DOI: 10.14529/jsfi180403

A Study on Cross-Architectural Modelling of Power Consumption Using Neural Networks

Miloš Puzović[1]***, Eun Kyung Lee***[2]***, Vadim V. Elisseev***[3]

On the path to Exascale, the goal of High Performance Computing (HPC) to achieve maximum performance becomes the goal of achieving maximum performance under strict power constraint. Novel approaches to hardware and software co-design of modern HPC systems have to be developed to address such challenges.

In this paper, we study prediction of power consumption of HPC systems using metrics obtained from hardware performance counters. We argue that this methodology is portable across different micro-architecture implementations and compare results obtained on Intel® 64, IBM® POWER™ and Cavium ThunderX® ARMv8 microarchitectures. We discuss optimal number and type of hardware performance counters required to accurately predict power consumption.

We compare accuracy of power predictions provided by models based on Linear Regression (LR) and Neural Networks (NN). We find that the NN-based model provides better accuracy of predictions than the LR model. We also find, that presently it is not yet possible to predict power consumption on a given microarchitecture using data obtained on a different microarchitecture. Results of our work can be used as a starting point for developing unified, cross-architectural models for predicting power consumption.

Keywords: hpc, power consumption, modelling, exascale, cross-architectural, neural networks.

Introduction

Upcoming High Performance Computing (HPC) systems are on the critical path towards delivering the highest level of performance for large scale applications. If contemporary technologies were used to build even more powerful HPC systems, the power consumption required by those systems would be unsustainable, as it would require hundreds of megawatts of power. Thus, current HPC systems must be built considering *energy efficiency* as the first and foremost design goal. Currently, the most power efficient HPC system on the Green500 [26] list is Shoubu System B located at ACCC, RIKEN with 17GFlops/Watt. In order to achieve a sustainable power draw, future HPC systems will have to feature power efficiency of around 50GFlops/Watt. Such power efficiency levels require novel software/hardware co–design, with software guiding static and dynamic power management.

Successful development of such software relies on availability of tools for measuring power consumption and temperature. Temperature and power sensors have been introduced to provide these type of measurements. Unfortunately, as a result of the fabrication process used to build microprocessors, measurements of changes in power and temperature happen significantly later then the actual event (*thermal inertia*). An alternative to using power and temperature sensors would be to directly measure processor events that are causing power and temperature changes thus eliminating time lag limitation. The best proxy for measuring processor events is the *hardware performance counters*, because they offer a reliable interface to detect power and temperature variations within a real system. Considering a very large number of hardware performance counters typically available on modern systems, the task of selecting counters that are most representative of the full system power is becoming a challenge. Moreover, complexity of

[1]The Hartree Centre, STFC Daresbury Laboratory, Sci-Tech Daresbury, Cheshire, UK
[2]IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
[3]IBM Research, STFC Daresbury Laboratory, Sci-Tech Daresbury, Cheshire, UK

this problem significantly increases when different types of micro–architecture implementations are considered. The matter is further complicated by the fact that each architecture has a limited number of registers that can be simultaneously recorded without resorting to multiplexing and therefore reducing the accuracy. For example, IBM POWER8 architecture can simultaneously track at most six different performance counters, while the latest Intel 64 architecture can track up to eight.

In this paper, we extend recent works [2, 13] on estimation of power consumption of HPC systems with metrics obtained using hardware performance counters on three different micro–architecture implementations: Intel 64 Broadwell, IBM POWER8 and Cavium ThunderX ARMv8 architecture and argue that the methodology is portable across different micro–architecture implementations. We study advantages and disadvantages of a model presented in [2] and [13] in dealing with emerging HPC workloads. We improve accuracy of power consumption predictions by adoptong a model based on Neural Networks (NN) and discuss the optimal number and the type of hardware performance counters required to accurately predict power consumption within few percents of the actual power consumption. We believe that our results can guide system designers in implementing hardware counters which measure similar events between different types of architectures, which in turn will allow developing more general models of power consumption.

The rest of the paper is structured as follows. Section 1 describes the use of hardware performance counters for power consumption estimations. Section 2 explains advantages and disadvantages of the power estimation model from [2, 13] and describes methodology that was used to collect data on three different HPC systems. Section 3 describes NN based approach to developing a more accurate prediction model and shows results that we have obtained using the new model to estimate power on emerging HPC workloads. Finally, the section entitled "Conclusions and Future Work" presents conclusions and future directions of our work.

1. Related Work

The use of hardware performance counters to estimate power consumption has been around for some time [10]. Such approach is appealing, because it does not require information about power consumption of individual functional units, but its accuracy relies on selecting the most suitable hardware performance counters and on having a *very* representative set of applications, which can fully characterise the target platform in order to build a reliable expression for power estimation. Previous studies [18] have shown that it is sufficient to use only instructions per cycle (IPC) to characterise behaviour of an operating system. In this case, the constructed power consumption model is a *static* power model. In [4], authors have proposed a *dynamic* power model, where the framework used to construct it contains a set of heterogeneous power models. In all these examples, the power model has been constructed only for a micro–processor and memories, but not for events related to a chipset, I/O and disk. For majority of workloads, that is acceptable in terms of accuracy as microprocessors and memories consume most of the power, and the rest of the system makes up between 10% and 20% of total power (dependant on the type of the workload). The work presented in [7] shows how to assess the total power of a system using hardware performance counters. The authors use the *trickle–down* approach where values of *power inducing* hardware performance counters are propagated within different subsystems in order to simulate the total power. The model they have built was accurate within 9%.

The work presented in [23] considers two processor configurations at the opposite ends of a performance spectrum: a low power processor and a high performance processor. The authors have found that there exists a subset of hardware performance counters, which allows evaluating dynamic power consumption for each architecture with an average error of 5%.

The study presented in [2] demonstrates that it is possible to estimate power consumption on HPC architectures for workloads with very high CPU utilisation (all cores are utilised at more than 90%) with an average error within 5%.

Machine learning–based modelling has been gaining popularity for optimising power and energy consumption. It has been used for power and energy modelling on HPC kernels with different code variants [27] and for predicting a user's demands from the usage history for managing idle servers [12]. Genetic algorithm has been used to predict power using a wide range of hardware activity counters [17]. Neural Network has been used to minimise the cooling energy consumption [1] of a server. Borghesi et al. [8] have proposed a machine learning approach, which relies on resource requests from a user and an application to estimate power consumption of HPC workloads. Their model was able to handle cases when CPUs were not fully utilised.

However, all previous models were dealing with a particular micro architecture. In this paper, we present in–depth study on power consumption evaluation using hardware performance counters within three different micro–architecture implementations: Intel `64` Broadwell, IBM POWER8 and Cavium ThunderX ARMv8.

