Modular Energy Modeling using Energy/Utility

Marcus Hähnel
TU Dresden
Operating Systems Group
mhaehnel@tudos.org

Till Smejkal
TU Dresden
Operating Systems Group
tsmejkal@tudos.org

ABSTRACT

The modeling of the relationship between power usage and performance for complex computing systems is challenging due to the vast amount of tunable parameters that influence both metrics. To simplify the energy management of information systems from individual embedded machines to whole data centers we use a modular, hierarchical concept called Energy/Utility to model individual parts of a system. We present first results that show the decomposition of an individual asymmetric multi-processing system into hardware and software models. We show that using the Energy/Utility approach these models can stay manageable reducing total benchmark running time and modeling overhead while providing sufficiently high precision for performance and energy usage prediction.

KEYWORDS

Energy, Modeling, Energy/Utility

ACM Reference Format:
Marcus Hähnel and Till Smejkal. 2018. Modular Energy Modeling using Energy/Utility. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering Companion , April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3185768.3186311

1 INTRODUCTION

The energy consumption of devices or applications has become a major factor for the design and deployment of today's systems. For example, for mobile devices such as mobile phones, tablets or laptops, that rely heavily on their battery life, being more energy efficient has become an important differentiator. In data-centers, a trend to energy-based payment methods is emerging [16], giving customers a monetary incentive to deploy energy efficient applications and systems.

Accordingly, accurate knowledge about the energy usage of applications and the trade-offs between energy use and performance is vital. Consider for example a simple server system. Users usually have expectations regarding the performance of the system, such as a lower bound on the number of transactions or requests per second. But in addition, they want to save energy and thus may have a bound on the application's power usage or the energy spent to process a request. To schedule energy as a resource, the operating system needs to be aware of these trade-offs and the users' performance and power bounds so that it can configure the system in the most energy efficient way.

Several methods exist to monitor or estimate a system's energy usage. Typical examples are hardware measurement tools [14, 18], the RAPL power counters available with recent Intel® processors [10, 22, 28, 31], simulator-based energy computation [3, 27, 32], and model-based energy estimation during runtime [1, 29, 33]. While all of them provide their own set of properties and have their own use-case, we us a model-based approach in this work since it allows good estimation of the energy consumption at runtime without the need for specialized hardware.

Unfortunately, due to the increasing complexity of modern hardware, creating models that can accurately predict a device's energy consumption or an application's performance characteristics requires a lot of training effort because of the large amount of parameters influencing both, performance and energy use. Since an application's performance and energy characteristics are usually input dependent, the problem of exhaustively modeling complex applications for a diverse set of inputs quickly becomes intractable. To address this problem, we introduce a *modular energy modeling* approach based on the *Energy/Utility* concept [13]. By decoupling the hardware and software models of a system, we can significantly reduce the amount of benchmarking required for adequate model generation, while maintaining good accuracy. This enables us to quickly evaluate new configuration options of software and to easily adapt to changing workloads. In this paper, we will show first results that illustrate this modeling approach for a state-of-the-art embedded multi-core system.

The remainder of this paper is structured as follows: In Section 2 we introduce our modular energy modeling technique. Section 3 details the information about our modeling methodology as well as the verification setup. Results of our modular modeling approach are shown in Section 4 comparing them to a traditional modeling approach. In Section 5 we discuss techniques presented in related research and conclude our work in Section 6.

2 MODULAR ENERGY MODELING

The predominant approach to model the energy consumption and performance of a device, is to create one overall model that incorporates all of the device's properties and characteristics [4, 29]. Typically these models are based on hardware specific events such as performance counters [5, 8, 19, 25, 30], which can be easily monitored by the operating system and usually correlate well with the device's power consumption or an applications performance.

However, as user or operator of a system one goal should be to know the trade-off between energy savings and the corresponding reduction of the system performance — or *utility*. For example,

