University of Reading

An analytical methodology to derive power models based on hardware and software metrics

Article

Accepted Version

Dolz, M. F., Kunkel, J., Chasapis, K. and Catalán, S. (2016) An analytical methodology to derive power models based on hardware and software metrics. Computer Science - Research and Development, 31 (4). pp. 165-174. ISSN 1865-2034 doi: 10.1007/s00450-015-0298-8 Available at https://centaur.reading.ac.uk/77677/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

To link to this article DOI: http://dx.doi.org/10.1007/s00450-015-0298-8

Publisher: Springer

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

University of Reading

CentAUR

Central Archive at the University of Reading

Reading's research outputs online

Noname manuscript No.
(will be inserted by the editor)

An Analytical Methodology to Derive Power Models based on Hardware and Software Metrics

Manuel F. Dolz · Julian Kunkel · Konstantinos Chasapis · Sandra Catalán

Received: date / Accepted: date

Abstract The use of models to predict the power consumption of a system is an appealing alternative to wattmeters since they avoid hardware costs and are easy to deploy. In this paper, we present an analytical methodology to build models with a reduced number of features in order to estimate power consumption at node level. We aim at building simple power models by performing a per-component analysis (CPU, memory, network, I/O) through the execution of four standard benchmarks. While they are executed, information from all the available hardware counters and resource utilization metrics provided by the system is collected. Based on correlations among the recorded metrics and their correlation with the instantaneous power, our methodology allows *i)* to identify the significant metrics; and *ii)* to assign weights to the selected metrics in order to derive reduced models. The reduction also aims at extracting models that are based on a set of hardware counters and utilization metrics that can be obtained simultaneously and, thus, can be gathered and computed on-line. The utility of our procedure is validated using real-life applications on an Intel Sandy Bridge architecture.

M. F. Dolz, K. Chasapis
Dept. of Informatics
University of Hamburg, 22.527–Hamburg, Germany
E-mail: manuel.dolz@informatik.uni-hamburg.de

J. Kunkel
German Climate Computing Center
DKRZ GmbH, 20.146–Hamburg, Germany
E-mail: kunkel@dkrz.de

S. Catalán
Depto. de Ingeniería y Ciencia de Computadores
Universitat Jaume I, 12.071–Castellón, Spain
E-mail: catalans@icc.uji.es

Keywords HPC · power modeling · hardware/software metrics · statistical analysis · multicore processors.

1 Introduction

The clock frequency of microprocessors was increasing for several decades allowing us to improve the systems performance following the Dennard scaling, which formulates the ability to maintain a constant power density while reducing the size of the transistors. However, after the breakdown of this law around 2006, power consumption came in the spotlight, since shrinking the size of the transistors created a greater thermal runaway and, consequently, a higher power consumption. For this reason, reducing power consumption has been identified as an essential challenge that the HPC (High Performance Computing) community has to face in order to pave the way towards the Exascale era [1].

One of the firsts steps to address the energy challenge is to foster a deep understanding of the power consumed by the major components of the system (CPU, memory, network, I/O) by measuring and modeling it. According to [18], we consider the following characteristics as important for modelling power:

- *Accuracy.* The models should be precise enough to evaluate strategies for reducing energy consumption.
- *Simplicity.* The prediction should be computed fast, avoiding significant overhead on the target machine and being easy to understand.
- *Inexpensiveness.* The framework/devices employed to collect measures and to apply models should not be expensive nor time consuming.
- *Portability.* The model should be applicable to many platforms/architectures.

The final publication is available at Springer via http://dx.doi.org/10.1007/s00450-015-0298-8

For instance, a model featuring the aforementioned characteristics could easily be exploited to make power-aware scheduling with the aim of reducing the power consumption while preserving performance. Nevertheless, information about the system and its actual power consumption is needed in order to provide good estimations at runtime. Nowadays, microprocessors offer more than 200 performance counters, each quantifying a specific hardware event such as L1 cache misses. A vast effort has been spent to build tools that permit to retrieve these performance monitoring counters and power consumption information about the system. PAPI ("Performance Application Programming Interface") [16] or LIKWID[1], present new APIs in order to extract information about the processor's events from the hardware performance counters.

Apart from CPU counters, certain behavior of the architecture (e.g., I/O, network usage, etc.) and the actual node's power consumption need also to be acquired. Therefore, monitoring tools like `top`, `htop` or `iotop` provide valuable operating system statistics. In the last years, the process of relating power consumption to the applications' behavior has been improved with the establishment of power measurement tools, e.g., `pmlib` [2] and PowerPack [10].

Generally, building accurate and simple power models is a crucial task to further understand architectural power behavior, allowing the use of power-/energy-saving techniques. The main contributions of this work are *i)* the building of models that estimate the total instantaneous power consumed by a platform using the hardware performance counters and the resource utilization information; *ii)* a methodology that identifies the most appropriate parameters in order to derive reduced power models; and *iii)* the validation of the models on a recent architecture.