2. Motivation

2.1. Linear Regression Model

As the starting point for predicting power consumption, we used the model introduced in [2]. The model's equation for power consumption of a given application as a function of CPU frequency f is the following:

$$\mathbf{P}(f) = A(f) \times \ \mathbf{P}(f_0) + B(f) \times \ \mathbf{TPI}(f_0) + C(f). \tag{1}$$

In equation (1), power consumption $\mathbf{P}$ of an application running with CPU frequency f is estimated by measurements performed at reference frequency f_0. In order to use equation (1), we need to measure, power and transactions per instruction ($\mathbf{TPI}$) for each application at frequency f_0. Transactions per instruction are defined as a ratio of the number of cache lines written to and read from memory ($\mathbf{C}$) to a number of instructions executed ($\mathbf{I}$) :

$$\mathbf{TPI}(f) = \frac{\mathbf{C}(f)}{\mathbf{I}(f)}.$$

Coefficients $A(f)$, $B(f)$ and $C(f)$ are system-specific. They are used for characterising power on a given HPC system at CPU frequency f. It is important to note that for every new microarchitecture implementation, it is always necessary to obtain new $A(f)$, $B(f)$ and $C(f)$ coefficients as they are not *transferable*.

Equation (1) assumes that we have a way to measure power consumed by an application at a specific frequency. This is not always possible as described in the Section . Furthermore, the majority of power sensors expose *power lag* and *distortion* where power consumption lags behind the actual benchmark activity and the shape of power consumption does not actually match the benchmark activity [9]. Therefore, we need to modify equation (1) to use measurements obtained

from hardware performance counters. The new equation is:

$$\mathbf{P}(f) = A(f) \times \ \mathbf{GIPS}(f_0) + B(f) \times \ \mathbf{GBS}(f_0) + C(f). \tag{2}$$

In equation (2), **GIPS** is *the instruction rate*, which is defined as a number of giga instructions executed per second, and **GBS** is *the memory bandwidth*, which is defined as a number of gigabytes written to and read from the memory per second. With such metrics, coefficients A, B and C at frequency f have the following meaning:

- A is the power consumed by all executed instructions.
- B is the power consumed by data transfers.
- C is the power consumed statically.

Table 1. Intel Xeon E5 v4, IBM Power System S822LC and Cavium ThunderX characteristics

Architecture		POWER8	Intel `x86-64`	ARMv8 64bit
Processor		IBM Power System S822LC	Intel Xeon E5-2698 v4	Cavium ThunderX
Core	Frequency	3.5 GHz	2.3 GHz	2.0 GHz
	# of cores	10	16	48
	# of threads	80	32	48
Execution unit	Type	out-of-order	out-of-order	in-order
	# of issue/commit	10 / 8	8 / 4	2 / 2
L1D Cache	Policy	NUCA	Write-allocate	Write-through
	Type	Private	Private	Private
	Size	64 KB/core	32 KB	32KB
	Associativity	8-way	8-way	32-way
L1I Cache	Size	32KB/core	32KB	78 KB
	Associativity	8-way	8-way	39-way
L2 Cache	Policy	NUCA	Write-back	Write-back
	Type	Private	Private	Shared
	Size	512KB/core	256KB	16MB
	Associativity	8-way	8-way	16-way
L3 Cache	Policy	NUCA	Write-back	N/A
	Size	8MB/core	40MB	N/A
	Type	Shared	Shared	N/A
SMP Interconnect	Bus Type	Integrated SMP interconnect	QPI	CCPI
	Bus speed	9.6 GB/s per channel	9.6 GB/s	10.3 GHz
Memory	Type	DDR4 1600	DDR4 2133	DDR4 2133
	# of channels	8	4	4
	Access speed	1600 MHz	2133 MHz	2100 MHz

2.2. Methodology

In order to estimate power as given in equation (2), we need to read hardware performance counters for the total number of instructions that have been executed (*or retired*) and the total number of bytes that have been read and written to a memory controller.

Tab. 1 shows relevant characteristics of three different microarchitecture implementations that we are using throughout the paper. Tab. 2 shows hardware performance counters that were used to measure events that are required to calculate **GIPS** and **GBS** for equation (2) for the model described in Section 2.1. We used architecture-independent `libpfm` library [14] to read performance counters for events listed in Tab. 2. This approach allowed us to run our profiled benchmarks unmodified on both architectures.

To find coefficients A, B and C, we ran a set of compute kernels as in [2] that provide a broad spectrum of **GIPS** and **GBS** characteristics. We used NAS Parallel Benchmarks (NPB) [3] and STREAM [19], which have been designed to test the performance of HPC systems. NPB suite is a mix of workloads that has been derived from computational fluid dynamics, unstructured

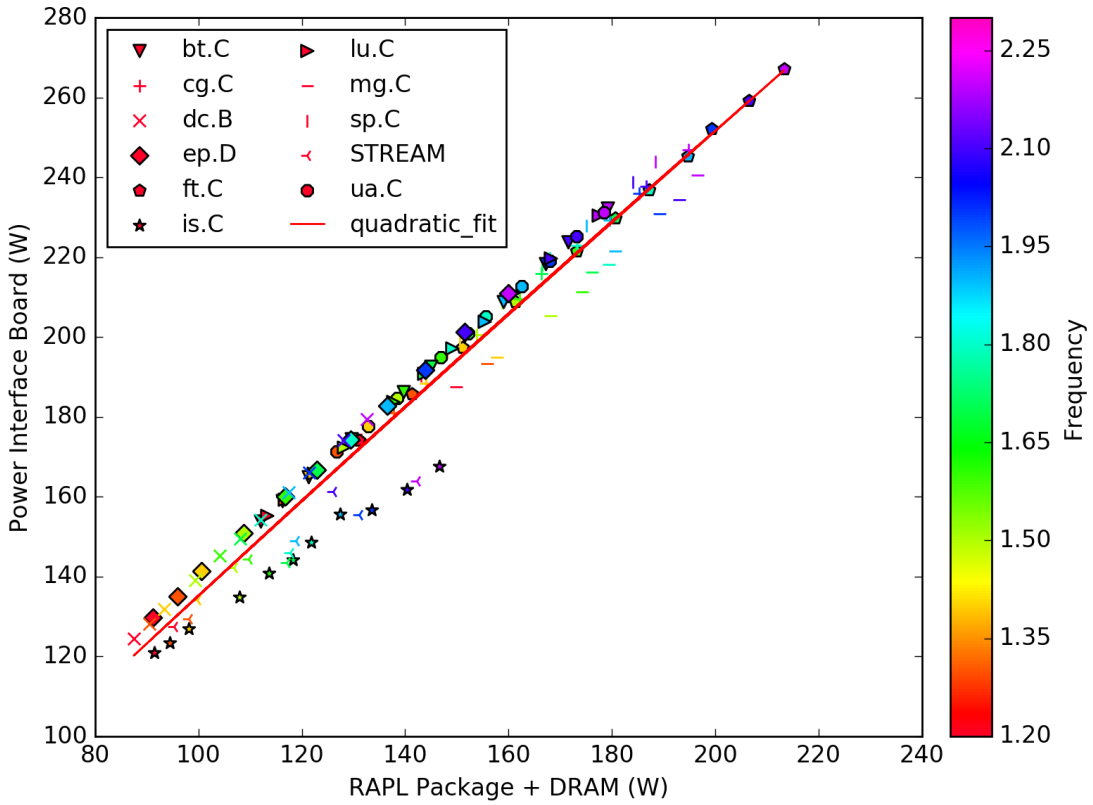


Figure 1. Correlation between the total node power and the power consumed by cores and memories

adaptive mesh, parallel I/O, multi-zone applications and computational grids, whilst STREAM is a simple synthetic benchmark that is used to measure sustainable memory bandwidth and corresponding computation rate for straightforward compute kernels. We used the OpenMPI [24] version of benchmarks with different classes/problem sizes in order to cover a wider range of workloads. As for the NPB suite convention, benchmark class is appended as a suffix to the benchmark's name; for example, `ft.C` stands for *discrete 3D fast Fourier Transform, class C.*.