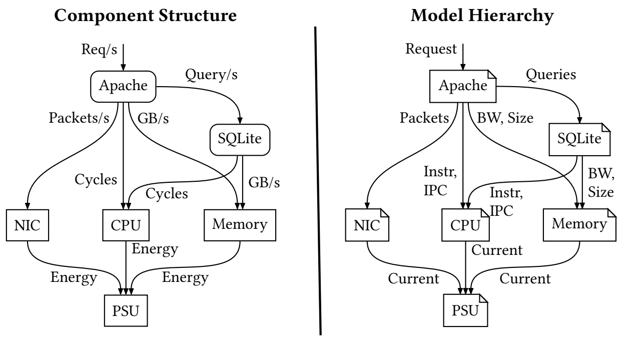

Figure 1: Overview of the modular modeling approach. Power estimation is based many small models, stacked together to resemble the device structure, model intermediate steps from the user measurable utility (Requests/s) to power

reducing the system power draw by 50 % may be acceptable when the throughput is only halved as well, but may be deemed to costly when just 25 % of the system performance can be reached. Making this trade-off visible requires a model that maps the achievable utility (e.g. the application performance) to the energy use of the system. Modeling both of these aspects in one overall model has several severe disadvantages:

- **Large configuration space.** The configuration space of a system which needs to be modeled and benchmarked is effectively a cross-product of all of its software *and* hardware configuration options. Modeling this space exhaustively quickly becomes intractable for a real system.
- **Limited adaptability.** If one component in the system changes (e.g. the CPU or a vital software part such as the network stack) all benchmarks have to be re-done, leading to large costs when the system is modified.
- **Workload dependence.** Generating accurate models requires that the calibration workloads are a representative of the production workloads. Once usage patterns change significantly, calibration of the model has to be re-run for the new workload types on all configuration settings.

To tackle these problems of power and performance modeling, we utilize the modular Energy/Utility approach, that was first described by Härtig et al. [13]. The general idea of this technique is outlined in Figure 1: Instead of building only one model that combines all the characteristics of the device and works for all possible applications, Energy/Utility uses multiple smaller models that describe different components of the system. These models are combined in a hierarchy and composed by the operating system to estimate the system's power consumption and performance.

With the system configuration shown in Figure 1, six models that interact with each other are used to estimate the power consumption and performance of the Apache application. Each of the models describes a component in the system that provides a service to either the user or another component — for example, the CPU that can execute instructions, the memory subsystem that can perform memory accesses or the SQL database that can be queried for information. To provide their service, every component (and hence the corresponding model) may in turn require services implemented by other components. How the models interact is defined by the dependencies between the system components. For instance, Apache provides the service of handling website requests. To provide this service, Apache uses a database for internal management state as well as other system components such as the CPU, the memory, and the network interface. Accordingly, the model that is used to describe Apache interacts with the CPU model, the memory model, the NIC model and the SQL database model. To be able to combine the individual models, they have to decide on a common interface. As shown in Figure 1, the input interface for the NIC model is *bandwidth*. Hence, since Apache uses the network interface, the model that describes Apache has to have a function that maps Apache's input interface *requests* to network bandwidth.

Each individual model in the hierarchy has a configuration space that describes how the operating system can manage the device or configure the software. These are for example, different frequencies and core counts for the CPU, speed setting for the NIC or the number of workers in the Apache server. These configurations are internal to the model and not visible for other models in the hierarchy. Since all the models in the hierarchy do not depend on the internal configuration options of other models, they can be calibrated independently — or even in parallel by different parties. Only the interface with which the models can be combined has to be well defined beforehand.

When using the modular Energy/Utility approach, moving an application to new hardware or changing a single component does not require the re-calibration of the whole model hierarchy. Only a new hierarchy needs to be put together consisting of the models of all new parts. Similarly, if a model of an application needs to be re-calibrated due to workload changes (e.g. a different type of database queries) this re-calibration can be performed independent of the configuration space of the hardware (e.g. frequencies). The software must just be benchmarked with its own set of configuration options since all the other configuration options are already handled by the other models in the hierarchy. This significantly reduces the cost of training the models. The same applies to hardware changes.

Energy/Utility even allows to re-calibrate a software model on the fly during production, as no expensive — and possibly invasive — calibration benchmarks are required. The operating system only needs to be able observe the application's performance as well as its interaction with other components. Both can be achieved during run-time through software monitoring interfaces and hardware performance counters.