The rest of the paper is organized as follows. In Section 2, we describe related work in the area in order to motivate the paper. Simplified power models and our analytical methodology to derive reduced power models is described in Section 3. Afterwards, the models are validated against different workloads in Section 4. Finally, we complete this work with a few concluding remarks 5.

2 Related Work

Work related to this paper can be classified into *i)* models for energy consumption based on hardware and software characteristics and *ii)* tools for estimating energy metrics based on such models.

[1] `http://code.google.com/p/likwid/`

In the first group, several works can be found following different approaches. R. Bertran et al. describe power models for multi-core processors using performance counters [5] targeting only the CPU power consumption to construct a linear power model that is later validated with SPECcpu2006 suite. On the other hand, in [4] hardware counters and specific benchmarks to stress the memory and the CPU are used to account the thread-specific energy. Our work differs from those since we aim at modeling at node level, just as targeted in [7] where a methodology to estimate power consumption using machine learning techniques is proposed. In their work, a set of predefined "Key Performance Indicators" is used to build a model that takes into account processor usage, memory accesses and network utilization. Although our target is similar, we follow a different strategy; instead of selecting the performance indicators a priori, we deploy a methodology based on empirical observation to identify the most relevant ones. Finally, Rivoire et. al [18] compare full system power models and conclude that those taking into account OS utilization metrics and performance counters are more accurate.

In the second group, we identify tools that make use of models. In [15], the authors present a simulator which replays recorded utilization traces of applications and uses a simplified power model based on utilization metrics to estimate the performance. Another example of model application is presented in [13] with vEC ("Virtual energy counters"). This tool can be used to estimate energy consumption (CPU, bus, cache and memory) relying on the analytical model presented in [14]. Finally, authors in [9] present a power analysis per system component and the "Mantis" tool, a real time predictor of power consumption. This tool operates in three stages: 1) utilization metrics and hardware counters are collected, 2) the model parameters are derived and fed accordingly to this information and, 3) the parameters are adjusted based on the executed workload.

Finally, the authors in [6] present a methodology to reduce the number of power lines to be monitored from internal DC wattmeters. They aim for building reduced power models in order to estimate the total power consumption of the platform. Their methodology clusters correlated power lines and selects one representant per cluster, which, in turn, will be a parameter of the final model. Our work follows a similar approach, however we expose a more sophisticated methodology that is performed in two steps and we target hardware and software metrics for modelling power.

3 Analytical derivation of reduced models

Our motivation of building reduced models is given by the impossibility to access many hardware counters at once. Moreover, complex models processing too many hardware and software counters could increase power consumption, as has been observed in the recent literature [3]. Our aim is to discard redundant features from the model, and to obtain accurate values for the coefficients that remain valid for any application and core count. In this paper, we construct linear models using an analytical approach: after building the training and validation sets, we perform a two-step workflow in order to drop redundant features and reduce the complexity of the model without compromising their accuracy. Firstly, we filter and group features exhibiting similar behavior and assign a representant to each group. Then we sort the representants relative to each group's correlation to power. Finally, we select the best correlated features to derive reduced models according to the characteristics described in Section 1.

3.1 Linear model

Let us consider that the total power consumption by the system should be computed for time intervals of length k. During the time interval $t = ((t-1)\cdot k, t\cdot k]$, the power consumption $P_T(t)$ is given by

$$P_T(t) \approx c + \sum_{i=1}^{n} f_i(t)\cdot c_i \tag{1}$$

where $f_i(t)$ denotes the value of the i-th feature during that interval and c_i is a corresponding fixed model coefficient. An estimate for a constant contribution to power is given by c.

Indeed, we aim at estimating the total instantaneous power with a reduced number of features. Therefore, assuming a system offers n available features, $f = \{f_1, f_2, \ldots, f_n\}$, we can replace (1) to use only r features, being $r \ll n$, and re-computing the coefficients so that the model is still able to produce good estimations of the total power consumption.

3.2 Building training and validation sets

In order to calibrate the fixed coefficients in Equation 1, we use a multi-core server platform comprised of an Intel Xeon CPU "Sandy Bridge" E31275 processor with 4 cores running at 3.40 GHz (with the `performance` governor and active Turbo Boost[2]), 16 GB of DDR3 RAM (1333 MHz), one 160 GB Intel SSD and an Intel PRO/1000 Gigabit network card. From this platform, we collect the following information:

- *Power consumption* is captured in a frequency of 20 Hz from an external ZES-ZIMMER LMG450 wattmeter, a highly advanced precision wattmeter, using the `pmlib` framework [2].
- *Hardware counters* are gathered at 10 Hz[3] leveraging `likwid-perfctr` command from the LIKWID tool with the timeline option set. In this architecture, we have identified 220 hardware counters that are accessible through `PMC`, `FIXC` and `PWR` registers. Among them, `PWR` registers are used by the Intel RAPL interface, therefore providing access to the estimated power consumption of the socket [8].
- *Operating system statistics and temperature sensors* are also retrieved at 10 Hz using an instance of the `pmlib` server reading CPU, memory, network and I/O utilization and temperature. Operating system statistics are retrieved using the `psutil` Python library, while temperature sensors are accessed using the `pysensors` library interfacing the `lm_sensors` kernel module. On this architecture, the ACPI interface (virtual device) and socket/core temperatures are available.