Firstly, we studied which part of an HPC node contributes the most to power consumption. For brevity, results presented in this section were obtained on Intel Xeon E5 v4 microarchi-

Table 2. Hardware performance counters used on Intel Xeon E5 v4, IBM POWER8 and Cavium ThunderX microarchitecture implementations to calculate **GIPS** and **GBS** provided in equation (2)

Microarchitecture	Memory	Instructions
Intel Xeon E5 v4	UNC_CBO_CCACHE_LOOKUP.I UNC_CBO_CCACHE_LOOKUP.ANY_REQ UNC_ARB_TRK_REQUEST.EVICTIONS	EVEN_INSTR_RETIRED
IBM POWER8	PM_MEM_READ PM_MEM_PREF MEM_RWITM	PM_RUN_INST_CMPL
Cavium ThunderX ARM	L2D_CACHE_REFILL_LD L2D_CACHE_REFILL_ST L2D_CACHE_WB_VICTIM L2D_CACHE_WB_CLEAN	CPU_CYCLES

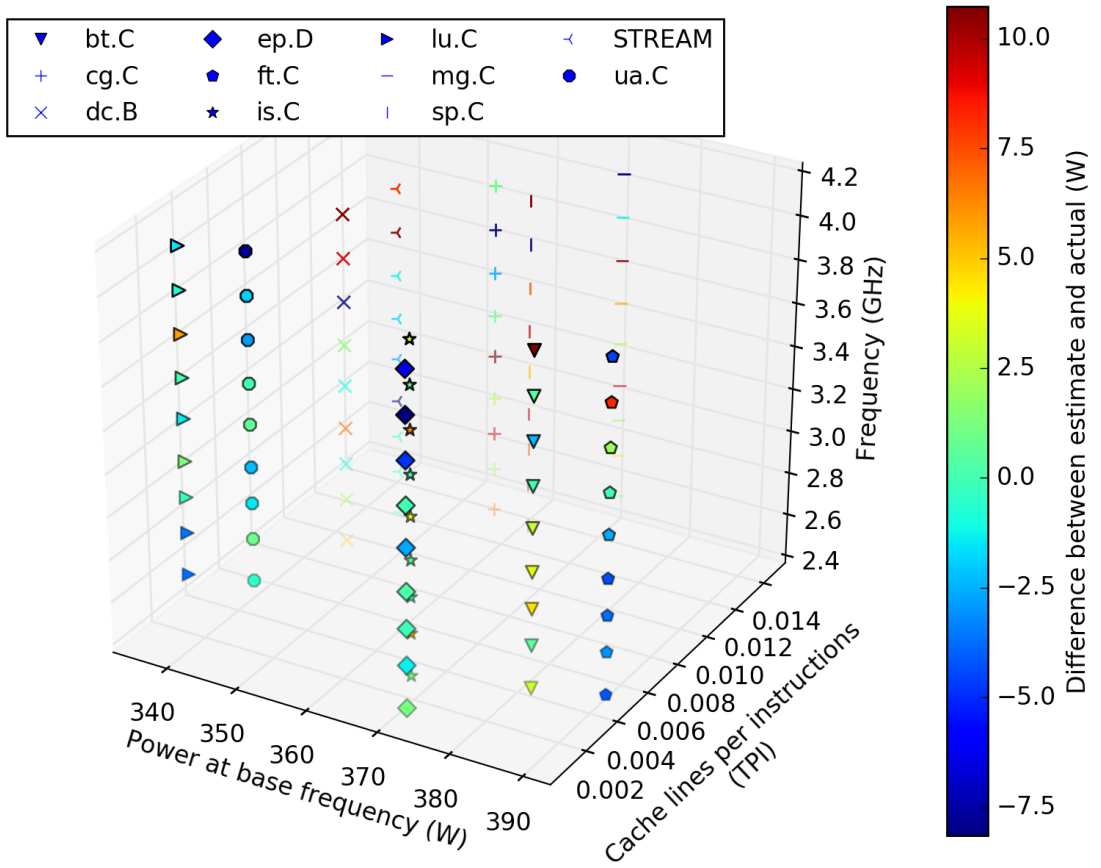


Figure 2. Accuracy of power consumption estimation of NPB and STREAM benchmarks using equation (2) on IBM POWER8

tecture, but we has the same observations on IBM POWER8 micro-architecture. In order to measure power consumption on IBM POWER8 (S822LC model) server, we used an On-Chip Controller (OCC) that collects temperature and power data from various sensors. This data is available either from a Baseboard Management Controller(BMC) via IPMI protocol or from the system memory via kernel module [5]. We used a power interface board (PIB) to get total power consumption on an Intel-based server. The PIB has a *hot-swap controller* (HSC), TI LM25062 [25] that measures and reports power. It works by having a shunt resistor and an I-sense/V-sense circuit. Thus, it is possible to position HSC with the sense circuit between the 12V DC power connector and the on-board voltage regulators to monitor the total power consumed by the attached HPC node. In addition to total power of the HPC node, we also measured power consumed only by processor and the memory. This was measured by reading information from model-specific registers (MSR) that are used for the Running Average Power Limit (RAPL) future. If there is high correlation between these two measurements for NPB benchmarks, then we can deduce that it is sufficient to only model cores and memory performance events in order to estimate total power as it can be calculated from cores and memory power using the fitted line. Fig. 1 shows the results we have obtained.

