# Understanding Hardware and Software Metrics with respect to Power Consumption

Julian Kunkel[a], Manuel F. Dolz[b]

[a]*German Climate Computing Center, DKRZ GmbH, 20.146–Hamburg, Germany*
[b]*Dept. of Computer Science, University Carlos III of Madrid, 28.911–Leganés, Spain*

**Abstract**

Analyzing and understanding energy consumption of applications is an important task which allows researchers to develop novel strategies for optimizing and conserving energy. A typical methodology is to reduce the complexity of real systems and applications by developing a simplified performance model from observed behavior. In the literature, many of these models are known; however, inherent to any simplification is that some measured data cannot be explained well. While analyzing a models accuracy, it is highly important to identify the properties of such prediction errors. Such knowledge can then be used to improve the model or to optimize the benchmarks used for training the model parameters. For such a benchmark suite, it is important that the benchmarks cover all the aspects of system behavior to avoid overfitting of the model for certain scenarios. It is not trivial to identify the overlap between the benchmarks and answer the question if a benchmark causes different hardware behavior. Inspection of all the available hardware and software counters by humans is a tedious task given the large amount of real-time data they produce.

In this paper, we utilize statistical techniques to foster understand and investigate hardware counters as potential indicators of energy behavior. We capture hardware and software counters including power with a fixed frequency and analyze the resulting timelines of these measurements. The concepts introduced can be applied to any set of measurements in order to compare them to another set of measurements. We demonstrate how these techniques can aid identifying interesting behavior and significantly reducing the number of features that must be inspected. Next, we propose counters that can potentially be used for building linear models for predicting with a relative accuracy of 3%. Finally, we validate the completeness of a benchmark suite, from the point of view of using the available architectural components, for generating accurate models.

*Keywords:* HPC, data analysis, power modeling, statistical methods, performance counters, energy consumption.

## 1. Introduction

Power and energy consumption have been identified as the single largest challenges in the design of future Exascale high-performance computing (HPC) systems [1]. Basically, the vast increase of levels of parallelism to the point of millions of processors working concurrently is a challenge that will need radical changes in hardware and software design (e.g., programming models, compilers, I/O libraries, etc.) [2]. Thus, understanding how computers use power is key to develop a new hardware and software stack in order to face the Exascale challenge.

Nevertheless, the implementation and deployment of Exascale systems calls for a holistic approach that, with the use of power and performance tracing tools and wattmeters, allows the inspection of power bottlenecks and energy hotspots of current scientific parallel software. However, acquisition costs and deployment

*Email addresses:* `kunkel@dkrz.de` (Julian Kunkel), `mdolz@inf.uc3m.es` (Manuel F. Dolz)

The final publication is available at ScienceDirect via the DOI 10.1016/j.suscom.2017.10.016

of power measurement devices can be, due to the nature of the platforms and number of nodes, infeasible. Recent research has significantly demonstrated that a promising alternative in order to mitigate this issue is the design of power models [3, 4, 5]. Taking into account that most of the current processors feature a large set of hardware counters, temperature sensors, and resource usage statistics provided by the operating system, one could cleverly use this information to predict power drawn by individual components and system power consumption. For instance, a per-component power model could be easily exploited to make energy-aware scheduling with the aim of reducing the power consumption while preserving performance [6].

As highlighted in [7], a good power model should always be accurate, simple, inexpensive and portable. Indeed, recent works have shown that many power models can fulfill all these properties while providing fairly good estimations [3, 4, 5]. In this sense, a classical methodology to validate the accuracy of power models has only been carried out by calculating the preciseness and responsiveness of their predictions [8]. However, this approach may lead to optimistic results because often *i)* the static power is included when computing relative accuracy, *ii)* the model is applied to complete application runs where statistical effects negate individual errors, *iii)* the selection of similar training and validation data may lead to optimistic errors, *iv)* a few strong outliers of the model may still lead to a good relative accuracy. Thus, while analyzing model accuracy is important, we believe it is even more relevant to identify the properties of such predictors, i.e., relations between metrics in order to understand the source of these errors. Therefore, in this paper we aim at understanding and identifying behavior of hardware counters in order to explain and model power consumption. We also validate the completeness of a benchmark set, from the point of view of utilizing available architectural features, for generating accurate models. Note that we leave the building of power models part as future work.

The paper is structured as follows: First, we present related work in Section 2. In Section 3, we describe the statistical methods that are used to draw conclusions over a set of hardware counters and benchmarks. Afterwards, in Section 4 we demonstrate their use on several experiments to *i)* identify properties for the highest and lowest energy consumption, *ii)* localize outliers of a linear model based on the Intel RAPL (Running Average Power Limit) interface, *iii)* identify different phases of the `linpack` benchmark, and *iv)* investigate the impact of benchmarks that encompass a training suite for models. Finally, we conclude the paper in Section 5.

## 2. Related Work

We classify the work related to this paper in three different categories: *i)* building of power and energy models using statistical analysis using hardware counters; *ii)* techniques leveraging power models to dynamically limit energy consumption; and *iii)* integrated interfaces in current architectures to provide on-line power measurements.

In the literature, we find a large collection of works using statistical analysis for building power and energy models based on hardware counters. For instance, the approach by Xiao et al. [9] presents a methodology for building system-level power models based on regression analysis without the need of power measurements at component level. In this sense, the regression models describe the aggregate power consumption of the processors, the wireless network interface and the display using hardware performance counters. Similarly, Shang et al. [10] present an efficient adaptive regression-based high-level power model to estimate FPGA power consumption. In order to improve on-line power estimation accuracy, they use adaptive regression methods to lessen the problem of biased training sequences and to finally achieve a good trade-off between efficiency and accuracy. To deal with the accuracy issue, the work by McCullough et al. [11] investigated on the accuracy of hardware counters-based models and concluded that the inherent complexity of the system architectures and the current microprocessors are, in most of the cases, the root cause of the model errors. Nevertheless, they stated that hardware counters are, as of today, the only source to obtain fine-grain information about the platform.

Research leveraging power models to dynamically limit energy consumption can also be found in the literature. For instance, we encounter works using models for controlling dynamic voltage and frequency scaling (DVFS) [12] and for guiding compilers to generate energy-efficient codes [13]. Similar works have also used models to reduce power in system components other than processors, such as RAM [14] and disks [15]. Researchers have also analyzed the impact of techniques using such models to control a single knob —either DVFS or dynamic concurrency throttling (DCT)— for dynamic power management on shared-memory [16, 17], and on distributed-memory parallel systems [18, 19]. The work from Curtis-Maury et al. [20] differs from earlier research since it uses multiple knobs in several key aspects, such as for DVFS and DCT. To do so, this work proposes methods to generalize multi-dimensional prediction models that leverage statistical analysis for estimating how DVFS and DCT influence the performance of applications.

On the other hand, we find that many hardware manufacturers have integrated hardware counters in order to provide on-line measurements and reduce the energy consumption. For example, the Intel Running Average Power Limit (RAPL) counters [21], the AMD Application Power Management (APM) interface [22] and the IBM Power7 interface [23] provide power measurements based on models and sensors of the processors. Besides, recent NVIDIA GPUs report power usage via the NVIDIA Management Library (NVML) [24]. It is important to highlight also the Intel Intelligent Platform Management Interface (IPMI) [25] which measures total server power using on-board sensors and supports the reading of additional sensors.

All in all, we conclude that the application of statistical methods for data analysis [26] in the domain of energy-efficiency has been used as a fundamental technique to derive power models. However, none of those studies has used advanced statistical methods prior ensuring that independent variables of the models are reliable and robust enough for building them. From the techniques we apply in this paper, and to the best of our knowledge, only the Pearson product-moment correlation coefficient has been previously used [27]. In this sense, the novelty of this paper lies in the use of statistical methods, normally applied to other sciences, to the specific field of power modeling. To some extent, as well, such knowledge can be used to improve the models or to develop benchmarks that aid in the design of models for future architectures. In sum, this paper uses advanced statistical methods *i)* to analyze measured data in much more detail, *ii)* to reveal interesting properties and *iii)* to validate the completeness of a benchmark set for generating accurate models.

## 3. Methodology

In this section, we explain in detail the methodology used to apply the statistical methods for analysis performed in this paper. First, we give details about the HPC plaform and the benchmark suite used to build the target data set. Next, we formalize this data set for further analysis and, finally, we describe three advanced statistical methods used along the paper.

### *3.1. Target platform and benchmark suite*

Our measurements were gathered on an Intel Xeon CPU "Sandy Bridge" E31275 processor with 4 cores running at 3.40 GHz (with the `performance` governor and active Turbo Boost[1]), and 16 GB of DDR3 RAM (1333 MHz).[2] We collected the following information:

1. *Power consumption* is captured with a frequency of 20 Hz from an external ZES-Zimmer LMG450 [28], a highly advanced precision wattmeter, using the PMLib framework [29].
2. *Hardware counters* are gathered at 10 Hz[3] leveraging the `likwid-perfctr` command from the LIKWID tool [4] with the timeline option set. In this architecture, we have identified 220 hardware counters that are accessible through `PMC`, `FIXC` and `PWR` registers, the latter capturing the RAPL socket power.
3. *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.