In order to emulate the different phases of an application and to stress the different components of the architecture, we selected the following benchmarks:

- `linpack`[4]: This pre-compiled linear algebra code from Intel contains the optimized LINPACK benchmark. Internally using MKL libraries, it performs FPU/ALU instructions in purpose of stressing the CPU.
- `stream`[5]: This benchmark is intended to obtain the best possible memory bandwidth by means of simple vector kernels.
- `iperf`[6]: This tool performs network throughput measurements. We test both a server and a client running TCP throughput tests.
- `IOR`[7]: This benchmark tool is used for benchmarking POSIX performance of a local SSD.

To gather the data for training and validation, we execute k-combinations with repetition (with $k = 4$ cores) of the aforementioned benchmarks for 60 s. For each combination of benchmarks, we collect hardware counters, OS statistics and temperature sensors. Since

[2] We cover Turbo Boost on purpose, as several HPC centers enable it for specific workloads.

[3] We consider 10 samples/s sufficient enough for our experiments, ensuring negligible overhead on the total power consumption due to monitoring processes [3].

[4] `https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download`

[5] `https://www.cs.virginia.edu/stream/`

[6] `https://iperf.fr/`

[7] `http://sourceforge.net/projects/ior-sio/`

we cannot measure all hardware counters simultaneously, each combination is run more than 50 times, each time capturing a different set of counters. Between the runs of I/O sensitive benchmarks we clear the cache to retain similar start conditions[8]. Next, we use Python scripts to merge data, check for consistency and availability of all features, drop overflowing values[9] and interpolate the data of the features to the timestamps of the measured power.

Also, we derive averaged (node-level) values of features that have been collected at core level. Finally, we build a matrix in which the rows contain samples gathered of each combination of benchmarks and the columns the values of each feature collected. In total, $n = 253$ features are captured for 78 benchmark runs in the training set and 21 runs in the validation set resulting in a matrix of $112,303 \times 253$ (250 MiB CSV file).

3.3 Metric-filtering algorithm

The filtering step is implemented in the statistical tool R and proceeds as follows:

1. As for the first step, we compute a correlation matrix m of dimension $n \times n$, where entry m_{ij} denotes the linear dependence between feature i and j. Set $h = h_0$.
2. We classify the features into a small number of disjoint clusters using the k-means algorithm, $C = \{c_1, c_2, \ldots, c_r\}$. All features belonging to a cluster must have a correlation threshold of h. Thus, k-means is called in a loop fashion to form $1, 2, 3, \ldots, n$ clusters with the initial training set.
3. When a formed cluster has a correlation equal or greater than h among all their features, we found a relevant group. First, we remove all previous representants from the cluster. Then the feature with the highest sum of correlation is determined as representant and stored in a separate list. All group members except the representant are then purged from the matrix. The representant is kept to ensure we are not loosing a feature relevant for forming subsequent clusters. We go to Step 2. If no more clusters can be extracted we go to Step 4.
4. If $h > 0.5$, we reduce h to $h = h \cdot h_0$ in order to capture less related features and goto Step 2. Otherwise terminate: features that were not grouped so far are also considered as representants.

[8] Due to identical start conditions, we assume the counters of repeated runs behave similarly. Based on our results, this assumption seems justified.

[9] About four samples per 600 contain overflowing counters.

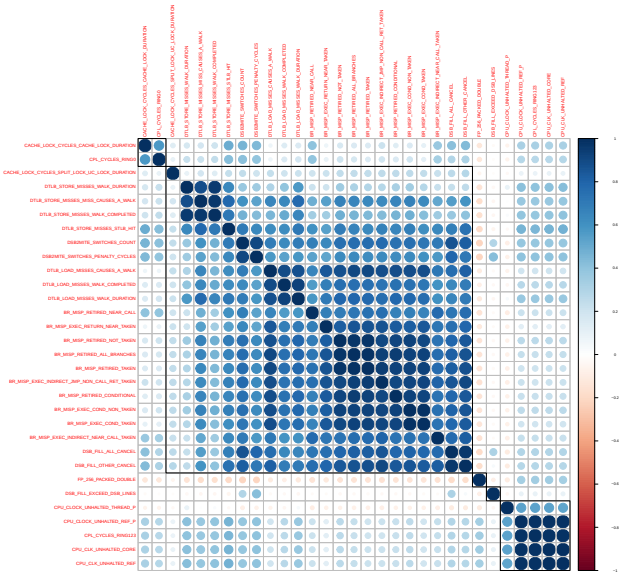


Fig. 1: Partial correlation matrix obtained from the training set. Blue and red circles stand for positive and negative correlations, respectively.