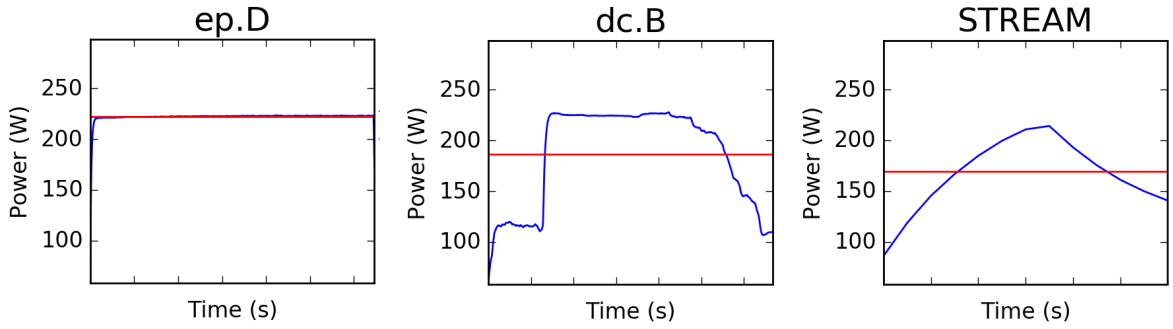


Figure 3. Actual instantaneous power (blue) and mean power (red) used by the model to represent power consumption of the application

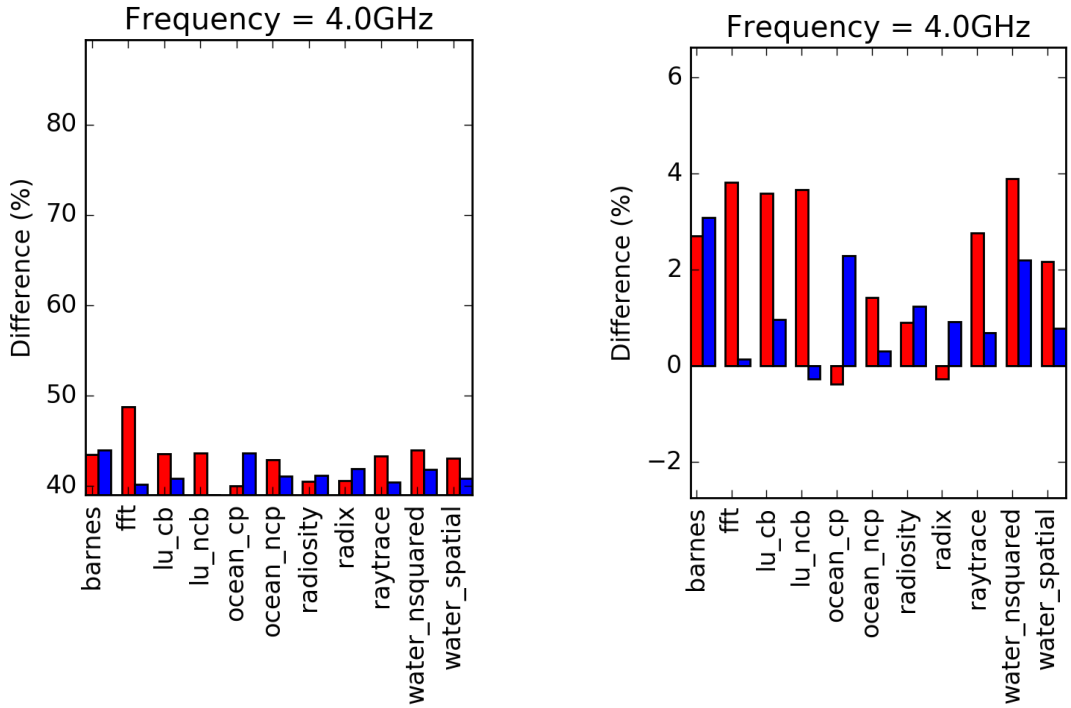
There is almost perfect correlation between RAPL and PIB values that can be approximated by a quadratic fit[4]. The outliers are three benchmarks *integer sort* (`is.C`), `STREAM` and *multi-grid* (`mg.C`). These three benchmarks are using uncore events that are not covered by RAPL.

Secondly, in order to calculate coefficients A, B and C on Intel Xeon E5 v4, we ran NPB and STREAM benchmarks at CPU frequencies ranging from 1.2 GHz to 2.3 GHz in steps of 0.1 GHz and performed a linear regression using the obtained results and equation 2. We used the nominal frequency of 2.0 GHz as a baseline frequency (f_0). For IBM POWER8 we have used CPU frequencies ranging from 2.5 GHz to 4.0 GHz and nominal frequency of 3.4 GHz. Since Cavium ThunderX does not support power gating, we eliminated implementation of this microarchitecture in this test. As soon as we obtained the coefficients, we estimated accuracy of the model by comparing estimated and actual power consumption for the benchmarks used by regression analysis. Figure 2 shows the difference between the estimated power consumption using a linear regression model and actual power reading from IBM POWER8 for all NPB and STREAM benchmarks and frequencies from 2.5 GHz to 4.0 GHz.

If the model is good, we expect most of benchmarks under various frequencies to be in one of the shades of green or light blue because in this case the difference between estimated and actual power is close to zero. This is applicable for such benchmarks as *unstructured adaptive mesh* (`ua.C`), *lower-upper Gauss-Seidel solver* (`lu.C`) and *block tri-diagonal solver* (`bt.C`). On the other hand, for such benchmarks as *data cube* (`dc.B`) and `mg.C`, accuracy depends on the CPU frequency. At higher frequency, the model overestimates the consumed power, while at lower frequencies it underestimates it. It also should be noted that for benchmarks `ft.C` and *embarrassingly parallel* (`ep.D`) the model estimates power quite accurately at high frequencies while it overestimates it at lower frequencies. Both of these behaviours could be caused by choosing the nominal frequency (namely, 3.4 GHz) as a baseline frequency. Furthermore, we can also notice on Fig. 2 that NPB and STREAM benchmarks do not cover the whole spectrum of possible combinations. For example, we can see that none of the benchmarks are in the top-right corner, where there is a high TPI and high average power consumption; and also, none are in the bottom-left corner where there is a low TPI and mid- to low-power consumption. This shows that some benchmarks might be over- or underestimated because they belong to an uncovered region.

Thirdly, the model provided in [2] uses power consumption averaged over the application's runtime. This approximation works well when there is no significant power variability during its

[4]The model of quadratic fit is $-0.0003 \times \ x^2 + 1.2614 \times \ x + 12.3405$



(a) Regression model

(b) Neural Network-based model

Figure 4. Power estimation accuracy of SPLASH2 benchmark suite on IBM POWER8

operation. But that is not always the case. Shown in blue in Fig. 3 is the shape of the running power during execution of the benchmarks, and shown in red is the constant power that is used for building the model. While comparing Fig. 3 with Fig. 2, We noticed that that model is fairly accurate for such benchmarks as `ep.D`, where there is an overlap between the blue and the red lines. Benchmarks such as `bt.C`, `lu.C` and `ua.C` belong to this category as well. For benchmarks in which it is not the case, such as `dc.B` and `STREAM`, we noted significant differences. This can have a significant impact on accuracy of the estimation.

Finally, in order to conduct further testing of the model, we ran all benchmarks from the SPLASH2 [28] benchmark suite. None of the results from these benchmarks were used by the linear regression model to find coefficients. Figure 4 shows results for POWER8 at the frequency of 4.0 GHz. The red bar is the whole benchmark run (including sequential and parallel regions of the code); the blue bar indicates that only parallel region of the code is measured. With only parallel region measured, the CPU utilisation is higher and, as a result, we expect to see higher accuracy of the power estimation model.