In order to emulate the different phases of an application and for utilizing different components of the architecture, we selected the following applications:

[1]We cover Turbo Boost on purpose, as several HPC centers enable it for specific workloads.
[2]Due to space limits, we are only able to carry out the evaluation using a single platform.
[3]We consider 10 samples/s sufficient enough for our experiments, ensuring negligible overhead on the total power consumption due to monitoring processes [30].
[4]`http://code.google.com/p/likwid/`

1. `idle`: To measure the idle state of the machine, we used the `sleep` Linux utility to suspend the executing process for an interval of time. This benchmark is denoted as `idl`.
2. `linpack`: This pre-compiled linear algebra code from Intel contains the optimized LINPACK [31] benchmark[5]. Internally using MKL libraries, it performs FPU/ALU instructions with the purpose of utilizing the CPU. In the following this application is denoted as `cpu`.
3. `stream`:[6] This benchmark is intended to obtain the best possible memory bandwidth by means of simple vector kernels. This benchmark is subsequently denoted as `mem`.
4. `iperf`: This tool[7] performs network throughput measurements. We test both a server and a client running TCP throughput tests, which are denoted as `nts` and `ntp`, respectively.
5. `IOR`:[8] This benchmark tool is used to investigate POSIX performance to a local SSD, and is designated as `ior` and `iow` for writes, respectively.
6. `Kernel make`: We compile the Linux kernel v3.19.2 by executing `make -j`. This benchmark is denoted as `mke`.
7. `Quantum Espresso`:[9] This is a software suite for electronic-structure calculation and materials modeling. From this suite, we used 17 different on four cores of the machine using MPI.

### *3.2. Building the data set*

To build the data set, we executed the benchmarks 1–7 individually from one to four cores. Next, we also ran combinations of benchmarks 1–5 concurrently on multiple cores. Over a runtime of 60 s, we collected, at the different allowed frequencies, the features of hardware counters, OS statistics and temperature sensors. (Note that for `Quantum Espresso` we captured the whole runtime of each experiment.) Since we could only measure 4 hardware counters simultaneously, each benchmark was run more than 60 times, each time capturing a new set of counters[10]. Among runs of I/O sensitive benchmarks, we cleared the cache to retain similar start conditions. We derived averaged values for features that were collected at core level for several reasons: 1) it prevents issues caused by migration of processes and threads between cores; 2) it is invariant to the number of available cores and executed processes; 3) it accounts to the fact that it does not matter on which logical core a process is executed. In total, for 99 experiments 252 features were recorded for 112,303 timestamps resulting in a CSV file of 295 MB.

Afterwards, we postprocessed the data using Python scripts in order to *i)* merge data from different benchmarks, *ii)* check for consistency and availability of all features, *iii)* drop overflowing values[11], and *iv)* interpolate the data of the features to the time stamps of the measured power.

To motivate further application of statistical methods, we formalize our target data set. As stated, this set consisted of 252 hardware and software features which resulted in sequences of 112,303 time intervals $I_t = ((t-1)\cdot k, t\cdot k]$ with duration $k$. Note that $k$ was calculated as the inverse of the frequency of the stream captured at the highest rate. Also, in each time interval, we relate the tuples of measurements as $m(t) = (f_1(t), \dots, f_i(t), \dots, f_n(t))$ where $f_i(t)$ is the value of the $i$-th feature for the time interval $t$. An excerpt of our data table, represented as a matrix, is given in Table 1.

### *3.3. Advanced statistical methods*

In the following, we describe the three advanced statistical methods that are used throughout the paper to identify the relations. (Note that for the sake of simplicity, we do not describe these methods in detail.) With them, the goal is to identify causal relations between two sets of measurements, where a set is defined as $S = \{m(t_i), m(t_j), ...\}$. For example, two interesting sets ($R$ and $O$) could be obtained by selecting measurements exhibiting regular performance behavior and the remaining measurements, i.e., the outliers. Using these methods, we can investigate the differences in the characteristics of $R$ and $O$.

[5] `https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download`
[6] `https://www.cs.virginia.edu/stream/`
[7] `https://iperf.fr/`
[8] `http://sourceforge.net/projects/ior-sio/`
[9] `http://www.quantum-espresso.org`
[10] Due to identical start conditions, we assume counters of repeated runs to behave similarly.
[11] About four samples per 600 contain overflowing counters.

| Application 0 | Application 1 | Application 2 | Application 3 | Time interval (s) | `AGU_BYPASS_CANCEL_COUNT` (Executed load operations) | `ARITH_FPU_DIV_ACTIVE` (Cycles that the divider is active) | `ARITH_NUM_DIV` (Number of divides) | `BR_INST_EXEC_ALL_BRANCHES` (Counts near executed branches) | sensors (°C) | `sensors_Phy_id_0` (°C) | CPU utilization (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| `ntc` | `mem` | `cpu` | `cpu` | 1.70 | 381 | 53940444 | 8991759 | 17780300 | 53.8 | 57.9 | 77.21 |
| `ntc` | `mem` | `cpu` | `cpu` | 1.75 | 373 | 53918367 | 8988070 | 17864184 | 53.8 | 58.0 | 77.50 |
| `ntc` | `mem` | `cpu` | `cpu` | 1.80 | 371 | 53956891 | 8994500 | 16592948 | 53.8 | 58.0 | 77.50 |
| `ntc` | `mem` | `cpu` | `cpu` | 1.85 | 373 | 54029089 | 9006552 | 14568724 | 53.6 | 58.0 | 77.42 |
| `ntc` | `mem` | `cpu` | `cpu` | 1.90 | 374 | 54017954 | 9004704 | 13824812 | 53.5 | 58.0 | 77.30 |
| ... | | | | | | | | | | | |
| `cpu` | `idl` | `idl` | `idl` | 1.70 | 384 | 29577022 | 4930281 | 1338917 | 44.0 | 46.0 | 27.32 |
| `cpu` | `idl` | `idl` | `idl` | 1.75 | 381 | 29622258 | 4937823 | 1338546 | 44.0 | 46.0 | 26.44 |
| `cpu` | `idl` | `idl` | `idl` | 1.80 | 379 | 29663007 | 4944617 | 1338024 | 44.0 | 46.0 | 25.19 |
| `cpu` | `idl` | `idl` | `idl` | 1.85 | 381 | 29683252 | 4947990 | 1336816 | 44.0 | 46.0 | 26.05 |
| `cpu` | `idl` | `idl` | `idl` | 1.90 | 382 | 29692978 | 4949610 | 1336118 | 44.0 | 46.0 | 27.30 |

Table 1: Excerpt of the data table with a few features (out of 252).

*Correlation..* Outliers of a performance prediction model can be identified using the residual error $err = observation - prediction$ and defining a threshold upon which a measurement is considered to be an outlier. Linear correlation between any feature and the prediction error can easily be obtained by computing the Pearson product-moment correlation coefficient [27]. When comparing two variables, a correlation coefficient of 1 indicates that those two variables behave alike, e.g., a hypothesis might be that power is correlated to the number of instructions processed. Similarly, a value of 0 shows that there is no linear correlation. This also allows for identifying features with a linear correlation to a model's error.

*Kolmogorov-Smirnov (KS) test..* In statistics, there are many hypothesis tests for evaluating whether observations are modeled by a certain process. Most of these tests impose certain requirements on the data and offer parameters that must be carefully adjusted. The Student's $t$-test, for example, allows for testing whether two data sets are significantly different; however, it needs normal distributed data. The Kolmogorov-Smirnov test [32], in contrast, is a non-parametric test which can be used for testing whether the empirical probability distribution of one data set behaves like that of another. Moreover, it supports testing of the probability distributions for one data set creates lower values than the distribution of another and returns a probability value ($p$-value). Thus, we may identify features of $O$ that typically show higher, lower or similar values to those of $R$. Once we identify that one feature is typically higher than another, the mean values can be compared to determine a ratio. We call this the *outliers-factor*. (OFactor is the mean value of outliers divided by the mean of the regular cases.) This method is much more robust than just comparing means of data sets.

*Principal component analysis (PCA)..* The PCA [33] is a procedure that converts the observations –with potentially correlated features– into a set of linearly uncorrelated variables called principal components. The principal components form an orthonormal basis of the data, where the extracted components are ordered by decreasing variance that is described. Each component $C(i)$ is a basis vector and describes the contribution of all features to this basis, $C(i) = (c_{i_1}, ..., c_{i_n})^T$. Usually a few components allow to describe the input data sufficiently well and later components contribute little to it. Therefore, the complexity of the analysis can be reduced to the analysis of the principal components. By using this technique, we can identify outliers or similarities between data sets such as benchmarks.