The purpose of the algorithm is to extract a representant for each cluster that is unique to any other feature. As an example, Figure 1 shows a block of a correlation matrix forming 5 clusters. The features of the top left and bottom right are well correlated with each other and behave similar in respect to other features. The big group in the middle contains features showing slightly different behavior in respect to other features.

3.4 Building reduced models

The last step of our methodology is to assemble reduced models using the subset of representative features obtained in the previous step. The obtained representants are sorted by correlation with respect to the measured power consumption. In order to reduce the number of combinations and derive reduced power models, we set a second threshold f, taking only the f features with the highest correlation to the measured power consumption. This guarantees that only those that are highly correlated with the target of the model will be part of the reduced models. In fact, the linear least-square method delivers good estimations without redundant features but correlated to the estimated variable [17].

In order to estimate the corresponding coefficients, we leverage multiple linear regression. Since there are r representative features, there are r-combinations, i.e., $c_T = \prod_{s=1}^{r} \#c_s$ of representative counters. However, taking into account that in this specific architecture there are only 4 `PMC` and 3 `FIXC` (fixed) registers that

#g	Corr.	Group representant	Other metrics
1	0.970	PWR_PKG_ENERGY (PKG_ENERGY)	SENSORS_PHY_ID.0, PWR_PP0_ENERGY, SENSORS_CPU
2	0.906	CPU_CLK_UNHALTED_CORE (UNHC)	CPL_CYCLES_RING123, CPU_CLK_UNHALTED_CORE, CPU_CLK_UNHALTED_REF, CPU_CLOCK_UNHALTED_REF_P
3	0.881	L1D_BLOCKS_BANK_-CONFLICT_CYCLES (L1CC)	L1D_BLOCKS_BANK_CONFLICT_CYCLES, L2_TRANS_ALL_REQUESTS
4	0.836	L2_RQSTS_MISS (L2RM)	L2_TRANS_DEMAND_DATA_RD
5	0.832	L2_RQSTS_ALL_PF (L2RQ)	L2_TRANS_ALL_PREF
6	0.826	HW_PRE_REQ_DL1_MISS (DL1M)	HW_PRE_REQ_DL1_MISS, L1D_REPLACEMENT
7	0.812	L2_LINES_OUT_DEMAND_CLEAN (L2DC)	OFFCORE_REQUESTS_DEMAND_DATA_RD
8	0.810	UOPS_DISPATCHED_CORE (UOPSD)	MEM_UOP_RETIRED_LOADS, UOPS_DISPATCHED_PORT_PORT_2_LD, UOPS_DISPATCHED_PORT_PORT_3_LD, UOPS_DISPATCHED_THREAD, UOPS_RETIRED_ALL
9	0.801	INSTR_RETIRED_ANY (INRA)	INST_RETIRED_PREC_DIST, INST_RETIRED_ANY_P, MEMLOAD_UOPS_RETIRED_L1_HIT, UOPS_ISSUED_ANY, UOPS_RETIRED_RETIRE_SLOTS
10	0.781	L2_LINES_IN_E (L2LI)	L2_LINES_IN_ALL, L2_RQSTS_PF_MISS, L2_TRANS_L2_FILL, OFFCORE_REQUESTS_ALL_DATA_RD
11	0.773	UOPS_DISPATCHED_PORT_PORT_0 (UOPS0)	UOPS_DISPATCHED_PORT_PORT_1
12	0.762	RESOURCE_STALLS_RS (RSRS)	CPU_UTIL
13	0.753	UOPS_DISPATCHED_PORT_PORT_2 (UOPS2)	UOPS_DISPATCHED_PORT_PORT_3
14	0.744	L2_RQSTS_ALL_DEMAND_DATA_RD (L2DD)	L2_RQSTS_ALL_DEM_AND_DATA_RD_HIT
15	0.675	INT_MISC_STALL_CYCLES (INTS)	RESOURCE_STALLS_ANY

Table 1: List of the 15 most correlated representants in respect to power with $h = 0.95$ for the correlation threshold between clusters.

can be read at once, apart from OS statistics and temperature, only combinations of groups of 4 representants are tested. With our methodology, we automatically determined and extracted the top $f = 15$ representants in respect to the correlation of their group to power consumption. The automatically identified representants and the features belonging to their group with $h = 0.95$ are given in Table 1.

4 Validation

To assess the quality and benefit of the models created by our methodology, we employ Quantum Espresso [11], a software suite for ab initio quantum chemistry methods of electronic-structure calculation and materials modeling, executing a set of 17 different experiments running on 4 cores of the machine using MPI. Also, our validation set contains the compilation of the Linux kernel v3.19.2 by executing `make -j` with 1 to 4 cores.