Figure 4 (a) shows that the model based on equation (2) provides accuracy of power evaluation only between 30% and 40%. It is a clear indication that the linear regression model has difficulty in covering a broader range of application with different execution profiles. Figure 4 (b) shows that the NN-based model is accurate within 5% for exactly the same range of parameters.

In the next Section, we will discuss in details how to use the Neural Network-based approach to improve accuracy of power estimation. In order to improve the accuracy as shown in analysis in this Section, it is not enough just to focus on performance counters that account for executed instructions and data transfers. If we apply the Neural Network-based model developed in the next Section on hardware performance counters from Tab. 2, we drop power consumption misprediction to a further extent than the one shown in Fig. 4 (b). Furthermore, it is necessary to

increase the number of benchmarks in the training set in order to cover the wide spectrum of future application characteristics. In this regard, we start the next Section by describing additional hardware performance counters that we sampled in order to improve accuracy. We also review an extended benchmark set before describing the Neural Network model that utilities them.

3. The Neural Network Model

In this Section, we provide details of our Neural Network–based power consumption prediction model, as well as the training sets used and the model's validation. Our model improves linear regression model predictions while using the same hardware performance counters, and extends the model by taking into account a wider range of counters.

Our model is a proof–of–the-concept prototype that can make fine–grained power predictions. We used the same hardware given in Section 2 as described in Tab. 1. The MATLAB Neural Network Toolbox [11] is used for the modelling.

3.1. Hardware Performance Counters and Benchmarks

It is possible to reveal plenty of power consumption information by monitoring hardware performance counters because they directly measure events that correlate with the energy used during the application execution. The aim of our work is to find the minimum set of hardware performance counters that highly correlate with the amount of power consumed. As in [23], we looked at a large number of available hardware performance counters across three different micro architecture implementations and selected the following counters:

- **Instructions Per Cycle (IPC)** – it was previously shown that power consumption of a processor is highly dependent on this metric.
- **Dispatched/Fetched Instructions (IFETCH)** – the previous metric (IPC) only accounts for instructions that have been retired, but it doesn't take into account instructions that have been speculatively executed. These instructions still consume power. Therefore, we keep track of them using this counter.
- **Stalls (STALL)** – due to multiple issues and out-of-order execution, contemporary processors *stall* due to dependencies such as data and resource conflicts. The conflicts draw power and are not accounted by any of the previous counters.
- **Branch hit ratio (BR)** – in order to find contribution to power consumed of speculatively executed instructions due to branch misprediction, we also measure percentage of correctly predicted branches during the application execution and use that information to further refine the results from **IFETCH** and **STALL** hardware performance counters.
- **Floating point instructions (FLOPS)** – for HPC applications, the largest contributor towards power consumption are instructions that are utilising the floating point unit as they represent the majority of executed instructions.
- **Cache and memory hit and miss** – once there is a miss, the processor needs to bring data from the memory in order to operate on it; the power is required to move this data. Due to the fact that with the previous counters we only measured power that is consumed within the processor, we also measure the number of hits in local cache (**L1**) and the number of misses in the shared last level cache (**LCCM**).

Table 3. Hardware performance counters used on Intel Xeon E5 v4, IBM POWER8 and Cavium ThunderX microarchitecture implementations to record metrics as described in Section 3.1

Event	Intel Xeon E5 v4	IBM S822LC	Cavium ThunderX
IPC	EVENT_CPU_CLK_UNHALTED EVENT_INST_RETIRED	PM_RUN_CYC PM_RUN_INST_CMPL	CPU_CYCLES INST_RETIRED
IFETCH	EVENT_ISSUED	PM_INST_DISP	ISSUE
STALL	EVENT_RESOURCE_STALLS	PMU_CMPLU_STALL	STALL_BACKEND
BR	EVENT_BR_INST_EXEC EVENT_BR_MISP_EXEC	PM_BR_CMPL PM_BM_MPRED_CMPL	BR_RETIRED BR_MIS_RETIRED
FLOPS	FP_ARITH_INST	PM_FLOP	ASE_SPEC VFP_SPEC
L1	MEM_LOAD_RETIRED_L1_HIT	PM_DATA_FROM_L2	L1D_CACHE_REFILL L1D_CACHE
LCCM	UNC_CBO_CCACHE_LOOKUP.ANY_REQ UNC_CBO_CCACHE_LOOKUP.I UNC_ARB_TRK_REQUEST.EVICTIONS	PM_MEM_READ PM_MEM_PREF PM_MEM_RWITM	L2D_CACHE_REFILL_LD L2D_CACHE_REFULL_ST L2D_CACHE_WB_VICTIM L2D_CACHE_WB_CLEAN

The hardware performance counters that we have obtained are intuitive. Power consumption of the processors depends on the number of directly executed instructions (**IPC**). The power used to move data and instructions from memories before execution is accounted by **L1** and the **IFETCH** counter, while moving data from further away is modelled by **LCCM** and the processor stalls (**STALL**) whilst it is waiting for data to become available. Furthermore, the accuracy of the processor stalls is improved by tracking correctly predicted branches (**BR**) as well. Finally, since the focus of this work lies on HPC–like workloads, we are also tracking the floating point instructions (**FLOPS**) since they represent the majority of instructions in these types of workloads. Table 3 shows the events that we sampled for each of the three microarchitecture implementations shown in Tab. 1.

In order to increase the size of the training set for our model, we have used extra **four** benchmark suites in addition to the NPB benchmark suite. The first two benchmark suites are Splash-2 [28] and the Princeton Application Repository for Shared–Memory Computers (PARSEC) [6]. Splash2 is a mature benchmark suite that contains a wide range of HPC and graphic applications, while PARSEC consists of benchmarks which are representatives of *emerging* workloads found in domains of data mining and media processing. We used 12 benchmarks from each of Splash2 and Parsec suites . The third benchmark suite is Mantevo [15]. Mantevo pioneered the concept of using *miniapps* to drive hardware/software co–design. The miniapps are proxies which combine some or all aspects of dominant computational kernels into standalone applications. We used 12 miniapps from Mantevo benchmark suite, which cover domains of finite elements, molecular dynamics, contact detection and electrical circuits. The fourth benchmark suite is a set of proxy applications from LLNL [16] that represent Monte Carlo particle transport, radiation diffusion and *Livermore Loops*. We used 7 benchmarks from this suite.