3 EXPERIMENTAL SETUP & METHODOLOGY

In this paper we show that it is possible to provide a modular model [1] for a reasonably complex system that

- provides accurate energy and performance data compared to a monolithic model and
- can adapt to new configuration options and workloads

[1]We will make measurement data, the generated models and the scripts to generate the models available on GitHub upon publication

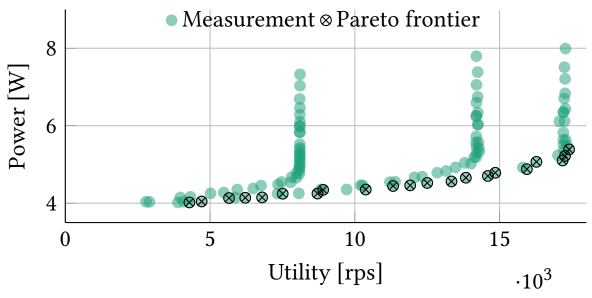

Figure 2: Directly measured profile

3.1 Experimental Setup

For our experiments we use the ARM big.LITTLE system Odroid XU3 from Hardkernel. We choose this system because its out-of-order architecture is complex enough to feature most of the modeling obstacles of a full blown x86 system, while also featuring integrated energy monitoring sensors. This allows easy and fast generation of the individual component models.

The CPU consists of two core clusters with four cores each. The *LITTLE* cluster features less powerful in-order ARM A9 cores while the *big* cluster has full-fledged out-of-order ARM A15 cores. We concentrate on the *big* cluster as it presents a more interesting modeling challenge and is closer to traditional servers because of its out-of-order architecture. In addition, Hähnel et al. [12] already used a similar system for energy modeling in the past. The system has 2 GB of DRAM and the big cluster features clock frequency scaling from 200 MHz to 2 GHz in 100 MHz steps. Furthermore, the system is equipped with a gigabit Ethernet USB adapter.

We measure the whole system energy consumption using an external power meter — an Odroid SmartPower power meter. The energy consumption of the individual system components such as the memory, the GPU, the little cluster and the big cluster are gathered using the built-in INA-231 energy sensors.

The application that we model is a MEMCACHED server. MEMCACHED is an in-memory key-value store, typically used as a cache for web applications. We use a default configured *memaslap* benchmark as workload and run the server with configurations for 1 and 2 threads and memory cache sizes of 32, 64 and 256 MB.

3.2 Decomposition

Figure 2 illustrates the monolithic energy/utility profile of MEMCACHED when measured for all configurations directly. Each dot (●) in the graph represents one configuration combination, the x axis denotes performance in requests per second, the y axis is the total system power. We also highlight the Pareto frontier (⊗) in the profile as this subset of configurations are the ones of interest for resource scheduling. In the following, we decompose this measurement and create separate hardware and software models for the system. All models are generated using polynomial regression.

Hardware Model. In this paper we focus on the CPU and memory models. To allow proper decomposition into the hardware and the software model, the CPU and memory model need to have an interface that can be used by software without knowledge of the CPU internals. As already shown by previous work [1, 12, 29], the power usage of the CPU is not only dependent on its configuration (e.g. frequency, core count, idle mode) but also on which functional parts of the CPU are used. The dominating factor for this system is the memory-intensiveness of a workload. Accordingly, we define the interface between the software and the hardware model to be the number of required instructions and the memory intensiveness of the instruction stream as memory instructions per cycle (IPC).

Based on this interface we define the dynamic part of the CPU power model as:

$$IPC = f(frequency, n_{cores}, memoryIntensiveness)$$
$$P_{ARM} = f(frequency, n_{cores}, IPC)$$

The adjusted R^2 for both model parts is 0.98 indicating an excellent fit of the models. For the memory we identified

$$P_{MEM} = f(frequency, memoryIntensiveness)$$

to be the best representation with an adjusted R^2 of 0.98. All models are second degree polynomials.

Software Model. The software model for MEMCACHED provides all the necessary data to map the cost of executing a request to the resource usage from memory and CPU. The model is defined as:

$$IPR = f(size_{memcache}, threads_{memcache})$$
$$memoryHeaviness = f(size_{memcache}, IPR)$$

The models' adjusted R^2 of 0.97 and 0.96 indicate a good fit. All models are second degree polynomials. The IPR metric (*instructions per request*) is in addition to the software configuration also dependent on the workload composition. The model was done for the composition of the benchmark, but due to the modularity of the Energy/Utility approach can easily adapt to changing workloads.

3.3 Monolithic Model

For comparison we use a traditional monolithic modeling approach that tries to capture the system behavior in one single model. Accordingly, we combine our hardware and software configuration settings as model parameters and define the model as:

$$P = f(frequency, n_{cores}, size_{memcache})$$
$$rps = f(frequency, n_{cores}, size_{memcache})$$

We again use a second degree polynomial for all components but the frequency component in the power model. We find that a better fit is achieved by a third degree polynomial for the frequency. The adjusted R^2 for these two models is 0.98 and 0.92 respectively.