4.1 Basic models

In order to assess the obtained models, we used a set of a simple baseline models:

- *Average power.* A (bad) prediction for power is the mean value of the power. We expect that any model to be considered valuable should perform better than this naive approach.
- *Single hardware counters.* For example, the RAPL counter `PWR_PKG_ENERGY` should provide good estimations for system performance in a linear model.
- *CPU utilization.* Using the utilization statistics of the CPUs, as reported by the OS, should provide fair estimations, CPUs power consumption constitutes more than 50% of the total power consumption on typical systems [1].
- *OS statistics.* Statistics provided by the OS are comprised of utilization values from the memory subsystem, I/O and network. Since utilization of components is expected to correlate with their contribution to the power consumption, they may give good estimations of the total power consumption.
- *Temperature sensors.* The Poole-Frenkel effect explains how the power-consumption of static leakage power increases with the temperature. It is independent of clockspeed and solely dependent on temperature and voltage [12].
- *CPU utilization, OS and sensors.* Combinations of previous metrics may improve the predictions.

4.2 Evaluation of the models

In order to evaluate the reliability of our approach, we compare derived models against the set of basic models. All models are created on the training set and evaluated on the validation data set. Table 2 collects statistics of the prediction accuracy of each model. The Mean and Max columns represent the average and maximum value of the average absolute error, while Q1 and Q3 correspond to the first and third quartiles of the same, respectively. Also root-mean-square error (RMSE) is included, it is sensitive to huge absolute errors. The last, the coefficient of determination, (R^2) determines the goodness of the fit provided by the model. It is a rough estimate for the fraction of points that is explained by the model, thus, an R^2 of 1 indicates that the model explains all observations, and 0 indicates the opposite.

Note that baseline models are represented from B0 to B5, reduced models working solely on hardware counters range M0 to M5, and combinations of the latter with OS statistics and temperature are represented from D0 to D9 on the table. Furthermore, since there is a large number of combinations among the selected features and, due to the reduced number of hardware counters that can be read at once, we selected the most interesting models under those conditions.

As can be seen, the naive baseline model B0, average power, leads to almost 0 R^2, confirming this is a bad estimate. Nevertheless, it achieves a mean error

#m	Model	Q1	Mean	Q3	Max	RMSE	R^2
B0	AVG_POWER	20.5	24.4	27.3	46.7	24.3	0.00
B1	CPU_UTIL	11.3	13.6	17.1	59.7	15.1	0.61
B2	OS	27.9	31.3	34.4	90.5	31.2	0.88
B3	SENS	9.0	11.4	13.3	86.9	17.9	0.97
B4	OS + SENS	5.5	7.5	12.7	88.2	16.6	0.98
B5	RAPL_PKG_POWER	1.4	4.8	6.4	51.6	6.0	0.96
B6	CPL_CYCLES_RING123	14.4	19.2	24.6	98.1	22.2	0.89
B7	CPU_CLK_UNHALTED_CORE	14.2	19.0	24.5	98.1	20.6	0.89
B8	MEM_UOP_RETIRED_LOADS	8.6	16.8	25.2	99.2	21.0	0.74
M0	PKG_ENERGY + L2DC + UOPS + INTS	0.7	1.5	2.9	50.8	4.5	0.98
M1	PKG_ENERGY + L2DC + UNHC + DL1M	2.1	4.5	6.8	48.0	6.5	0.98
M2	UNHC + L2RM + L2DC + L2LI	2.3	5.0	9.2	77.3	10.0	0.96
M3	UNHC + L1CC + L2RQ + UOPSD	2.1	4.4	10.8	70.7	10.1	0.95
M4	L2RQ + RSRS + INTS + UOPS2	6.5	15.9	25.0	104.1	20.0	0.95
M5	INRA + UNHR + UNHC + L2RQ + RSRS + UOPS2 + INTS	9.4	15.2	23.0	88.0	18.5	0.96
D0	L2DC + UOPS0 + INTS + SENS	1.5	3.1	5.5	70.6	12.1	0.98
D1	L2DC + UOPS0 + INTS + SENS + OS	1.6	3.2	7.0	76.0	12.0	0.99
D2	UNHC + DL1M + L2DC + SENS	4.1	6.3	9.1	73.6	12.0	0.98
D3	UNHC + DL1M + L2DC + SENS + OS	1.1	2.5	4.9	79.9	11.8	0.99
D4	L2RQ + RSRS + UOPS2 + INTS + OS	4.1	8.5	14.9	97.4	13.2	0.96
D5	L2RQ + RSRS + UOPS2 + INTS + SENS	3.7	7.9	12.7	77.6	13.8	0.98
D6	L2RQ + RSRS + UOPS2 + INTS + SENS + OS	2.3	4.0	7.8	81.0	12.2	0.99
D7	INRA + UNHR + UNHC + L2RQ + RSRS + UOPS2 + INTS + OS	3.6	7.7	13.2	97.1	12.1	0.97
D8	INRA + UNHR + UNHC + L2RQ + RSRS + UOPS2 + INTS + SENS	5.9	10.8	16.3	92.0	15.8	0.98
D9	INRA + UNHR + UNHC + L2RQ + RSRS + UOPS2 + INTS + SENS + OS	2.1	6.1	9.1	84.9	12.8	0.99