3.2. Model Description and Validation

The computational neural network (NN) is inspired by biological neural networks to predict or approximate functions that can depend on a large number of inputs. There are two phases when in using a neural network: training and deployment. During the training phase, neurons in each layer are adjusted iteratively using the training data. Then, the trained neurons are used to predict the new output in the deployment phase. Fig. 5 shows our neural network design. First,

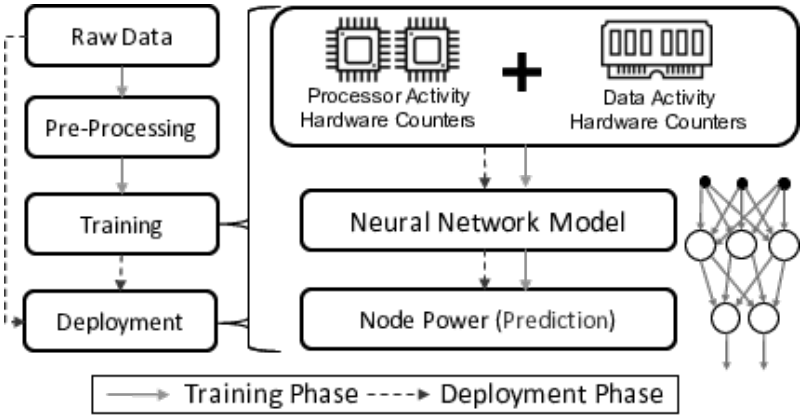


Figure 5. Neural Network–based prediction approach

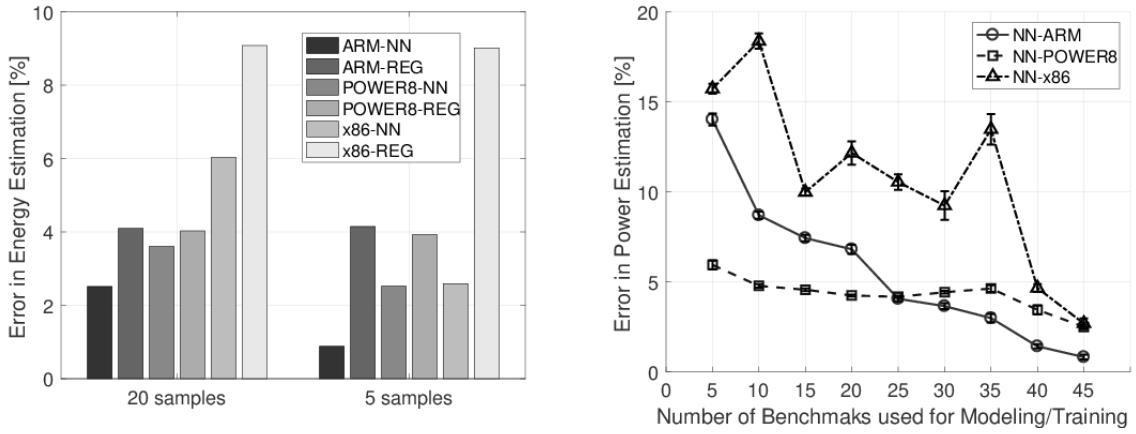
input data is pre-processed to obtain steady-state power. Second, pre-processed input data is fed to the input layer of the neural network for training.

In our case, the input layer consists of processor activity hardware counters and data activity hardware counters as shown in Tab. 3, which we identified to be the major sources of power draw. An entire data set was collected for three different microarchitectures using five benchmark suites. In order to have the same basis for the comparison between the neural network and a regression model, we updated the regression model from Section 2.1 to take into account hardware performance counters that we sampled as shown in Tab. 3. We conducted experiments with different numbers of runs per each benchmark to test the impact of data set size on the neural network model's accuracy. As shown in Fig. 6 (a), we noted that if the same benchmarks are run multiple times, overall accuracy of the NN model increases. Figure 6 (b) shows that accuracy of the NN model increases as we increase the number of different benchmarks in the training set.

If we 5 times run each benchmark from each suite on Cavium ThunderX ARM implementation, the error in power consumption estimation is around 3%, while if each benchmark is ran 20 times, the error drops below 1%. We noticed no further increase in accuracy by increasing the number of runs beyond 20, which indicates a convergence threshold for the NN model. Neither accuracy improvement was observed by the regression model (REG) as well.

Accuracy improvement with an increased number of runs of the same benchmark occurs due to the fact that a benchmark's profile differs from run to run as illustrated in Fig. 7. For example, by using the Cavium ThunderX microarchitecture we obtain a different spread of values for **IFETCH** counter between different runs of PARSEC benchmarks. Such variance in data provides additional data points for NN model training, which improves its accuracy. As mentioned in the previous paragraph, we determined that the ideal number of runs for each benchmark from each suite is 20. It should be noted that data variance/noise has a negative impact on accuracy of the linear regression model.

After the training phase, the neural network is deployed to provide node power predictions. The information flow of the training phase and the deployment phase is shown in Fig. 5 as a solid blue line and a dotted red line, respectively. In a neural network, each connection between neighbouring layers has a weight to scale data and a bias that allows shifting the activation function. Data points from the input layer are inserted as inputs to the next consecutive layers (hidden layers). Then, the hidden layers sum the data fed to them, scale (weight) the data, and process it until the data reaches the last layer that outputs the predicted node power:



(a) The number of each benchmark runs (b) The number of benchmarks

Figure 6. Power estimation accuracy as a function of benchmark runs and number of benchmarks

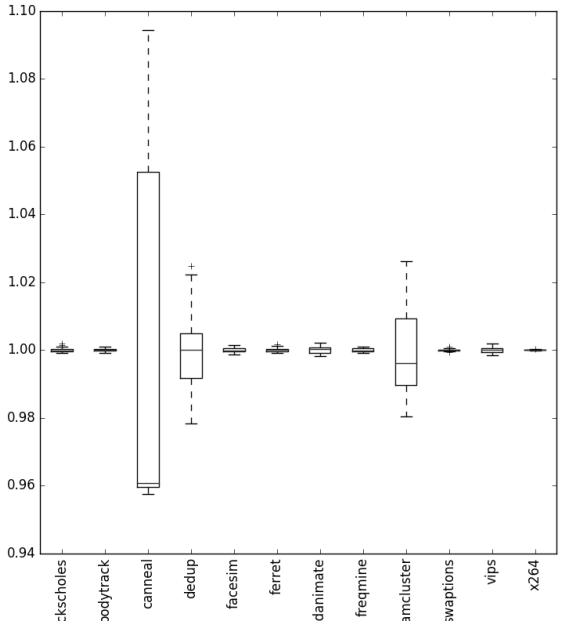
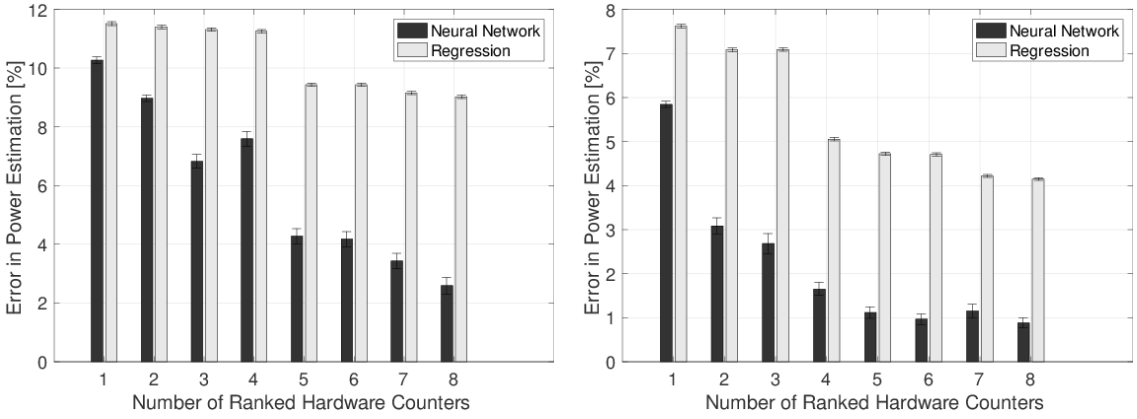