4 RESULTS

The model evaluation in Table 1 shows the mean absolute percentage error and geometric mean relative absolute error (GMRAE) for the two modeling approaches for all training configurations. While the monolithic approach is quite good at prediction power it has significant errors in the performance predictions, despite the good fit indicated by the high adjusted R^2. We attribute this to the difficulty in modeling the interactions between individual hardware components which, especially in respect to performance, are not always additive in their effects and thus hard to capture using polynomial models with additive model parameters.

	Power		Performance	
	MAPE	GMRAE	MAPE	GMRAE
monolithic	1.71 %	1.08 %	22.13 %	5.99 %
modular	2.63 %	1.73 %	4.08 %	2.40 %

Table 1: Mean absolute percentage error (MAPE) and geometric mean relative absolute error in percent (GMRAE) for power and performance for the two models

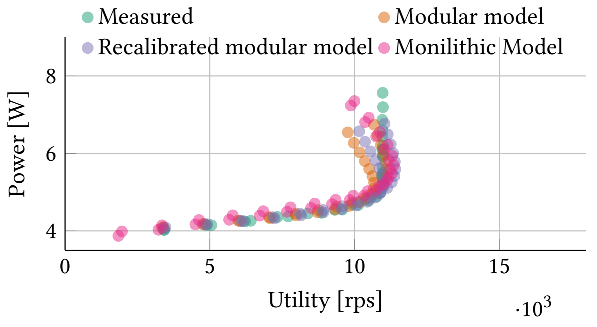

Figure 3: Additional config measured, modeled and monolithically modeled

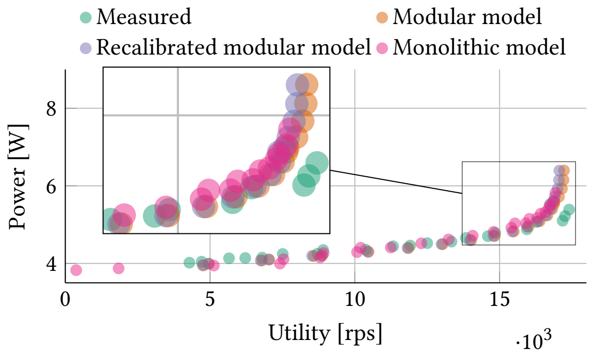

Figure 4: Modeled vs. real Pareto frontier

4.1 Adaptability

We introduce a new additional software configuration that changes the load characteristics of the application by adding the option to have a 128 MB MEMCACHED cache size.

Figure 3 shows how well the different models can estimate the performance and power of this new configuration at the different hardware configuration points. We also add a recalibrated software model, as this easily is possible using our modular approach by observing the changed CPU usage patterns of the application at this new cache size. Please note how the monolithic model severely underestimates the achievable power and performance at the lower end of the configuration spectrum. This is especially sever as these points will become part of the Pareto frontier.

To further illustrate this point we also plot the Pareto frontiers obtained by the different modeling approaches and the measured values in Figure 4. Again, notice how at the low performance spectrum the monolithic model significantly deviates from the measured values. For the rest of the values both modeling approaches capture the Pareto frontier satisfactorily.

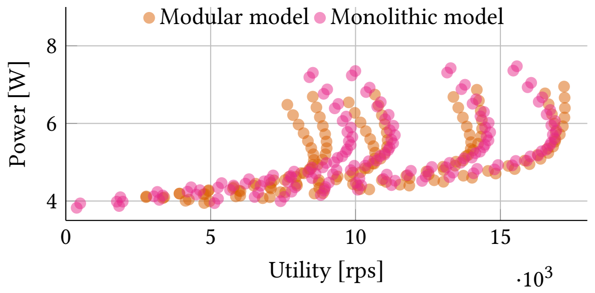

Figure 5: Model profiles (modular and monolithic)