Table 2: Absolute error of baseline, simple and combined models against the training and validation sets.

of 24.4 W and RMSE of 24.3 which serves as reference. Improved baseline models (from B1 to B4), based on sensors (SENS) improve the accuracy, while CPU utilization only and the OS statistics are not able to explain the behavior of the validation set. In fact, the OS behavior, which also includes CPU utilization, leads to a worse model than B0, this is presumably due to overfitting of the training data. On the other hand, some selected hardware counters working alone (models from B6 to B7) are not enough to mimic power consumption, their R^2 ranges from 0.74 to 0.89 yet their mean error is more than 16 W. The model built on top of the RAPL interface achieves R^2 of 0.96 and a mean error of 4.8 W, which is considerable well. Nevertheless, for the sake of portability, our models should not always depend on the RAPL interface, despite its goodness.

Finally, models created with our algorithmic methodology that work solely with hardware counters (from M0 to M5) provide a good R^2, ranging from 0.95 to 0.98. They provide similar accuracy than RAPL and if used together can improve the accuracy down to 1.5 W. Although their high accuracy, we learned that complementing them with OS statistics and/or temperature sensors (models from D0 to D9), we may even increase their accuracy. It is important to remark that, R^2 alone is not sufficient to assess the model quality, e.g., M3 and M4 achieve a similar value but the mean error of the latter is higher.

As complementary information, Figure 2 plots power traces in order to compare the real (green points) and estimated (black points) power consumption using some of the baseline cases, ranging from models that work with only one feature to the most elaborated ones combining hardware counters, OS statistics and temperature sensors. We include some benchmarks (training set) but also cases from the validation set. Even though the model shown in each row is created on our training set, we can see interesting differences in the quality of the different models for predicting even the training set. For example, CPU_UTIL alone cannot predict well to the power consumption of both training and validation sets.

Figure 3 shows statistics for the test set and the predicted average. The training set for this experiment is either our general training set. For the training applications we also include the *other* set which excludes the particular benchmark but includes all other data. The difference between the prediction accuracy of *p-train* and *p-other* is an indicator for the importance of a particular benchmark for building the model with the selected features. It can be seen, for example, that iperf client, IOR read and write are well predicted and do not contribute much to the CPU_CLK_UNHALTED_CORE model. For the L2DC+UOPS0+INTS+OS+SENS model, both IOR read and write are more important. In either case, the idle run is revealing important behavior that needs to be trained. The model built on CPU_CLOCK_UNHALTED_CORE is not fitting the idle case well, it also overestimates the power of the validation set. As expected this feature alone is not sufficient.