## 4. Analysis of the power consumption

In this section, we demonstrate the benefits of the aforementioned statistics methods and evaluate them on several examples using the gathered data set. As a simple exercise, we analyze the behavior of the measurements with the lowest and highest 10% power consumption. Then, we investigate cases in which a linear model based on RAPL cannot cover the power measurement well. Next, interesting phases of `linpack` are identified and assessed. Finally, the composition of our test suite's applications is investigated.

### *4.1. Power consumption*

At first, we investigate the correlation of any feature to the external power consumption. An excerpt of the features is given in Table 2. Next, we group the resulting correlated variables depending on the amount of their correlations. With the results, some observations can be made. For instance, we notice the weak negative correlation of I/O to power consumption. This is by the fact that `IOR` does not perform CPU-intense operations while doing I/O. Indeed, `IOR` it does not perform many CPU operations. Since CPU accounts for the highest contribution to energy consumption on typical platforms, synchronous I/O is expected to show negative correlation.

Our second observation is that features in the next group are not correlated at all, neither are the amount of free memory of the system and the page cache size. Also, the number of flushes of the second level/shared TLB (`TLB_FLUSH_STLB_ANY`) does not correlate to power. In the next group, there are features with a weak correlation, for example, the eviction rate on the L1 data cache. Finally, we focus on the last group, which shows highly correlated counters. For example, the number of clock cycles during which the core is busy and the CPU temperature are highly correlated to external power. Note that all these correlations apply to the full data set. It may happen that a feature is correlated according to this data without having a stringent causal connection.

| Correlation | Feature |
|---|---|
| -0.285 | `io_Write_bytes` |
| -0.057 | `net_Bytes_sent` |
| ... | ... |
| 0.001 | `memory_VM_Cached` |
| 0.004 | `ITLB_ITLB_FLUSH` |
| 0.006 | `MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP...` |
| 0.019 | `FP_ASSIST_SIMD_INPUT` |
| 0.023 | `memory_VM_Free` |
| 0.026 | `PARTIAL_RAT_STALLS_MUL_SINGLE_UOP` |
| 0.027 | `FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE` |
| 0.028 | `TLB_FLUSH_STLB_ANY` |
| ... | ... |
| 0.512 | `LOAD_HIT_PRE_SW_PF` |
| 0.520 | `L2_STORE_LOCK_RQSTS_HIT_E` |
| 0.525 | `L1D_ALL_M_REPLACEMENT` |
| 0.525 | `L1D_M_EVICT` |
| ... | ... |
| 0.903 | `CPL_CYCLES_RING123` |
| 0.904 | `L2_TRANS_ALL_REQUESTS` |
| 0.909 | `CPU_CLK_UNHALTED_REF` |
| 0.913 | `CPU_CLK_UNHALTED_CORE` |
| 0.951 | sensors (ACPI temperature of all cores) |
| 0.979 | `PWR_PKG_ENERGY` |

Table 2: Correlation of features to power. Lower case features are OS statistics, others are hardware counters with their LIKWID name.

#### *4.1.1. Analysis of the outliers*

In this section, we determine the 10% and 90% quantiles and consider the lowest and highest 10% of data points as two sets of outliers ($L$ and $H$). In our case, $L$ concerns measurements with less than 50.4 W and $H$ those higher than 117.2 W. Next, we compare these outliers to the other 90% of data points using the KS-test. Note also that we only consider $p$-values of at least 0.95. Table 3 shows a selection of features and their OFactor for $L$. The first part of the table shows that the identified features in $L$ have typically higher values than the others. For instance, it can be seen that the average value of `net_Bytes_recv` is lower but,

| OFactor | Feature |
|---|---|
| KS reports a higher probability distribution: | |
| 1.82 | net_Bytes_sent |
| 1.51 | net_Pack_sent |
| 1.23 | memory_VM_Available |
| 1.31 | net_Pack_recv |
| 0.43 | net_Bytes_recv |
| KS reports a lower probability distribution: | |
| 0.000 | OTHER_ASSISTS_AVX_TO_SSE |
| 0.002 | FP_256_PACKED_DOUBLE |
| 0.004 | io_Read_bytes |
| 0.005 | L2_LINES_OUT_PF_CLEAN |
| 0.005 | LOAD_HIT_PRE_HW_PF |
| 0.005 | RESOURCE_STALLS_ROB |
| 0.007 | L3_LAT_CACHE_MISS |
| 0.010 | INSTR_RETIRED_ANY |
| 0.012 | RESOURCE_STALLS2_OOO_RSRC |
| ... | ... |
| 0.042 | UOPS_DISPATCHED_PORT_PORT_0 |
| 0.043 | ARITH_NUM_DIV |
| 0.043 | UOPS_DISPATCHED_CORE |
| ... | ... |
| 0.508 | External power |
| 0.654 | sensors |

(a) Lowest 10%.

| OFactor | Feature |
|---|---|
| KS reports a higher probability distribution: | |
| 32.358 | OTHER_ASSISTS_AVX_TO_SSE |
| 6.572 | FP_256_PACKED_DOUBLE |
| 5.187 | L2_LINES_OUT_PF_CLEAN |
| 4.613 | LOAD_HIT_PRE_SW_PF |
| 4.271 | OFFCORE_REQUESTS_ALL_DATA_RD |
| ... | ... |
| 1.907 | RESOURCE_STALLS2_OOO_RSRC |
| 1.892 | RESOURCE_STALLS2_ALL_PRF_CONTROL |
| 1.825 | MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS |
| 1.823 | CPU utilization |
| 1.508 | External power |
| 1.452 | LOAD_BLOCKS_ALL_BLOCK |
| 1.314 | sensors_Phy_id_0 |
| 1.308 | sensors |
| KS reports a lower probability distribution: | |
| 0.000 | io_Read_bytes |
| 0.000 | PARTIAL_RAT_STALLS_MUL_SINGLE_UOP |
| 0.002 | ILD_STALL_LCP |
| 0.006 | io_Write_time |
| 0.011 | FP_ASSIST_ANY |
| ... | ... |
| 0.034 | FP_ASSIST_X87_INPUT |
| 0.034 | TLB_FLUSH_STLB_ANY |
| 0.037 | MEM_UOP_RETIRED_STORES_LOCK |
| 0.042 | MEM_UOP_RETIRED_LOADS_LOCK |
| 0.045 | BR_INST_EXEC_DIRECT_NEAR_CALL_TAKEN |
| ... | ... |
| 0.220 | net_Pack_recv |
| 0.717 | MEM_LOAD_UOPS_LLC_HIT_RETIRED_XSNP_HIT |

(b) Highest 10%.

Table 3: Features identified with the KS-test. Outliers are the measurements with highest/lowest 10% of power. OFactor(feature) = mean(outliers)/mean(regular data).

according to KS, the probability distribution of the outliers favors higher values. Compared to our typical measurement, more than 120 of all features have lower values than with regular execution, e.g., the number of FP operations is just 1/500 of the normal data. Clearly, the external power of $L$ is lower (about half of the maximum) as we have created $L$ by selecting measurements with the lowest power consumption.

Table 3b shows an excerpt of factors for the outliers with higher power consumption. The KS-test identifies 55 features with higher values and 33 features with lower values. The highest factors are the number of assists, i.e., transitions from AVX-256 instructions to legacy SSE, floating point 256 operations and L2 clean cache lines eviction events triggered by L2 prefetch. Lower values are, for example, I/O and FP assists.

#### *4.1.2. Discussion*

This example shows that it is possible to considerably reduce the number of features to inspect manually applying the KS-test. However, not all the features behave symmetrically. Therefore, high values from one set of outliers does not automatically lead to low values for the other set. For instance, the feature `RESOURCE_STALLS2_OOO_RSRC` is expressed in both sets of outliers. To illustrate the methodology using the KS-test and the OFactor, a few key statistics of this metric are listed in Table 4. In this case, the probability distributions between the outliers and the complementing set overlaps, e.g., the maximum value for the lowest 10% is higher than the 3rd quartile for the highest 90%.

Once we have identified this behavior, we should to identify the cause. In this simple example, the selection of benchmarks in our data set is causing the divergence of the features between lowest and highest energy consumption. For $L$, the main data points from at least one idle core are included when either `iperf` or `IOR` run. Also, $H$ includes runs with at least one `linpack` and `stream` runs on different CPUs. Therefore, we identify differences between these applications. In general, this methodology becomes more interesting when we pick outliers with more complex rules.

| Experiment | Min. | 1st. | 2nd | Mean | 3rd | Max. |
|---|---|---|---|---|---|---|
| Lowest 10% | 894 | 5892 | 12 K | 186 K | 14 K | 75 M |
| Highest 90% | 913 | 1 M | 7,4 M | 15 M | 22 M | 246 M |
| Highest 10% | 37K | 4,9 M | 18 M | 24 M | 35 M | 147 M |
| Lowest 90% | 894 | 19 K | 3.4 M | 13 M | 16 M | 246 M |

Table 4: Statistics of the `RESOURCE_STALLS2_OOO_RSRC` feature for the power outliers. The data sets with the 90% regular behavior are included for reference.

| Description | Idle | 1 CPU | 2 CPU | 3 CPU | 4 CPU |
|---|---|---|---|---|---|
| RAPL | 4.41 | 32.76 | 53.37 | 71.77 | 87.47 |
| Ext. power | 42.51 | 79.01 | 103.42 | 126.53 | 148.78 |
| Fitted | 42.38 | 78.28 | 104.36 | 127.66 | 147.54 |

Table 5: Measured power and linear fitting in Watts for idle and one to four active cores running `linpack`.