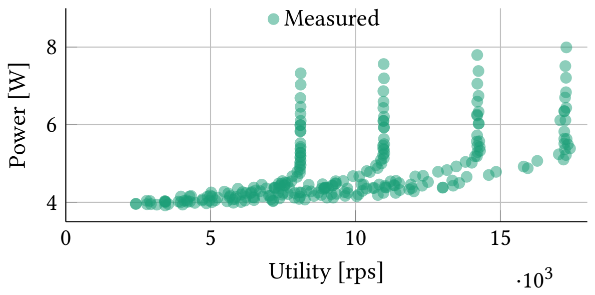

Figure 6: Measured profile with new software configuration

Figure 5 shows how the two modeling approaches capture the behavior of the real hardware for the performance and power tradeoffs. We measured all the additional configurations and plot the complete profile for all four MEMCACHED cache sizes in Figure 6 as a baseline for Figure 5. Especially in the area where increasing the frequency is of no additional benefit for performance (the four vertical lines formed by the dots) the monolithic model fails to capture the effect of the memory boundedness adequately. The two tailed nature of the memory-bounded area in both models is not completely clear. We assume that this is still an inefficiency in our model, where the decomposition does not capture all system aspects. However, we note that the monolithic model suffers from the same problem, albeit to a lesser degree. Instead it overestimates the performance impact increased frequency has, resulting in decreased performance at higher CPU clock speeds that is not visible in the real system.

Table 2 shows the same metrics for the model estimation errors as found in Table 1, but including the additional configuration. We show MAPE and GMRAE for the monolithic model, the original modular model and the re-trained modular model. Retraining is only required for the software model. The hardware models remain unchanged, enabling the immediate re-training upon the observation of the new load pattern.

	Power		Performance	
	MAPE	GMRAE	MAPE	GMRAE
monolithic	1.68 %	1.11 %	20.83	% 6.24 %
modular	2.52 %	1.68 %	4.13 %	2.56 %
modular recalibrated	2.45 %	1.58 %	3.83 %	2.58 %

Table 2: MAPE and GMRAE for power and performance for the models with an additional software configuration

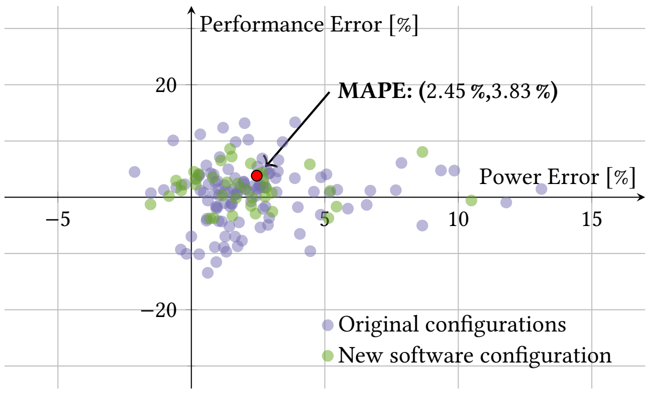

Figure 7: Relative performance and power error for recalibrated modular model

4.2 Model accuracy

Finally we want to show the prediction error for the two metrics that are predicted for each configuration. In Figure 7 we plot the relative percentage error for the power model on the x-axis and the relative percentage error for performance model on the y-axis. We use the re-calibrated model in this graph. The green dots (●) mark the configuration that were added later and caused a change in the software model. The blue dots (●) mark the original configurations. The red dot (●) illustrates where the MAPE lies in this graph. We note that the performance model usually stays within an error margin of 10 % while the performance model tends to over-estimate the performance but is also well below 10 % in most cases.

Figure 8 shows the same statistics for the monolithic model. Again green dots (●) illustrate the additional configurations while pink dots (●) show the original training set. Note that, while the model is vastly better at predicting power usage, it has a significant spread for the errors in the performance prediction, with errors ranging as high as 30 %. For the additional data, predictions are mostly wrong in significantly over-estimating the performance of a given configuration.

5 RELATED WORK

Modeling the energy consumption of computer systems at various different levels has been a widely discussed research topic for many years. In the literature, there exist various examples that aim to model individual parts of the system such as the processor [1, 6, 15, 17, 30], the memory subsystem [6, 30] or the networking interface [6, 11]. Other approaches do not concentrate on