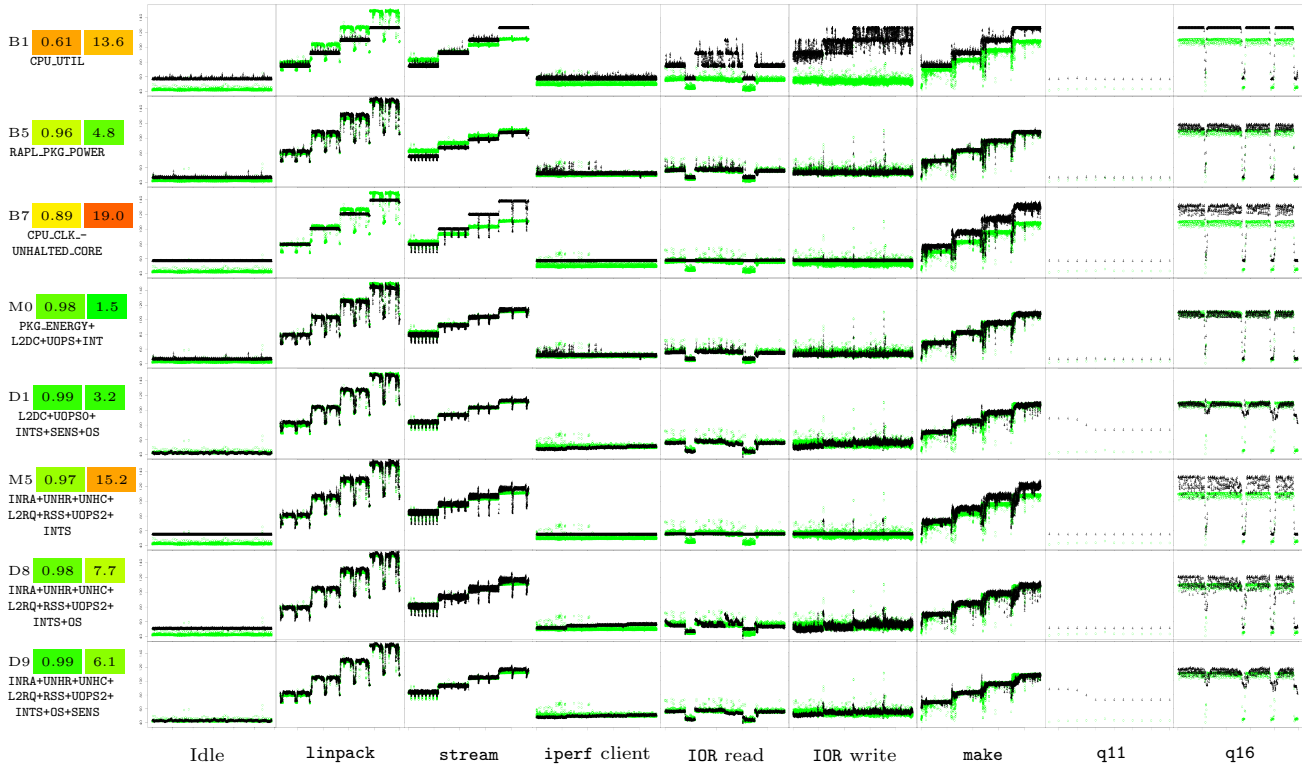
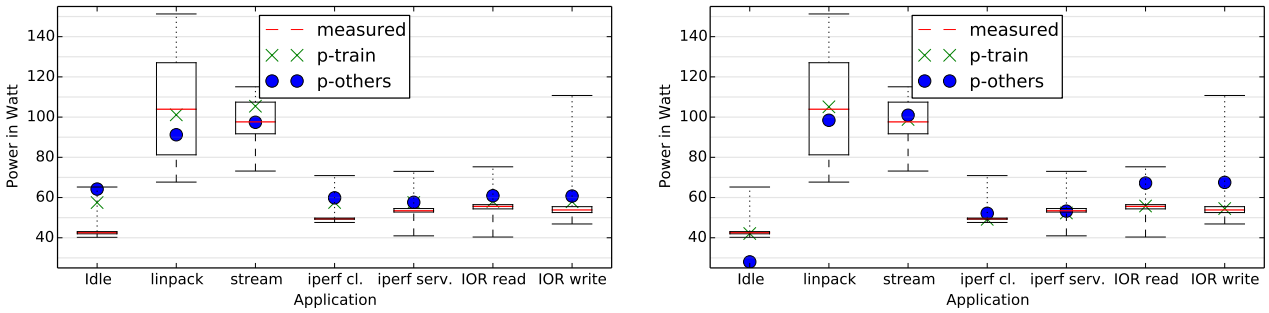


Fig. 2: Real (green) and estimated (black) power consumption traces of some of the training and validation benchmarks using models from the Table 2. The R^2 and average error is given as reference after the model number. Note that the X-axis represents the timeline of samples while the Y-axis measures power from 38 to 160 W.

(a) Training set with `CPU_CLK_UNHALTED_CORE`

(b) Training set with `L2DC+UOPS0+INTS+OS+SENS`



(c) Training set with `CPU_CLK_UNHALTED_CORE`

(d) Training set with `L2DC+UOPS0+INTS+OS+SENS`

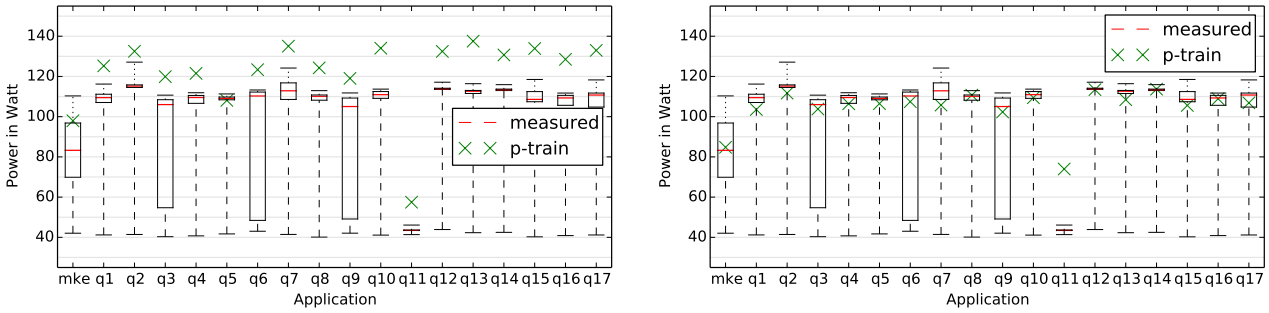


Fig. 3: Statistics for the validation sets using the models `CPU_CLK_UNHALTED_CORE` and `L2DC+UOPS0+INTS+OS+SENS`.

5 Concluding remarks

We presented an effective methodology to determine hardware and software metrics that are important when building power models. With this, a system administrator could alternatively run a daemon, constantly reading those metrics and estimating power consumption, with negligible overhead on the system. Thus, in practice, these models can eventually replace physical wattmeters that are currently necessary to monitor machines on an HPC platform.