## *4.2. Correlation of RAPL counters to external power*

In this experiment, we first build a primitive model for predicting power based on the RAPL counter, and then assess the outliers of the model. Therewith, we are able to understand the characteristics of mispredictions.

### *4.2.1. Model outliers*

The model is built by using our measurements for the idle case and one to four active cores running `linpack`. Using linear regression to the median power for 0–4 active cores, we obtain the model $External\ power = 36.808 + RAPL \cdot 1.266$. As can be seen, the coefficient for RAPL is above 1. This is mainly because the external power measurements are always higher than those obtained by the RAPL counters, which only measure the power consumed by the processor sockets, but do not consider other hardware components such as motherboard, HDDs, NICs, nor power dissipated by the PSU itself. In our case, the 350 W PSU is certified with 80+ Gold level which guarantees 88%, 92% and 88% efficiency under 20%, 50% and 100% levels of load, respectively. Measured power consumption and fitted model results are given in Table 5.

Next, we compute the error of the model and build the outliers from 5% of the highest and 5% lowest error. The RAPL power and corresponding external power consumption is shown in Figure 1. The red values are considered to be outliers as their error to the accurate prediction is above 11.2 W (95% quantile) and below $-3.8$ W (5% quantile). We call these the upper and lower outliers, respectively. It can be observed also that the RAPL power varies considerably for a fixed external power and external power varies for a fixed RAPL power, but there is a high correlation between those counters and the model.

Note that now the outliers stem from our diverse set of benchmarks including `Quantum Espresso`. Inspecting the applications, it turns out that a large share of the outliers is formed by experiments of which one core runs `linpack` and/or `stream` together with `iperf`. This matches our expectations that external communication and memory operations cannot be explained well by the RAPL counters. However, the `IOR` benchmark is underrepresented in the set of outliers, and thus, it can be well predicted with the RAPL model.

### *4.2.2. Correlation to error*

One hypothesis is that a feature is directly correlated to the prediction error because it may not be taken into account by the internal algorithm for computing the RAPL model-based counters. Table 6 gives an excerpt of the features most correlated to the prediction error. As can be seen, there is only a weak correlation of some hardware counters to the error. Figure 2 shows the value of the counter `LOAD_HIT_PRE_HW_PF`[12] and its distance to the error. It can be observed also that the weak causal relation is hard to identify by humans. Similarly, the counter `CPU_CLOCK_UNHALTED_THREAD_P`[13] reveals that the more cycles are actually performed,

[12]Not SW-prefetch load dispatches that hit fill buffer allocated for H/W prefetch.
[13]Number of thread cycles while the thread is not in a halt state.

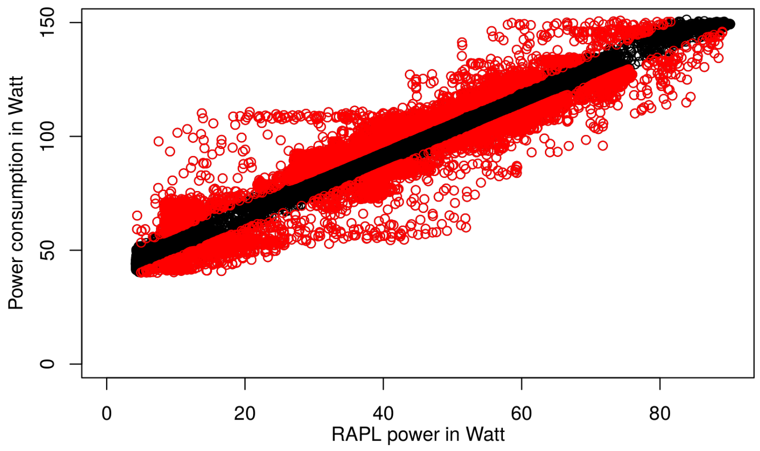


Figure 1: RAPL counters vs. External power.

| $p$ | Feature |
|---|---|
| 0.51 | LOAD_HIT_PRE_HW_PF |
| 0.51 | CPU_CLOCK_UNHALTED_THREAD_P |
| 0.46 | MEMLOAD_UOPS_RETIRED_HIT_LFB |
| 0.45 | IDQ_DSB_UOPS |
| 0.45 | RESOURCE_STALLS_ROB |
| 0.43 | LOAD_BLOCKS_ALL_BLOCK |
| 0.42 | L3_LAT_CACHE_MISS |
| 0.42 | OFFCORE_REQUESTS_OUTSTANDING_DEMAND_RFO |
| 0.41 | MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS |
| 0.40 | L2_RQSTS_ALL_DEM_AND_DATA_RD_HIT |
| 0.40 | LOAD_BLOCKS_STORE_FORWARD |
| 0.39 | UOPS_DISPATCHED_PORT_PORT_1 |
| 0.38 | INSTR_RETIRED_ANY |

Table 6: Correlation of features to the prediction error for the RAPL model.

the higher the potential error that can be obtained. Since there is a high correlation between this counter and power consumption, it is, indeed, not an interesting outcome.

### *4.2.3. Applying the KS-test*

We can also analyze the outliers by investigating one set in which we underestimate the power and one where we overestimate it, or by comparing both sets to a regular execution. Table 7(a) shows the results of the KS-test comparing outliers with the regular execution. The KS-test correctly identifies that network operations are much more likely to occur in the outlier set while for I/O operations is less likely. This fact has been known after inspecting the application mix. For example, it is also important to remark that `LOAD_BLOCKS_STORE_FORWARD`[14] is a bit higher than normal and counters responsible for stalls in multiply packed/scalar single precision operations allocated and for cache locks are lower.

Applying the KS-test to the data sets with the lower or upper outliers and comparing it with the remaining data shows many similarities but also describes some new characteristics of this data as listed in Table 7(b, c). For example, the number of instructions executed in a distant branch is higher for the lower outliers. In this case, the RAPL counter reports a larger fraction of the energy of the system than expected.

## *4.3. Analyzing individual applications*

As scientific applications usually execute multiple phases repeatedly, we can treat the application execution as a black box and perform a principal component analysis in order to better understand the properties and deviation of the application behavior. We illustrate this approach using the `linpack` benchmark running

[14]Number of loads blocked by overlapping with store buffer that cannot be forwarded.

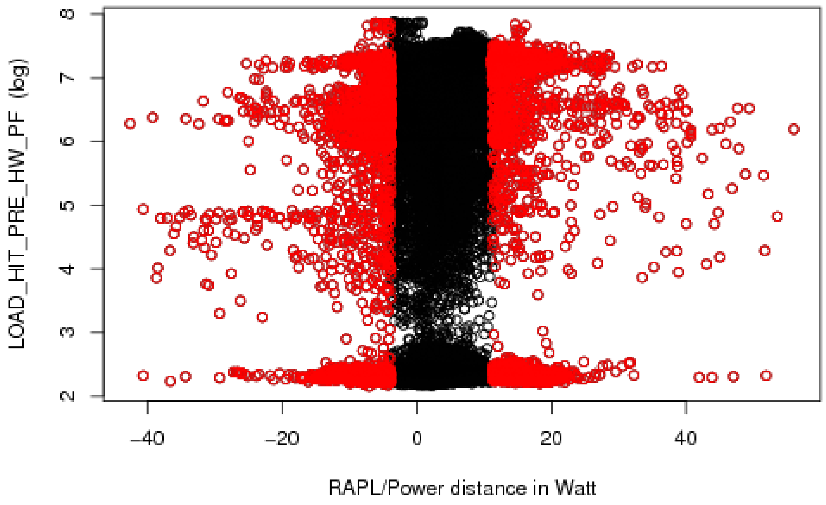
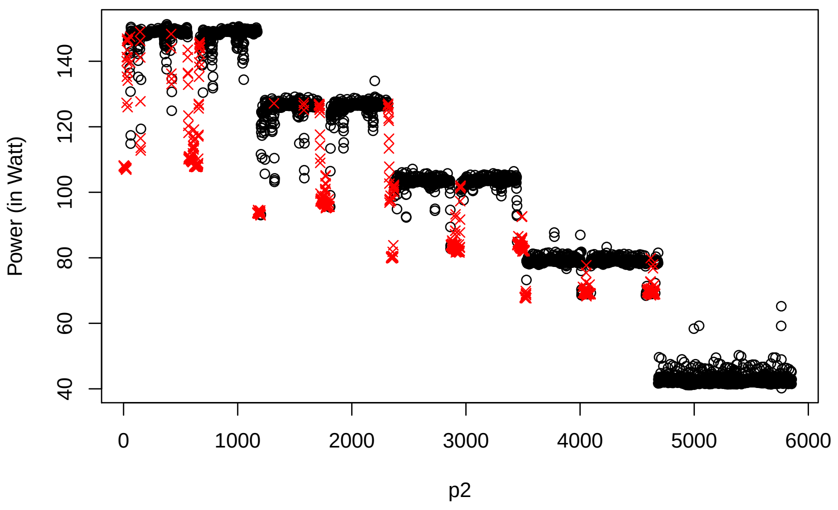


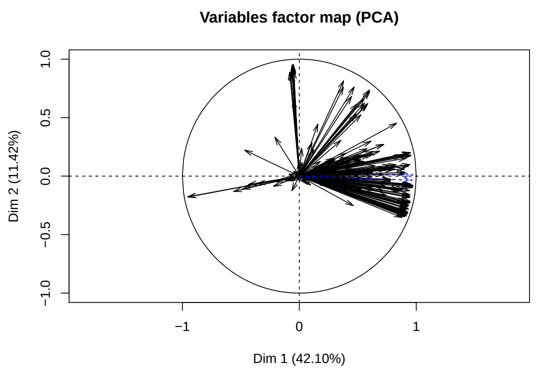
Figure 2: Prediction error vs. LOAD_HIT_PRE_HW_PF.