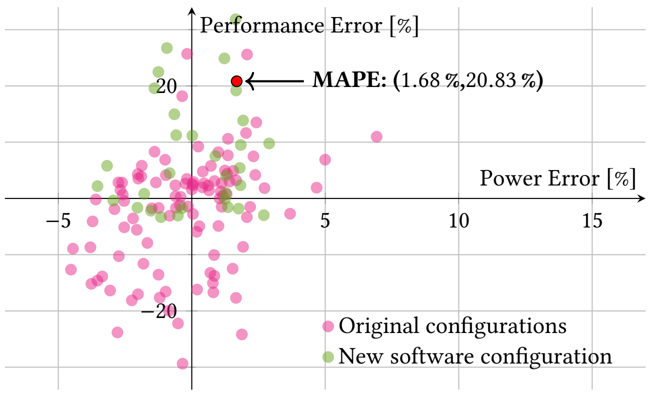

Figure 8: Relative performance and power error for monolithic model (without recalibration)

individual components but predict the overall system power using models [4, 9, 20, 25]. Besides calculating the energy consumption of a device or its components, researchers also use power models to determine the power draw of applications [24, 25, 29] as well as of virtual machines [2, 5, 7]. Our work examines a hierarchical modeling approach based on the Energy/Utility idea to tackle the typical problems of power/energy and performance modeling. We propose to model different parts of the system independently and combine them according to the device configuration, thereby allowing power and performance estimations at different system levels as well as with different device configurations.

Based on the different platform characteristics and model requirements, researches have used different events as basis for their power models. A very wide spread approach is to use performance counters combined with special platform knowledge to build power and performance models [1, 4, 17, 25, 29, 30]. Such a technique allows to create fine grained models with a small prediction error in the range of 5 % to 10 %. However, to be able to generate such performance counter based models, the performance counters have to be identified that best correlate with the to-be-modeled characteristics. Finding these counters either necessitate internal knowledge of the target platform [7] or a very time consuming exhaustive search through the configuration space [29]. Unfortunately, the accuracy of performance counter based power models comes at a cost. Ever time the model is deployed on a new device, the search for the model basis (the best correlating performance counters) have to be rerun to achieve the best possible result.

To make their approaches less platform dependent and thereby more portable to other devices, Pathak et al. [23, 24] as well as Zheng et al. [33] use system-call traces as a basis for their models instead of performance counters. Shye et al. [26] use a very similar technique to reduce the platform dependency of their models. They extract the power consumption of the device from the component usage (e.g. disk accesses, CPU utilization, or network usage) as reported by the operating system. In our work, we are also able to create platform independent models while still preserving a high prediction accuracy. Our hierarchical modeling approach allows changing models at the various hierarchy levels following changing device configurations. Hence, depending on the device's processor, a different processor power model can be plugged in the modeling hierarchy and thereby be used by the higher level models. This technique allows adaptation of the overall model to a new hardware platform without the need to completely retrain everything. A similar approach is also described by Malkamäki et al. [21], although they only concentrate on lower level system components and their power consumption and not on higher layers such as the application layer like our solution.

6 CONCLUSION

We have shown that a modular energy modeling approach is feasible for a complex out-of-order architecture. We modeled CPU, memory and an application separately and combined the models to generate an Energy/Utility profile for the system showing the trade-offs between power and performance. Our approach has very good prediction accuracy for power and performance and can compete with a traditional monolithic modeling approach. Individual models using the modular approach can even be simpler than the monolithic models as they better model the real relationships between components, their energy use and performance. We envision, motivated by these early results and the composability of our models, to scale this Energy/Utility approach from individual machines over racks up to whole data-centers.

ACKNOWLEDGMENTS

This work is supported by the German Research Foundation (DFG) within the CRC 912 — HAEC.

REFERENCES

[1] Frank Bellosa. 2000. 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*. ACM, 37–42.

[2] Ramon Bertran, Yolanda Becerra, David Carrera, Vicenç Beltran, Marc Gonzalez, Xavier Martorell, Jordi Torres, and Eduard Ayguade. 2010. Accurate energy accounting for shared virtualized environments using pmc-based power modeling techniques. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*. IEEE, 1–8.

[3] Mario Bielert, Florina Ciorba, Kim Feldhoff, Thomas Ilsche, and Wolfgang Nagel. 2015. HAEC-SIM: A Simulation Framework for Highly Adaptive Energy-Efficient Computing Platforms. *EAI Endorsed Transactions on Energy Web* 16, 8 (8 2015). https://doi.org/10.4108/eai.24-8-2015.2261105

[4] William Lloyd Bircher and Lizy K John. 2012. Complete system power estimation using processor performance events. *IEEE Trans. Comput.* 61, 4 (2012), 563–577.

[5] Ata E Husain Bohra and Vipin Chaudhary. 2010. VMeter: Power modelling for virtualized clouds. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. Ieee, 1–8.