Our methodology is comprised of a simple but efficient way to construct derived models. First, a calibration step with a few benchmarks is run in order to stress different components of the architecture. It comprises combinations of benchmarks with variation in number of cores used. Each combination is run for 60 s and continuously repeated to collect all hardware and software metrics available on the system. Additionally, we employ the R statistical tool to process all data in order to automatically apply our filtering algorithm. Finally, we manually combine a number of representants that can be accessed at once on our test system to derive our reduced model. Our model is architecture dependent, so the methodology and the models generated should be derived at platform level, however, the methodology can be applied on other architectures.

The results demonstrate that the automatically derived models are accurate. It can be observed, that by selecting the most appropriate hardware counters, the goodness of the fit measured in terms of R^2 ranges from 0.96 to 0.99, with a mean error down to 3 W without using RAPL counters. By adding OS statistics and especially temperature sensors to the model, the accuracy can be increased potentially. On the contrary, simple models based on temperature and/or OS statistics can yet provide fairly acceptable estimations.

In the future, we aim to *i)* Integrate features for DVFS and multi-socket machines to the model, *ii)* Develop a kernel module that can provide the estimated power in the `proc` interface, and *iii)* Automate the tool and provide it as a free and portable package.

Acknowledgements This work was supported by the EU Project FP7 318793 "EXA2GREEN" and the German Helmholtz Association's LSDMA project. We would like to thank Dr. Julen Larrucea, from DKRZ GmbH, for his technical support in configuring the Quantum Espresso software.

References

1. Ashby, S., *et al*: The opportunities and challenges of Exascale computing. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (2010)
2. Barreda, M., Barrachina, S., Catalán, S., Dolz, M.F., Fabregat, G., Mayo, R., Quintana, E.S.: A framework for power-performance analysis of parallel scientific applications. In: Third Int. Conference on Smart Grids, Green Communications and IT Energy-aware Technologies – Energy 2013 (2013)
3. Barreda, M., Cataln, S., Dolz, M.F., Mayo, R., Quintana-Ortí, E.S.: Automatic detection of power bottlenecks in parallel scientific applications. Computer Science - Research and Development **29**(3-4) (2014)
4. Bellosa, F.: The benefits of event: Driven energy accounting in power-sensitive systems. In: Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System, pp. 37–42. ACM (2000)
5. Bertran, R., Gonzalez, M., Martorell, X., Navarro, N., Ayguade, E.: Decomposable and responsive power models for multicore processors using performance counters. In: Proceedings of the 24th ACM International Conference on Supercomputing, pp. 147–158. ACM (2010)
6. Castaño, M., Catalán, S., Mayo, R., Quintana-Ortí, E.: Reducing the cost of power monitoring with dc wattmeters. Computer Science - Research and Development (2014)
7. Cupertino, L., Da Costa, G., Pierson, J.M.: Towards a generic power estimator. Computer Science - Research and Development pp. 1–9 (2014)
8. David, H., Gorbatov, E., Hanebutte, U., Khanna, R., Le, C.: RAPL: memory power estimation and capping. In: 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED) (2010)
9. Economou, D., Rivoire, S., Kozyrakis, C., Ranganathan, P.: Full-system power analysis and modeling for server environments. In: Workshop on Modeling Benchmarking and Simulation (MOBS), pp. 13–23 (2006)
10. Ge, R., Feng, X., Song, S., Chang, H.C., Li, D., Cameron, K.W.: Powerpack: Energy profiling and analysis of high-performance systems and applications. IEEE Trans. Parallel Distrib. Syst. **21**(5), 658–671 (2010)
11. Giannozzi, P., et al.: Quantum espresso: a modular and open-source software project for quantum simulations of materials. Journal of Physics: Condensed Matter (2009)
12. The Effect of Temperature on Power-Consumption with the Intel i7-2600K. URL `http://forums.anandtech.com/showthread.php?p=32451313`
13. Kadayif, I., Chinoda, T., Kandemir, M., Vijaykirsnan, N., Irwin, M.J., Sivasubramaniam, A.: vec: Virtual energy counters. In: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, pp. 28–31. ACM (2001)
14. Kim, H., Vijaykrishnan, N., Kandemir, M., Irwin, M.: Multiple access caches: Energy implications. In: VLSI, 2000. Proceedings. IEEE Computer Society Workshop on, pp. 53–58 (2000)
15. Minartz, T., Kunkel, J., Ludwig, T.: Simulation of power consumption of energy efficient cluster hardware. Computer Science - Research and Development (2010)
16. Mucci, P.J., Browne, S., Deane, C., Ho, G.: Papi: A portable interface to hardware performance counters. In: Proceedings of the Department of Defense HPCMP Users Group Conference, pp. 7–10 (1999)
17. Neter, J., Kutner, M.H., Nachtsheim, C.J., Wasserman, W.: Applied Linear Statistical Models. Irwin (1996)
18. Rivoire, S., Ranganathan, P., Kozyrakis, C.: A comparison of high-level full-system power models. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems, pp. 3–3. USENIX Association (2008)