Figure 7. The variance of **IFETCH** hardware performance counter on Cavium ThunderX microarchitecture implementation



(a) Intel 64 (b) Cavium ThunderX

Figure 8. Error in power estimation between regression and neural network model for Intel 64 and Cavium ThunderX ARM microarchitecture implementations

$$w_{ij}(k+1) = w_{ij}(k) - \eta \frac{\delta e_k}{\delta w_{ij}}, \tag{3}$$

where η is the learning rate parameter which determines the rate of learning. w_{ij} represents the scalar value of weight on the connection from layer i to j. e_k represents the error of NN at k^{th} iteration. $\delta e_k / \delta w_{ij}$ determines the weighted search direction for this iterative method.

The weights and biases of the network are updated only after the entire training set has been applied to the network. We used 10–fold cross-validation to validate our model where 90% of randomly selected data is used as a training set and 10% is used as a validation set. The gradients calculated for each training set are added together to determine the change in weights, and biases. The weights and biases are updated in the direction of the negative gradient of the performance function.

For our neural network model we have tried three commonly used back–propagation algorithms: *Levenberg–Marquardt* [21], *Scaled conjugate gradient* [20], *Resilient* [22], and *Bayesian Regularization*. The Bayesian Regularization algorithm performs the best in terms of accuracy (showing the minimum mean error) but has more computational overhead (showing higher training time). Scaled conjugate gradient and Resilient perform the best in terms of computational overhead, showing less time for training than the other algorithms. As a result of this analysis, we decided to use the well-balanced Levenberg–Marquardt back propagation algorithm for all our experiments. However, the Bayesian Regularization back–propagation can be used provided the model's accuracy is a constrain. Scaled conjugate gradient or resilient algorithms can be used provided the learning overhead is a constraint.

Figures 8 and 9 show results that we have obtained on three micro-architectures. For each microarchitecture, the NN model provides higher accuracy compared to the regression model. The largest improvement in accuracy was observed for Cavium ThunderX ARM architecture as it is the simplest architecture to model with an in–order processor and a simplified two–level cache hierarchy. Even though this microarchitecture is the simplest of the three studied, the linear regression model is less accurate than the neural network model. Note that both models are least accurate on Intel 64 microarchitecture, while they are the most accurate on IBM POWER8 microarchitecture. This is due to the fact that performance counters which we identified in Section 3.1 are better indicators of power consumption on IBM POWER8. As part of our further work, we are planning to research which counters are better suited for Intel 64 and

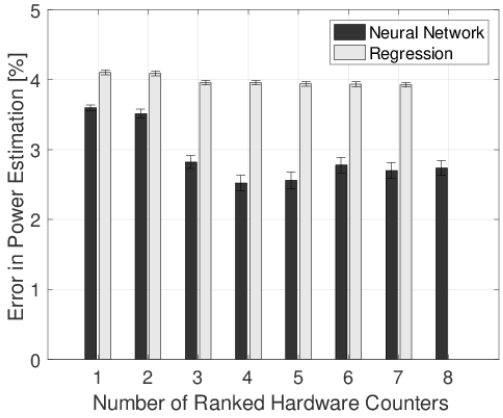


Figure 9. Error in power estimation between LR and NN models for POWER88 micro-architecture implementation

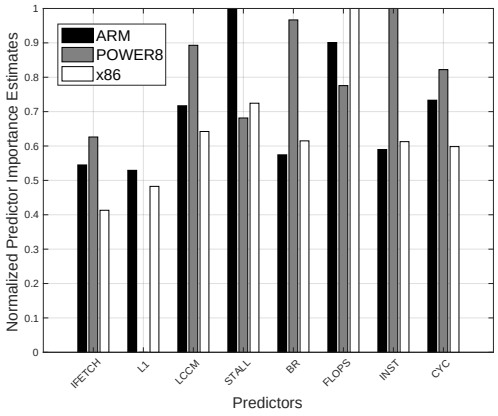


Figure 10. Importance of each hardware performance counters in power consumption estimation using the neural network model

Cavium ThunderX architectures. It is also important to note that significant improvements in accuracy for all three different microarchitectures stop around the fifth *most important* hardware counter. Figure 10 shows importance of each hardware performance counter from Section 3.1 for power consumption estimation.

Figure 10 suggests that it would be very difficult to use a neural network model developed for one architecture in order to estimate power consumption on the other architecture. Importance of different hardware counters for power consumption estimation is different for different microarchitectures. For example, Intel 64 and Cavium ThunderX microarchitectures are of similar importance to the same hardware performance counters except for **STALL** and **FLOPS**. **STALL** is more important for Cavium ThunderX micro-architecture because it is an in-order processor, and once there is a stall, instructions throughput and power consumption drop significantly. On the other hand, the floating point unit on Intel 64 is optimised for a very high throughput, and as a result, it consumes plenty of power on the floating-point heavy code. IBM POWER8 micro-architecture implementation behaves differently from Intel 64 and Cavium ThunderX. Since IBM POWER8 has been optimised for a high throughput, the counters that reflect the number of executed instructions and correctly predicted branches are the most important. Also, since the power cost of a cache that is missing and/or shared in a local is high, the access to remote memory (**LLC**) is very important when measuring power consumption. Note

Table 4. The error in estimating power on IBM POWER8 S822LC and Cavium ThunderX ARM when using the neural network model trained on Intel 64 microarchitecture implementation

Microarchitecture Implementation	IBM POWER8 S822LC	Cavium ThunderX ARM
Error in power estimation (%)	77%	64%

that due to a very well-developed pre-fetching mechanism on IBM POWER8 microarchitecture, local cache hits (**L1**) do not contribute to power consumption estimation on this architecture.

Nevertheless, we tried to apply a model trained on results from Intel 64 to estimate power of benchmarks' ran on Cavium ThunderX and IBM POWER8 microarchitecture implementations. The results are shown in Tab. 4. As expected, since Intel64 and Cavium ThunderX give similar importance to the same hardware performance counters, the power estimation on Cavium ThunderX microarchitecture implementation in Tab. 4 is more accurate then on IBM POWER8. Unfortunately, the accuracy is significantly worse when compared to Fig. 8 and 9. This is mainly due to the difference in importance of the same hardware performance counters that each microarchitecture implementation assigns. Furthermore, since the regression model is better off then the neural network model in both cases, there is space for improvement in the training phase of the neural network model in order to improve prediction accuracy. As part of the further work, we are planning to add results from Tab. 4 as part of training to the neural network model in order to make the neural network aware of different weights that each microarchitecture implementation assigns to the semantically same hardware performance counters.