[6] Aaron Carroll and Gernot Heiser. 2010. An analysis of power consumption in a smartphone. (2010).

[7] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. 2015. Process-level power estimation in vm-based systems. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 14.

[8] Thanh Do, Suhib Rawshdeh, and Weisong Shi. 2009. ptop: A process-level power profiling tool. (2009).

[9] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. 2006. Full-system power analysis and modeling for server environments. International Symposium on Computer Architecture-IEEE.

[10] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring energy consumption for short code paths using RAPL. *ACM SIGMETRICS Performance Evaluation Review* 40, 3 (2012), 13–17.

[11] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2013. eBond: energy saving in heterogeneous RAIN. In *Proceedings of the fourth international conference on Future energy systems*. ACM, 193–202.

[12] Marcus Hähnel and Hermann Härtig. 2014. Heterogeneity by the Numbers: A Study of the ODROID XU+E big.LITTLE Platform. In *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/hotpower14/workshop-program/presentation/hahnel

[13] Hermann Hartig, Marcus Volp, and Marcus Hahnel. 2013. The case for practical multi-resource and multi-level scheduling based on energy/utility. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*. IEEE, 175–182.

[14] Timo Hönig, Heiko Janker, Christopher Eibel, Oliver Mihelic, and Rüdiger Kapitza. 2014. Proactive Energy-Aware Programming with PEEK. In *2014 Conference on Timely Results in Operating Systems (TRIOS 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/trios14/technical-sessions/presentation/hoenig

[15] Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 93.

[16] Victor Jimenez, Francisco Cazorla, Roberto Gioiosa, Eren Kursun, Canturk Isci, Alper Buyuktosunoglu, Pradip Bose, and Mateo Valero. 2011. Energy-aware accounting and billing in large-scale computing facilities. *IEEE Micro* 31, 3 (2011), 60–71.

[17] Russ Joseph and Margaret Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM, 135–140.

[18] Vasilios Konstantakos, Alexander Chatzigeorgiou, Spiridon Nikolaidis, and Theodore Laopoulos. 2008. Energy consumption estimation in embedded systems. *IEEE Transactions on instrumentation and measurement* 57, 4 (2008), 797–804.

[19] Tao Li and Lizy Kurian John. 2003. Run-time modeling and estimation of operating system power consumption. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 31. ACM, 160–171.

[20] Yepang Liu, Chang Xu, and Shing-Chi Cheung. 2013. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*. IEEE, 2–10.

[21] Tuomo Malkamäki and Seppo J Ovaska. 2016. Modeling power flow in computer and server systems. In *Proceedings of the 2nd International Workshop on Energy-Aware Simulation*. ACM, 2.

[22] Heike McCraw, James Ralph, Anthony Danalis, and Jack Dongarra. 2014. Power monitoring with PAPI for extreme scale architectures and dataflow-based programming models. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 385–391.

[23] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. 2012. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 29–42.

[24] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*. ACM, 153–168.

[25] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. 2013. Power containers: An OS facility for fine-grained power and energy management on multicore servers. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 65–76.

[26] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. 2009. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 168–178.

[27] Tajana Simunic, Luca Benini, and Giovanni De Micheli. 1999. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference, 1999. Proceedings. 36th*. IEEE, 867–872.

[28] T Smejkal, M Hähnel, T Ilsche, M Roitzsch, WE Nagel, and H Härtig. 2017. E-Team: Practical Energy Accounting for Multi-Core Systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association.

[29] David C Snowdon, Etienne Le Sueur, Stefan M Petters, and Gernot Heiser. 2009. Koala: A platform for OS-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 289–302.

[30] David C Snowdon, Stefan M Petters, and Gernot Heiser. 2007. Accurate on-line prediction of processor and memoryenergy usage under voltage scaling. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*. ACM, 84–93.

[31] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*. IEEE, 207–216.

[32] Narayanan Vijaykrishnan, Mahmut Kandemir, Mary Jane Irwin, Hyun Suk Kim, and Wu Ye. 2000. Energy-driven integrated hardware-software optimizations using SimplePower. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 95–106.

[33] Heng Zeng, Carla S Ellis, Alvin R Lebeck, and Amin Vahdat. 2002. ECOSystem: Managing energy as a first class operating system resource. In *ACM Sigplan Notices*, Vol. 37. ACM, 123–132.