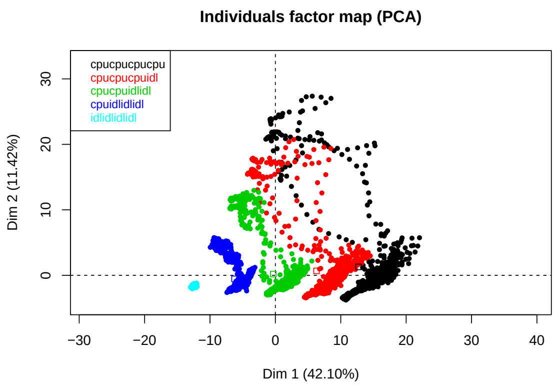
| OFactor | Feature |
|---|---|
| 14.10 | net_Bytes_recv |
| 8.83 | net_Bytes_sent |
| 1.82 | FP_ASSIST_X87_INPUT |
| 1.34 | LOAD_BLOCKS_STORE_FORWARD |
| 1.27 | L2_LINES_IN_I |
| 1.14 | MEMLOAD_UOPS_RETIRED_HIT_LFB |
| 1.14 | MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS |
| . . . | . . . |
| 0.66 | CACHE_LOCK_CYCLES_SPLIT_LOCK_UC_LOCK. . . |
| 0.21 | io_Read_time |
| 0.19 | io_Write_time |
| 0.15 | PARTIAL_RAT_STALLS_MUL_SINGLE_UOP |
| (a) All outliers | |
| 3.473 | FP_ASSIST_X87_INPUT |
| 2.684 | MEM_UOP_RETIRED_LOADS_LOCK |
| 2.606 | BR_INST_RETIRED_FAR_BRANCH |
| 0.730 | OTHER_ASSISTS_AVX_TO_SSE |
| (b) Upper outliers | |
| 1.897 | MEMLOAD_UOPS_RETIRED_HIT_LFB |
| 1.595 | MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS |
| 0.193 | FP_COMP_OPS_EXE_SSE_FP_SCALAR_SINGLE |
| 0.190 | DTLB_STORE_MISSES_STLB_HIT |
| 0.180 | CACHE_LOCK_CYCLES_SPLIT_. . . |
| 0.165 | FP_ASSIST_SIMD_OUTPUT |
| 0.102 | PARTIAL_RAT_STALLS_MUL_SINGLE_UOP |
| (c) Lower outliers | |

Table 7: Features reported by the KS-test comparing RAPL models outliers to typical cases.



(a) Timeline of external power.





(b) PCA, variables factor map.

(c) PCA, individuals factor map.

Figure 3: Analyzing `linpack` running on 4 to 0 cores. The outliers with a value $> 4$ in dimension 2 are marked in red in the timeline.

| Feature | Correlation |
|---|---|
| IDQ_MITE_UOPS | 0.955 |
| IDQ_MS_MITE_UOPS | 0.953 |
| ARITH_FPU_DIV_ACTIVE | 0.951 |
| ARITH_NUM_DIV | 0.951 |
| IDQ_MS_UOPS | 0.951 |
| INSTS_WRITTEN_TO_IQ_INSTS | 0.948 |
| ILD_STALL_IQ_FULL | 0.944 |
| FP_EXE_SSE_SCALAR_SINGLE | 0.937 |
| FP_EXE_SSE_SCALAR_DOUBLE | 0.937 |
| DSB_FILL_EXCEED_DSB_LINES | 0.912 |
| DSB2MITE_SWITCHES_PENALTY_CYCLES | 0.900 |
| FP_COMP_OPS_EXE_X87 | 0.889 |
| DSB2MITE_SWITCHES_COUNT | 0.883 |
| DSB_FILL_ALL_CANCEL | 0.876 |
| L2_RQSTS_RFO_HITS | 0.815 |

(a) Features with the highest correlation to dimension 2.

| OFactor | Feature |
|---|---|
| 0.025 | FP_256_PACKED_DOUBLE |
| 0.060 | OTHER_ASSISTS_AVX_TO_SSE |
| 0.230 | LOAD_HIT_PRE_SW_PF |
| 0.460 | BR_MISP_EXEC_COND_NON_TAKEN |
| 0.540 | BR_MISP_RETIRED_ALL_BRANCHES |
| 0.560 | BR_MISP_EXEC_COND_TAKEN |
| 21.80 | TLB_FLUSH_STLB_ANY |
| 49.50 | FP_COMP_OPS_EXE_SSE_FP_SCALAR_DOUBLE |
| 49.70 | IDQ_MS_MITE_UOPS |
| 50.00 | IDQ_MS_UOPS |
| 50.10 | ILD_STALL_IQ_FULL |
| 51.10 | DSB_FILL_EXCEED_DSB_LINES |
| 51.90 | ARITH_FPU_DIV_ACTIVE |
| 51.90 | ARITH_NUM_DIV |
| 207.80 | RESOURCE_STALLS_LB |

(b) Features identified with the KS-test on the PCA outliers of dimension 2.

Table 8: Analyzing `linpack` execution with PCA.

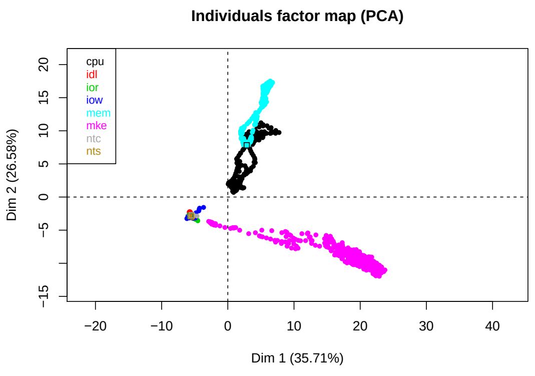
from 1 to 4 cores. Additionally, we include the idle measurements with the overall set for the analysis to identify only relevant fluctuations. The timeline of the measured external power is illustrated in Figure 3a, the figure concatenates the run with 4 cores down to 1 core and then the idle benchmark. Both black and red dots represent the measured power.

When applying the PCA to this data set, we obtain the variable factor map as shown in Figure 3b. This figure shows how the individual variables are aligned with respect to the new principal components, the bases in dimension 1 and 2. Each arrow represents one feature, and two features that have no correlation between them have a $90^\circ$ angle in the figure. There are several components such as power that are highly positively correlated to dimension 1. Several orthogonal aspects encompass dimension 2. Those two components alone describe 53% of the overall variance. When applying the transformation to each measurement, we obtain the projection onto dimension 1 and 2 according to Figure 3c. It can be observed that with each additional `linpack` core, the data points move right along dimension 1, which is also representing power. All executions show some diagonal cluster, however, there are data points aligned along dimension 2 that increase spread when running with more cores. Note that dimension 2 is in fact not correlated to power, the main contributing factors are listed in Table 8a. We can now investigate the data points that show a deviation along dimension 2. By selecting the data points with a value $> 4$ in dimension 2, we obtain the red points marked in Figure 3a. This represents that most of the points of a `linpack` phase utilize these features significantly more than the other data points. Regardless of the number of cores, these outliers can be easily detected.

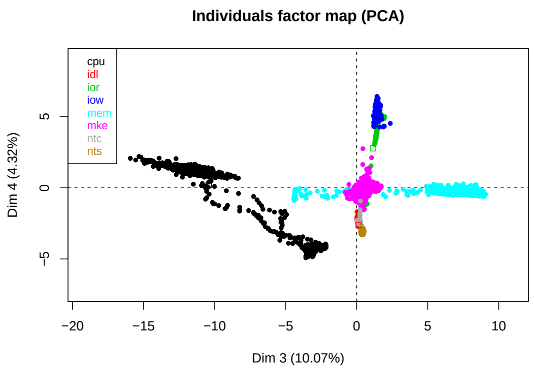
To further investigate the data, we apply the KS-test to these outliers. An excerpt of the result is shown in Table 8b. It can be seen that during the red marked phases, much less pre-fetched data is used but also less branch mispredictions occur. Indeed during these measurements, $50\times$ more scalar floating point operations but also divisions occur and $200\times$ resource stalls of the $\mu$op delivery due to load buffer (LB) overflow. By inspecting all hardware counters, we can try to identify the cause of the behavior: for example, the slow-down could be caused by the additional flushes of the second level TLB.