Conclusions and Future Work

We demonstrated how to use hardware counters to develop models of power consumption for a broad range of HPC applications on three different microarchitecture implementations: Intel 64, IBM POWER8 and Cavium ThunderX ARMv8.

We demonstrated 2× to 3× better accuracy predictions using the NN model for power consumption compared to the LR model.

We provided a comparative analysis of the importance of different hardware counters for power consumption across three microarchitectures. We believe that results of our work can contribute to software/hardware co–design efforts towards developing unified, cross-architectural models to predict power consumption.

References

1. Acun, B., Lee, E.K., Park, Y., V. Kalé, L.: Neural Network-Based Task Scheduling with Preemptive Fan Control. In: International Workshop on Energy Efficient Supercomputing (E2SC). ACM (2016), DOI: 10.1109/E2SC.2016.016

2. Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., Huber, H., Panda, R., Thomas, F., Wilde, T.: A Case Study of Energy Aware Scheduling on SuperMUC. In: Supercomputing – 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings. pp. 394–409 (2014), DOI: 10.1007/978-3-319-07518-1_25

3. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The NAS Parallel Benchmarks - Summary and Preliminary Results. In: Proceedings of the 1991 ACM/IEEE Conference on Supercomputing. pp. 158–165. Supercomputing '91 (1991), DOI: 10.1145/125826.125925

4. Bansal, N., Lahiri, K., Raghunathan, A., Chakradhar, S.T.: Power Monitors: A Framework for System-Level Power Estimation Using Heterogeneous Power Models. In: VLSI Design. pp. 579–585. IEEE Computer Society (2005), DOI: 10.1109/icvd.2005.138

5. Bhat, S.G.: Openpower based Inband OCC sensors. `https://github.com/shilpasri/-inband_sensors` (2017), accessed: 2018-08-31

6. Bienia, C.: Benchmarking Modern Multiprocessors. Ph.D. thesis, Princeton University (2011)

7. Bircher, W.L., John, L.K.: Complete System Power Estimation Using Processor Performance Events. IEEE Trans. Comput. 61(4), 563–577 (2012), DOI: 10.1109/tc.2011.47

8. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Predictive Modeling for Job Power Consumption in HPC Systems. In: High Performance Computing – 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings. pp. 181–199 (2016), DOI: 10.1007/978-3-319-41321-1_10

9. Burtscher, M., Zecena, I., Zong, Z.: Measuring GPU Power with the K20 Built-in Sensor. In: Proceedings of Workshop on General Purpose Processing Using GPUs. pp. 28:28–28:36. GPGPU-7, ACM, New York, NY, USA (2014), DOI: 10.1145/2588768.2576783

10. Contreras, G., Martonosi, M.: Power Prediction for Intel XScale®Processors Using Performance Monitoring Unit Events. In: Proceedings of the 2005 International Symposium on Low Power Electronics and Design. pp. 221–226. ISLPED '05 (2005), DOI: 10.1109/lpe.2005.195518

11. Demuth, H., Beale, M.: Neural Network Toolbox for Use with MATLAB. `http://www.mathworks.com/help/nnet/` (2017), accessed: 2018-08-31

12. Duy, T.V.T., Sato, Y., Inoguchi, Y.: Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In: Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. pp. 1–8. IEEE (2010), DOI: 10.1109/ipdpsw.2010.5470908

13. Elisseev, V., Baker, J., Morgan, N., Brochard, L., Hewitt, T.: Energy Aware Scheduling Study on BlueWonder. In: 4th International Workshop on Energy Efficient Supercomputing, E2SC@SC 2016, Salt Lake City, UT, USA, November 14, 2016. pp. 61–68 (2016), DOI: 10.1109/e2sc.2016.014

14. Eranian, S.: Perfmon2: a flexible performance monitoring interface for Linux. In: Proceedings of the Ottawa Linux Symposium. pp. 269–288 (2006), http://perfmon2.sourceforge.net/ols2006-perfmon2.pdf

15. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Edwards, H.C., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving Performance via Mini-applications. Tech. Rep. SAND2009-5574, Sandia National Laboratories (2009), DOI: 10.2172/993908

16. Heroux, Michael A and Neely, Rob, and Swaminarayan, Sriram: ASC Co-Design Proxy App Strategy. Tech. Rep. LLNL-TR-592878, Los Alamos National Laboratory (2013), DOI: 10.2172/1055856

17. Huang, W., Lefurgy, C., Kuk, W., Buyuktosunoglu, A., Floyd, M., Rajamani, K., Allen-Ware, M., Brock, B.: Accurate Fine-Grained Processor Power Proxies. In: Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 224–234. IEEE (2012), DOI: 10.1109/micro.2012.29

18. Li, T., John, L.K.: Run-time Modeling and Estimation of Operating System Power Consumption. In: Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. pp. 160–171. SIGMETRICS '03 (2003), DOI: 10.1145/781047.781048

19. McCalpin, J.D.: Memory Bandwidth and Machine Balance in Current High Performance Computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter pp. 19–25 (1995)

20. Moller, M.F.: A scaled conjugate gradient algorithm for fast supervised learning. NEURAL NETWORKS 6(4), 525–533 (1993), DOI: 10.1016/s0893-6080(05)80056-5

21. More, J.J.: The Levenberg–Marquardt Algorithm: Implementation and Theory. Numerical Analysis, ed. G. A. Watson, Lecture Notes in Mathematics 630, Springer Verlag pp. 105–116 (1977), DOI: 10.1007/bfb0067700

22. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: Neural Networks, 1993., IEEE International Conference on. pp. 586–591 (1993), DOI: 10.1109/icnn.1993.298623

23. Rodrigues, R., Annamalai, A., Koren, I., Kundu, S.: A Study on the Use of Performance Counters to Estimate Power in Microprocessors. IEEE Trans. on Circuits and Systems 60-II(12), 882–886 (2013), DOI: 10.1109/tcsii.2013.2285966

24. Intel: OpenMPI. https://www.open-mpi.org/ (2017), accessed: 2018-08-31

25. Texas Instruments: System Power Management and Protection IC With PMBus™. Texas Instruments (2013)

26. The Green500: The Green Lists. https://www.top500.org/green500/lists/2017/11 (2017), accessed: 2018-08-31

27. Tiwari, A., Laurenzano, M.A., Carrington, L., Snavely, A.: Modeling power and energy usage of HPC kernels. In: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. pp. 990–998. IEEE (2012), DOI: 10.1109/ipdpsw.2012.121

28. Woo, S.C., et al.: The SPLASH-2 Programs: Characterization and Methodological Considerations. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture. pp. 24–36. ISCA '95, ACM, New York, NY, USA (1995), DOI: 10.1109/isca.1995.524546