-fold cross validation and final ANN estimation accuracy assessment

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 is as

diverse in regard to power behavior as possible. The holdout data set is neither used for hyperparametrization nor to train the three final FFNNs and can thus be used to determine the actual performance of these FFNNs on data they have not yet seen. For the hyperparametrization, we loop over all possible hidden neuron per layer combinations for each FFNN size (*shallow*, *mid-sized*, *deep*) and execute the first cross validation loop. 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 for training the ANN. We repeat this step for each fold ten times to produce statistically significant results and to be able to remove outliers, i.e. diverging FFNNs. Thus, 100 FFNNs 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 over all folds. We then choose the hidden neuron parametrizations for each FFNN size with lowest average $RMSE$ under the assumption that these 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 where we generate an FFNN for each training/validation fold combination, i.e. 10 FFNNs for each FFNN size (*shallow*, *mid-sized* and *deep*) and choose those FFNNs for testing which performed best on their corresponding validation data. We use the second cross validation to minimize the risk of selecting an overfitting FFNN from the first cross validation where 100 different FFNNs were generated for each hyperparameter. The risk of the FFNN with highest accuracy on their respective validation fold being overfitting is higher when 10 such FFNNs are available to chose from rather than just one. In the final step, we test the three chosen FFNNs on the holdout data to assess their potential dynamic power estimation performance in an actual deployment environment.

4 Experimental Setup

We use HotSniper [15], which is based on the Sniper multicore simulator [5] and expands the Sniper simulator with periodic power simulations using McPAT [11]. Sniper simulator uses interval simulation to speedup simulation times while providing good simulation accuracy. It is widely used for research of processor power/thermal management and power modeling, e.g. in [17,19]. Compared to a cycle-level simulator, an interval simulator focuses on accurately simulating performance changes due to stalls of the execution flow, while modeling performance on a higher abstraction layer when the execution flow is continuous. Our experimental framework simulates a 16-Core processor with Intel Gainestown core microarchitecture, with the details given in Table 1. We trace common performance counters similar to other state-of-the-art work, e.g. [2], with a sampling rate of 10 kHz. Some works use a larger number of performance counters – in

Table 1. Simulation framework

16-Cores (2 × 2 Tiles) with NoC Interconnect		
Intel Gainestown Core Architecture		
Memory Architecture		
L1 Caches	32 KB (private)	
L2 Caches	256 KB (private)	
L3 Caches	8 MB (shared per tile)	
Performance Information		
Processor Unit	Performance Counter	
Core	Instruction per Cycle	IPC
	# Branch Instructions	BPU
	# Floating Point Instructions	FP
	% C0 State Residency of a Core	C0
L2 & L3 Cache	# Load Instructions	LxLI
	# Store Instructions	LxSI
	# Load Misses	LxLM
	# Store Misses	LxSM
	% Cycles Lost due to misses	LxCLK
Workloads for Model Generation and Evaluation		
Suite	Benchmarks	
PARSEC [3]	blackscholes, bodytrack, canneal, dedup, streamcluster, swaptions, x264	
Splash-2 [23]	barnes, cholesky, fft, fmm, lu, ocean, radiosity, radix, raytrace	

the range of 50–100 – to estimate power due to the possibility of the estimation error decreasing with increasing number of performance counter inputs. However, processors commonly only support recording a small number of performance counters – in the range of 5–15 – at the same time [4]. We use the power and energy information provided by McPAT in our evaluation which generates power information on package, core and core component granularity with microsecond resolution. Finally, the benchmarks used for training the ANNs and generating reference power models are also given in Table 1.

5 Evaluation

5.1 Reference Power Models

We compare the estimation accuracy of the final FFNNs with the state-of-the-art linear approach published in [2] and a polynomial regression model as proposed by McCullough et al. [13]. For the linear model, we execute a set of microbenchmarks and the PARSEC/Splash-2 benchmarks on the system, trace P_{pack} and PC_i and generate a core-level linear regression model. Although [13] argues that a polynomial regression model was not able to accurately capture the non-linear power relationships – possibly due to overfitting – we still use it to test

the hypotheses that our performance/power data could potentially be described well through polynomial regression and thus obviating the need for the complex ANN methodology. To generate the polynomial regression model, we use a similar methodology as described in Sect. 3 for FFNNs. First, we do repeated 10-fold cross validations to determine the best polynomial order and 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. We explored maximum polynomial orders of 1–6 for the independent PC_i inputs and found that a maximum polynomial order of 2 offered the best average estimation performance on the validation data.

5.2 Hidden Neuron Parametrization

We first look at the best model hyperparameters of the three hidden layer sizes and the average root mean squared error (RMSE) on the validation data from the 10-fold cross validation. For the three different FFNN sizes, the hidden neuron parametrization with lowest average RMSE is shown in Table 2. In addition the relative error to average core power is computed as $\frac{RMSE}{P_{avg}}$.