## *4.4. Investigating benchmark features*

A consideration that has been taken into account to build the application suite for creating the data was to utilize heterogeneous hardware features (CPU, memory, network and I/O). With the help of the PCA, the differences between the benchmarks can be analyzed. After applying it to a data set which contains the results of each application running on one core, the measurements are projected onto the first four dimensions according to Figure 4. We can see that according to dimensions 1 and 2 `IOR`, idle and `iperf` behave similarly. However, the application `make` is very diverse on dimension 1, while `linpack` and `stream` are mostly arranged in dimension 2 and have little overlap. Dimensions 3 and 4 separate `IOR` and `iperf`, still `iperf` has a lot in common with the idle benchmark. Note that those four dimensions together describe



(a) Dimensions 1 and 2.



(b) Dimensions 3 and 4.

Figure 4: PCA individual factor maps for our benchmarks.

already 75% of the overall benchmark variance. At best, an extremely diverse benchmark set would fill a large fraction of the graph, while currently only little space is occupied.

From this analysis, we can conclude that it is not justified to add the `iperf` benchmarks to our test suite. Even saturating our Gigabit Ethernet network is not sufficient to change the overall system behavior significantly. Also, the `IOR` benchmark brings little novelty and is a candidate for replacement. Clearly, the benefit of this analysis depends on the use case, for example, if we are training a linear model for power, we must avoid overfitting to a data set with typical cases and instead optimize the benchmark suite.

## *4.5. Building linear models*

In this section we investigate quantitatively the suitability of the 252 counters for building linear models and prediction power consumption. Equation 1 shows how a linear model predicts the power of an instance based on the values of $n$ features. In the equation, $f_i$ is the value of feature $i$ in the instance and the $c_i$ are coefficients fixed for the given linear model. The intercept $c$ is an estimate for a constant contribution to power.

$$\text{Power}(f_1, .., f_n) = c + \sum_{i=1}^{n} c_i \cdot f_i \tag{1}$$

Usually, the model is trained with a data set called training set and the model error is analyzed on a set called validation set. In our case, the standard training set are all benchmarks (1 to 5) and the standard validation set consists of the kernel `make` and `Quantum Espresso`. Additionally, the *trainNoIperf* set contains all training benchmarks without the two `iperf` benchmarks as they do not contribute much to the benchmark according to the PCA analysis and the set all contains every experiment conducted.

To assess the values, we consider the performance of a baseline model using the average power consumption as prediction achieves a mean error of 24.4 W on all data. Any linear model that is worse than this mean error is an indication that the chosen sets for training and validation exhibit different behavior.

To prepare the input data for training, we dropped the counters with fixed values for all observations. For the remaining 217 counters, we created linear models with one and two features and enumerated all possible variations of features. This means that for one feature 217 models are created while for two 46,872 models are possible. Additionally, for all relevant features which model achieve better mean error than our baseline model on the validation set, we computed the combinations with three features. This results in 240,464 models.

Table 9 shows the quartiles and mean error of all models for a given training and validation set for one and two features and Table 10 for the selected models with three features. The inherent model error of the models can be assessed when the model is build and validated on the same data set, e.g., "*all all* and *train*

*train*". As expected, in those two cases all models achieve a better accuracy than using a naive baseline model. When training the models with the training benchmarks and applying them to the validation set, about 50% of the models lead to a mean error of below 24 W and, thus, pass the expectation with our naive model. When using more features, the best models improve but also more bad models emerge.

The distribution of the average error across all models is shown in Figure 5 for one, two and three features. Even with one feature, a handful of models yield a prediction error below 10 W, among those is the RAPL counter and, interestingly, the CPU temperature. About 55 models have a model error of at least 100 W. The graph for models with two features is cropped after 20 W to focus on interesting models, many models include RAPL counters and sensors. The graph for models with three relevant features also shows that some linear combinations perform worse than 25 W although each of them individually achieved a better performance than 25 W. In general, if the benchmarks had covered the characteristics sufficiently, then all models would approximate the validation set better. Therefore, this is an indication that training and validation experiments stress some individual counters differently or that these counters are simply not relevant to power consumption as indicated by the PCA method. Note that, as also shown by the PCA, many counters are correlated and, therefore, their number could be reduced by choosing those that correlate better to the power consumption. Nevertheless, the goal of this section is to show that many variations achieve an appropriate prediction performance.

| Training | Validation | Min | Q1 | Median | Mean | Q3 | Max |
|---|---|---|---|---|---|---|---|
| *all* | *all* | 4.2 | 16.3 | 20.1 | 18.3 | 21.3 | 21.6 |
| *all* | *train* | 4.2 | 16.0 | 20.6 | 18.5 | 21.7 | 21.9 |
| *all* | *validation* | 3.9 | 14.6 | 17.9 | 17.3 | 20.4 | 25.2 |
| *all* | *allNoIperf* | 3.8 | 19.3 | 22.7 | 21.2 | 25.1 | 25.7 |
| *all* | *trainNoIperf* | 3.6 | 19.9 | 27.5 | 24.2 | 29.4 | 29.7 |
| *train* | *all* | 4.2 | 16.8 | 21.7 | 183.1 | 32.6 | 16070.0 |
| *train* | *train* | 3.7 | 15.1 | 18.7 | 17.5 | 21.7 | 22.0 |
| *train* | *validation* | 4.5 | 20.3 | 24.0 | 899.3 | 92.2 | 85560.0 |
| *train* | *allNoIperf* | 3.9 | 20.3 | 26.1 | 407.7 | 53.1 | 37600.0 |
| *train* | *trainNoIperf* | 3.4 | 17.2 | 23.8 | 22.4 | 29.2 | 30.4 |
| *trainNoIperf* | *all* | 4.3 | 17.8 | 21.8 | 210.9 | 41.9 | 15590.0 |
| *trainNoIperf* | *train* | 3.7 | 15.2 | 19.2 | 18.5 | 21.7 | 75.0 |
| *trainNoIperf* | *validation* | 5.0 | 19.6 | 28.4 | 1044.0 | 142.3 | 83000.0 |
| *trainNoIperf* | *allNoIperf* | 4.1 | 19.7 | 25.7 | 470.6 | 73.2 | 36480.0 |
| *trainNoIperf* | *trainNoIperf* | 3.3 | 16.1 | 22.8 | 21.5 | 28.5 | 29.5 |

(a) With one feature.

| Training | Validation | Min | Q1 | Median | Mean | Q3 | Max |
|---|---|---|---|---|---|---|---|
| *all* | *all* | 2.1 | 12.5 | 16.8 | 15.7 | 19.8 | 21.7 |
| *all* | *train* | 2.1 | 11.9 | 17.0 | 15.9 | 20.6 | 21.9 |
| *all* | *validation* | 2.3 | 12.4 | 14.7 | 15.0 | 17.7 | 28.2 |
| *all* | *allNoIperf* | 2.2 | 14.0 | 18.8 | 17.9 | 22.1 | 25.8 |
| *all* | *trainNoIperf* | 2.2 | 13.8 | 21.0 | 20.1 | 27.3 | 30.2 |
| *train* | *all* | 2.1 | 14.7 | 22.7 | 249.0 | 108.5 | 21110.0 |
| *train* | *train* | 2.0 | 10.1 | 15.0 | 14.3 | 18.3 | 22.0 |
| *train* | *validation* | 2.4 | 21.5 | 53.9 | 1265.0 | 507.1 | 112400.0 |
| *train* | *allNoIperf* | 2.4 | 18.6 | 33.0 | 565.6 | 233.6 | 49390.0 |
| *train* | *trainNoIperf* | 2.3 | 11.5 | 17.2 | 17.5 | 23.2 | 31.3 |
| *trainNoIperf* | *all* | 2.4 | 15.3 | 27.0 | 278.5 | 138.6 | 20370.0 |
| *trainNoIperf* | *train* | 2.3 | 10.3 | 15.3 | 15.6 | 19.8 | 99.6 |
| *trainNoIperf* | *validation* | 2.6 | 22.2 | 68.6 | 1416.0 | 658.8 | 108400.0 |
| *trainNoIperf* | *allNoIperf* | 2.3 | 18.7 | 39.7 | 631.4 | 299.4 | 47650.0 |
| *trainNoIperf* | *trainNoIperf* | 2.1 | 10.8 | 16.0 | 16.4 | 22.0 | 29.5 |

(b) With two features.

Table 9: Statistics for the mean prediction error of all linear models using the standard training and validation sets.

Table 11 shows the best 8 models that are not using RAPL counters. With more features, their quality increases significantly down to a mean error of 3.1 W which is about 3% of the observed power variation. We observe that about 4,787 combinations of two features achieve a better mean error than 20 W and 139,000 models with three features. Looking only at the best combinations with up to three features, it can be observed that external power can be modeled easily.

| Training | Validation | Min | Q1 | Median | Mean | Q3 | Max |
|---|---|---|---|---|---|---|---|
| *all* | *all* | 2.0 | 8.1 | 10.4 | 11.1 | 14.2 | 21.6 |
| *all* | *train* | 1.9 | 7.2 | 9.7 | 10.8 | 14.5 | 22.0 |
| *all* | *validation* | 1.9 | 10.5 | 12.7 | 12.5 | 14.8 | 26.0 |
| *all* | *allNoIperf* | 2.1 | 9.5 | 12.4 | 12.6 | 15.4 | 25.7 |
| *all* | *trainNoIperf* | 2.1 | 7.7 | 11.3 | 12.7 | 17.3 | 30.2 |
| *train* | *all* | 2.0 | 8.3 | 11.0 | 12.6 | 15.3 | 2886.0 |
| *train* | *train* | 1.9 | 5.7 | 8.5 | 9.9 | 13.9 | 21.8 |
| *train* | *validation* | 2.0 | 14.2 | 18.6 | 24.2 | 22.9 | 15300.0 |
| *train* | *allNoIperf* | 2.2 | 11.2 | 14.5 | 17.2 | 18.8 | 6735.0 |
| *train* | *trainNoIperf* | 2.2 | 6.3 | 9.9 | 11.7 | 16.1 | 30.4 |
| *trainNoIperf* | *all* | 2.2 | 8.4 | 11.4 | 14.8 | 16.9 | 3092.0 |
| *trainNoIperf* | *train* | 2.1 | 6.2 | 8.9 | 11.4 | 15.0 | 69.9 |
| *trainNoIperf* | *validation* | 2.3 | 13.8 | 18.2 | 29.4 | 24.1 | 16390.0 |
| *trainNoIperf* | *allNoIperf* | 2.2 | 10.2 | 13.9 | 18.8 | 18.7 | 7214.0 |
| *trainNoIperf* | *trainNoIperf* | 2.0 | 5.2 | 9.1 | 10.5 | 14.6 | 29.4 |

Table 10: Statistics for the mean prediction error of selected linear models with 3 features for our standard training and validation sets.

| `Features` | Mean error |
|---|---|
| `UOPS_DISPATCHED_PORT_PORT_0` | 10.11 |
| `UOPS_DISPATCHED_PORT_PORT_1` | 10.69 |
| `IDQ_DSB_UOPS` | 10.98 |
| `BR_INST_EXEC_COND_TAKEN` | 12.12 |
| `RESOURCE_STALLS_RS` | 12.77 |
| `L2_TRANS_RFO` | 12.86 |
| `L1D_BLOCKS_BANK_CONFLICT_CYCLES` | 12.99 |
| `INSTR_RETIRED_ANY` | 13.05 |
| `INST_RETIRED_PREC_DIST` | 13.42 |

(a) With 1 feature.

| Features | Mean error |
|---|---|
| `FP_256_PACKED_DOUBLE` + CPU utilization | 3.13 |
| `L2_LINES_OUT_PF_CLEAN` + CPU utilization | 3.83 |
| `OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD` + CPU utilization | 5.25 |
| `HW_INTERRUPTS_RECEIVED` + CPU utilization | 6.08 |
| `CPU_CLK_UNHALTED_CORE` + `L2_RQSTS_PF_MISS` | 6.11 |
| `CPU_CLK_UNHALTED_REF` + `L2_RQSTS_PF_MISS` | 6.31 |
| `OFFCORE_REQUESTS_ALL_DATA_RD` + CPU utilization | 6.57 |
| `INSTR_RETIRED_ANY` + `MEM_LOAD_UOPS_MISC_RETIRED_LLC_MISS` | 6.63 |

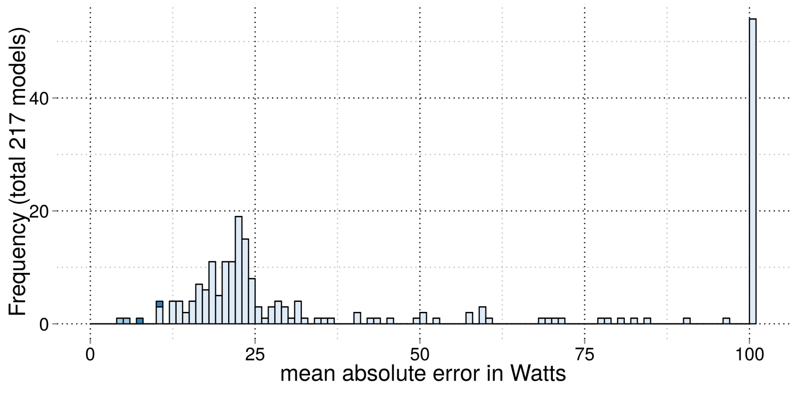
(b) With 2 features.

| Features | Mean error |
|---|---|
| `HW_INTERRUPTS_RECEIVED` + `LOAD_HIT_PRE_HW_PF` + CPU utilization | 3.12 |
| `HW_INTERRUPTS_RECEIVED` + `INSTS_WRITTEN_TO_IQ_INSTS` + CPU utilization | 3.61 |
| `HW_INTERRUPTS_RECEIVED` + `IDQ_MITE_UOPS` + CPU utilization | 3.93 |
| `DSB2MITE_SWITCHES_PENALTY_CYCLES` + `HW_INTERRUPTS_RECEIVED` + CPU utilization | 4.00 |
| `CPU_CLOCK_UNHALTED_THREAD_P` + `HW_INTERRUPTS_RECEIVED` + CPU utilization | 4.33 |
| `OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD` + `io_Read_count` + CPU utilization | 4.34 |
| `OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD` + `io_Read_bytes` + CPU utilization | 4.36 |
| `OFFCORE_REQUESTS_OUTSTANDING_ALL_DATA_RD` + `io_Read_time` + CPU utilization | 4.41 |

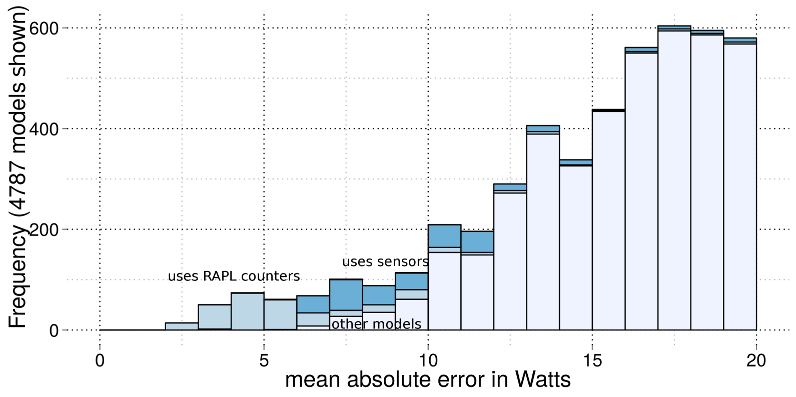
(c) With 3 features.

Table 11: Best models using the standard training and validation sets that are not using any sensor or RAPL counters.

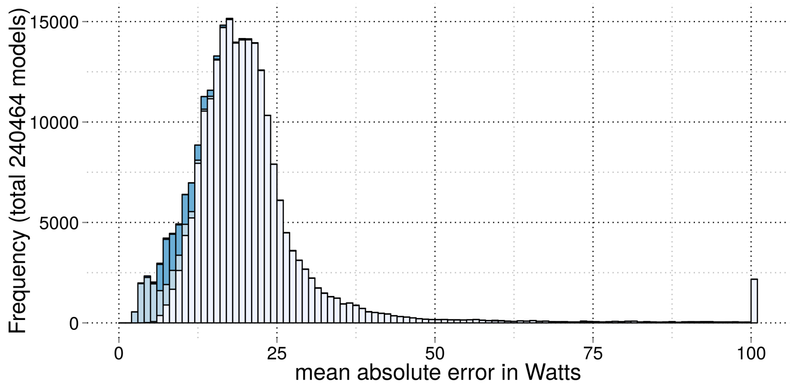
Finally, we analyze the influence of `iperf` to the created models and the differences to the validation results. By comparing the statistical results for the standard training set with the reduced training set without `iperf`, the typical model performs similarly (compare *train* and *trainNoIperf* in Table 9 and 10). The best models of the reduced training set with two and three features and their mean error are listed in Table 12. The best models on the validation set rely on different features compared to the standard training set. With the exception of the very best model, the 8 best models of the reduced training set perform better on our validation data. The reason is that the standard training set optimizes for all of the benchmarks equally and, as `iperf` behaves similarly to the idle benchmark, that behavior is overrepresented in the



(a) With one feature (values above 100 W are set to 100).