Table 2. Hyperparametrisation of shallow, mid and deep FFNNs with average RMSE on the randomized validation data

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%

Shallow FFNNs with 2 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 and deep FFNN offer worse performance than the shallow FFNN on the validation folds. Additionally, we also show the relative estimation error of the actual core power.

5.3 Estimation Accuracy

We use the hyperparameters from Table 2 to generate three final FFNNs (shallow, mid, deep) using the second round of 10-fold cross validation. Note that for actual deployment in a multicore processor, we would only generate the *shallow* FFNN as it performed best on the validation data in the 10-fold cross validation. However, for providing an extensive performance overview and analysis we also show the performance of the *mid-sized* and *deep* FFNN. The estimation accuracy of the three resulting FFNNs is then determined on the holdout data, i.e. data

which has neither been used for the hyperparametrization nor for training the FFNNs. Table 3 shows the RMSE and percentage errors of the three FFNNs, the model linear and the multivariate polynomial model as comparison.

Table 3. Estimation accuracy of FFNNs, linear model and polynomial model on the holdout data set

Model	RMSE	Relative error	MAPE
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%
Linear Model [2]	0.75 W	12%	12%
Polynomial Model	0.60 W	10%	11%

From the FFNNs, the *shallow* one has the best estimation performance with the *mid-sized* and *deep* FFNN having worse error as was to be expected from the validation data. This shows – at least for FFNNs – that shallow structures with 2 layers are sufficiently complex to adequately capture the nonlinear performance/power relationships while FFNNs with more than 2 hidden layers start to suffer from overfitting. Compared to the state-of-the-art linear model, we observe a decrease in relative error of 7.5%. Such an estimation improvement can be significant for both short-term power (density) management and long-term thermal 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 manager.

The best polynomial model had a 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 model. Compared to the polynomial model, the relative estimation error of the FFNN power estimator still decreases by 5.5%. The results of a *shallow* FFNN performing best as well as the best-suited polynomial order being 2, we interpret as the underlying non-linear performance counter/power relationship in itself being probably not overly complex. The comparatively bad polynomial model performance reconfirms the findings of McCullough et al. [13] that polynomial regression modeling very easily overfits the underlying performance/power data and is not well-suited to capture the non-linear relationships. In addition to the RMSE values and relative error values compared to average core power we also provide the mean absolute percentage error (MAPE) values in the final comparison shown in Table 3. Compared to the relative error, MAPE penalizes underestimations of the core power consumption stronger than overestimations of core power which could be advantageous for conservative power management algorithms. However, we do not observe significant qualitative differences between the relative error values and the MAPE values and have included MAPE for sake of completeness.

5.4 Run-Time Inference Overhead

To assess the computational and memory overhead of run-time inference of the FFNNs for producing a single power estimation, we first compute the necessary number of multiply accumulate (MAC) operations and the memory needed to store the neuron weights. The number of MAC operations is computed from the interconnections of each FFNN and the memory needed from the number of stored 32-bit weights values with both given in Table 4. Compared to the linear regression model, the *shallow* FFNN needs approximately between 60 times more MAC operations and 40 times more memory. Both, the computational and memory overhead are magnitudes higher for any FFNN implementation and we therefore discuss the feasibility and area overhead of a run-time inference implementation of the *shallow* FFNN in the following.

Table 4. Necessary computations and memory for a single power estimation/model inference

Model	Number of MAC operations	Memory in kBit
Shallow ANN	827	20
Mid-sized ANN	1971	56
Deep ANN	1426	39
Linear model [2]	14	0.5
Polynomial model	28	1.0

At least on IBM multicore processors, micro-controllers are integrated for both power estimation – so-called power proxies – and for power management purposes [9]. In the following, we approximate the transistor overhead for power estimations using the shallow ANN assuming integrated micro-controllers for power estimation. As a reference micro-controller, the 32-bit ARM Cortex M0 is well established, can be conservatively operated at 50 MHz with an implementation using less than 100k transistors [1]. In addition, the required SRAM to store the weights for the *shallow* ANN would add an additional 120k transistors. Each MAC operation and its associated load/store instructions takes 6 cycles on the M0 leading to approximately 5k cycles for a single inference of the shallow ANN, thus, power estimations could be executed with a periodicity of 100 μs.

Depending on the requirements of the P&T management, one micro-controller would be needed per core if 10 kHz power estimations are needed. Such a micro-controller implementation leads to an overhead of 250k transistors under conservative assumptions and has to be compared to the 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 estimation error by 7.5%, translating to better P&T 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 programmability of the power estimator through firmware changes. Also, lower estimation rates in the range of 1 kHz would allow one power estimating micro-controller to serve multiple cores thus also decreasing area overhead.