(b) With two features, X-axis is cropped after 20 W.



(c) With three features (based on combinations of good features).

Figure 5: Histogram for using the standard training and validation sets showing the average model error of all models with a fixed number of features. The stacked graph shows models using RAPL counters and sensors explicitly.

training set. This causes overfitting of the models while not bringing additional benefits. Therefore, the results from the PCA analysis, which revealed that this benchmark does not contribute significantly to the training set, are correct.

| Features | Mean error |
|---|---|
| OTHER_ASSISTS_AVX_TO_SSE + CPU utilization | 3.64 |
| RESOURCE_STALLS_ROB + CPU utilization | 4.09 |
| DSB2MITE_SWITCHES_COUNT + CPU utilization | 4.14 |
| CPU_CLOCK_UNHALTED_THREAD_P + CPU utilization | 4.39 |
| io_Read_time + CPU utilization | 4.87 |
| io_Read_count + CPU utilization | 4.96 |
| DSB_FILL_EXCEED_DSB_LINES + CPU utilization | 4.97 |
| CPU_CLK_UNHALTED_CORE + L2_RQSTS_PF_MISS | 4.99 |

(a) For two features.

| Features | Mean error |
|---|---|
| DSB2MITE_SWITCHES_COUNT + io_Read_count + CPU utilization | 3.44 |
| DSB2MITE_SWITCHES_COUNT + io_Read_bytes + CPU utilization | 3.46 |
| DSB2MITE_SWITCHES_COUNT + io_Read_time + CPU utilization | 3.47 |
| OTHER_ASSISTS_AVX_TO_SSE + io_Read_time + CPU utilization | 3.54 |
| FP_ASSIST_X87_INPUT + OTHER_ASSISTS_AVX_TO_SSE + CPU utilization | 3.64 |
| OTHER_ASSISTS_AVX_TO_SSE + io_Read_count + CPU utilization | 3.64 |
| OTHER_ASSISTS_AVX_TO_SSE + RESOURCE_STALLS_LB + CPU utilization | 3.65 |
| DSB_FILL_EXCEED_DSB_LINES + OTHER_ASSISTS_AVX_TO_SSE + CPU utilization | 3.68 |

(b) For 3 features.

Table 12: Best models using the NoIperf training set and not using any sensor or RAPL counters.

## 5. Summary and Conclusions

In this paper, we applied the statistical concepts of correlation, the Kolmogorov-Smirnov test and principal component analysis (PCA) to investigate hardware and software features. This allowed us to compare the inherent structure of two data sets used for training and validation. While these tools cannot identify the cause of abnormal behavior directly, they aid in reducing the number of features that must be checked by experts for significance during the study of the power consumption in general and development of models that mimic real power consumption. Applying these techniques to several test cases, we could identify many issues that are known by experts *a priori* and also some interesting behavior of the hardware counters. Therewith, these techniques can help to verify assumptions and reveal significant characteristics of selected outliers. Using PCA, we could automatically identify interesting measurements, as for the `linpack` benchmark, and assess the similarity of our benchmarks. We also demonstrate that a small combinations of some of hardware counters and resource utilization metrics can provide fairly good estimations of the power consumption. In the end, an on-line model computing total energy would avoid the use of complex and/or expensive wattmeters on large-scale platforms, and would provide feedback of energy consumptions to developers in order to reduce energy consumption. This, indeed, could be an advantage in the development of light power models, that need to be re-computed on-line in different power-aware software, such as green schedulers at operating system and facility scales.

As future work, we aim to build up general power models using the advanced set of statistical methods leveraged in this work. Furthermore, we plan to reduce the interval time and utilize intra-node power measurement to avoid fluctuations of the power supply unit. Also, we seek to improve our benchmark suite by adding further application kernels, and to extend our set of target platforms. Finally, we aim to automatically determine their (dis)similarities, with respect to hardware and software features.

## Acknowledgements

This work was supported by the EU Project FP7 318793 “EXA2GREEN” and the German Climate Computing Center (DKRZ).

## References

[1] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, and Others. The opportunities and challenges of exascale computing. Summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science, 2010.

[2] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, VECPAR'10, pages 1–25, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Philipp Gschwandtner, Michael Knobloch, Bernd Mohr, Dirk Pleiter, and Thomas Fahringer. Modeling CPU energy consumption of HPC applications on the IBM Power7. In *Parallel, Distributed and Network-Based Processing (PDP), 22nd Euromicro International Conference on*, pages 536–543. IEEE, 2014.

[4] M. Jarus, A. Oleksiak, T. Piontek, and J. Wglarz. Runtime power usage estimation of HPC servers for various classes of real-life applications. *Future Generation Computer Systems*, 36(0):299 – 310, 2014.

[5] Van Bui, Boyana Norris, Kevin Huck, Lois Curfman McInnes, Li Li, Oscar Hernandez, and Barbara Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Proceedings of the compFrame/HPC-GECO Workshop on Component Based High Performance*, CBHPC '08, pages 6:1–6:11, New York, NY, USA, 2008. ACM.

[6] Manuel F. Dolz, Francisco D. Igual, Thomas Ludwig, Luis Pi nuel, and Enrique S. Quintana-Ortí. Balancing task- and data-level parallelism to improve performance and energy consumption of matrix computations on the Intel Xeon Phi. *Computers & Electrical Engineering*, 2015.

[7] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the Conference on Power Aware Computing and Systems*, HotPower'08. USENIX Association, 2008.

[8] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, New York, NY, USA, 2010. ACM.

[9] Yu Xiao, R. Bhaumik, Zhirong Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A system-level model for runtime power estimation on mobile devices. In *Green Computing and Communications (GreenCom), IEEE/ACM Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, Dec 2010.

[10] Li Shang and N.K. Jha. High-level power modeling of CPLDS and FPGAS. In *Computer Design. International Conference on*, pages 46–51, 2001.

[11] John C McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C Snoeren, and Rajesh K Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf*, volume 20, 2011.

[12] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *SIGARCH Computer Architecture News*, 32(5):248–259, October 2004.

[13] Q. Wu, M. Martonosi, D.W. Clark, V.J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. Dynamic-compiler-driven control for microprocessor energy and performance. *Micro, IEEE*, 26(1):119–129, Jan 2006.

[14] Bruno Diniz, Dorgival Guedes, Wagner Meira, Jr., and Ricardo Bianchini. Limiting the power consumption of main memory. *SIGARCH Computer Architecture News*, 35(2):290–301, June 2007.

[15] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, pages 86–97, New York, NY, USA, 2003. ACM.

[16] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society.

[17] Matthew Curtis-Maury, James Dzierwa, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *Proceedings of the 20th Annual Intl. Conference on Supercomputing*, ICS '06, New York, NY, USA, 2006. ACM.

[18] Rong Ge, Xizhou Feng, and K.W. Cameron. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In *Supercomputing. Proceedings of the ACM/IEEE SC Conference*, Nov 2005.

[19] Ankush Varma, Brinda Ganesh, Mainak Sen, Suchismita Roy Choudhury, Lakshmi Srinivasan, and Bruce Jacob. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, New York, NY, USA, 2003. ACM.

[20] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM.

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

[22] R. Jotwani, S. Sundaram, S. Kosonocky, A. Schaefer, V. Andrade, G. Constant, A. Novak, and S. Naffziger. An x86-64 core implemented in 32nm SOI CMOS. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 106–107, Feb 2010.

[23] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen. IBM Power7 Systems. *IBM J. Res. Dev.*, 55(3):220–232, May 2011.

[24] NVIDIA. *NVML Reference Manual*, 2013.

[25] Ravi A Giri and Anand Vanchi. Increasing data center efficiency with server power measurements. *Intel Information Technology. IT@ Intel White Paper*, 2010.

[26] Ram Gnanadesikan. *Methods for statistical data analysis of multivariate observations*. John Wiley & Sons, 2011.

[27] A Onwuegbuzie, Larry Daniel, and N Leech. Pearson product-moment correlation coefficient. *Encyclopedia of measurement*

*and statistics*, pages 751–756, 2007.

[28] ZES ZIMMER Electronic Systems GmbH. LMG450 - Power Analyzer. `http://www.zes.com/en/Products/Precision-Power-Analyzer/LMG450`.

[29] M. Barreda, S. Barrachina, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, and E. S. Quintana-Ortí. 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.

[30] M. Barreda, S. Catalán, M. F. Dolz, R. Mayo, and E. S. Quintana-Ortí. Automatic detection of power bottlenecks in parallel scientific applications. *Computer Science - Research and Development*, 29(3-4):221–229, 2014.

[31] Jack Dongarra. The linpack benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing*, pages 456–474, London, UK, UK, 1988. Springer-Verlag.

[32] Ian T Young. Proof without prejudice: use of the kolmogorov-smirnov test for the analysis of histograms from flow systems and other sources. *Journal of Histochemistry & Cytochemistry*, 25(7):935–941, 1977.

[33] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.