6 Conclusion

In this paper, we investigate the use of FFNNs for fine-grained run-time power estimation on core-level with an estimation rate of 10 kHz. Suitable parametrizations of the number of hidden neurons per layer for three FFNN sizes (2, 3 and 5 hidden layers) were explored. To avoid underfitting non-linear relations between performance counters and dynamic power, and to avoid overfitting the training data, we used 10-fold cross validations for determining well-suited hidden neuron hyperparameters, and for the generation of the final three FFNNs for deployment. The *shallow* FFNN with 2 hidden layers proved to be 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. Thus, shallow FFNNs are sufficient to capture non-linear relations between performance and power, thus providing significant improvements in accuracy for fine-grained, high-rate power estimations.

Furthermore, we propose a micro-controller-based implementation 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. The higher run-time power estimation accuracy can be exploited by power and thermo management algorithms for more effective management decisions, e.g. power-density aware task mappings and frequency boosting.

Acknowledgments. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Projektnummer 146371743 - TRR 89 “Invasive Computing”.

References

1. ARM Limited: Cortex-M0 technical reference manual. Technical Report (2009)
2. Bertran, R., Gonzelez, M., Martorell, X., Navarro, N., Ayguade, E.: A systematic methodology to generate decomposable and responsive power models for CMPs. IEEE Trans. Comput. **62**(7), 1289–1302 (2013)
3. Bienia, C.: Benchmarking Modern Multiprocessors (2011)
4. Bircher, W.L., John, L.K.: Complete system power estimation using processor performance events. IEEE Trans. Comput. **61**(4), 563–577 (2012)
5. Carlson, T.E., Heirman, W., Eyerman, S., Hur, I., Eeckhout, L.: An evaluation of high-level mechanistic core models. ACM TACO **11**(3), 1–25 (2014)
6. Chadha, M., Ilsche, T., Bielert, M., Nagel, W.E.: A statistical approach to power estimation for x86 processors. In: Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017 (2017)

7. Cupertino, L.F., Da Costa, G., Pierson, J.-M.: Towards a generic power estimator. Comput. Sci. Res. Dev. **30**(2), 145–153 (2014). https://doi.org/10.1007/s00450-014-0264-x
8. Huang, G.B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Networks **17**(4), 879–892 (2006)
9. Huang, W., et al.: Accurate fine-grained processor power proxies. In: IEEE/ACM MICRO (2012)
10. Kim, Y., Mercati, P., More, A., Shriver, E., Rosing, T.: P4: phase-based power/performance prediction of heterogeneous systems via neural networks. In: IEEE/ACM ICCAD (2017)
11. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: IEEE MICRO (2009)
12. Lin, W., Wu, G., Wang, X., Li, K.: An artificial neural network approach to power consumption model construction for servers in cloud data centers. IEEE Trans. Sustain. Comput. **5**(3), 329–340 (2019)
13. McCullough, J.C., et al.: Evaluating the effectiveness of model-based power characterization. In: Usenix Atc (2011)
14. Möbius, C., Dargie, W., Schill, A.: Power consumption estimation models for processors, virtual machines, and servers. IEEE TPDS **25**(6), 1600–1614 (2014)
15. Pathania, A., Henkel, J.: HotSniper: sniper-based toolchain for many-core thermal simulations in open systems. IEEE Embed. Syst. Lett. **11**(2), 54–57 (2019)
16. Rapp, M., Pathania, A., Mitra, T., Henkel, J.: Prediction-Based Task Migration on S-NUCA Many-Cores. In: DATE (2019)
17. Rapp, M., Sagi, M., Pathania, A., Herkersdorf, A., Henkel, J.: Power-and cache-aware task mapping with dynamic power budgeting for many-cores. IEEE Trans. Comput. **69**(1), 1–13 (2019)
18. Rethinagiri, S.K., Palomar, O., Ben Atitallah, R., Niar, S., Unsal, O., Kestelman, A.C.: System-level power estimation tool for embedded processor based platforms. In: ACM RAPIDO (2014)
19. Samei, Y., Dömer, R.: Automated estimation of power consumption for rapid system level design. In: IEEE IPCCC (2014)
20. Shahid, A., Fahad, M., Manumachu, R.R., Lastovetsky, A.: Improving the accuracy of energy predictive models for multicore CPUs using additivity of performance monitoring counters. In: Malyshkin, V. (ed.) PaCT 2019. LNCS, vol. 11657, pp. 51–66. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25636-4_5
21. Su, B., Gu, J., Shen, L., Huang, W., Greathouse, J.L., Wang, Z.: PPEP: online performance, power, and energy prediction framework and DVFS space exploration. In: IEEE/ACM MICRO (2014)
22. Walker, M.J., et al.: Accurate and stable run-time power modeling for mobile and embedded CPUs. In: IEEE TCAD (2017)
23. Woof, S.C., Ohara, M., Torriet, E.: The SPLASH-2 programs: characterization and methodological considerations. In: ACM ISCA (1995)
24. Wu, W., Lin, W., He, L., Wu, G., Hsu, C.H.: A Power Consumption Model for Cloud Servers Based on Elman Neural Network. IEEE Transactions on Cloud Computing (2019)