2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

Performance Events Based Full System Estimation on Application Power Consumption

Shu Yang*, Zhongzhi Luan*, Binyang Li*, Ge Zhang†, Tianming Huang* and Depei Qian*†
*Sino-German Joint Software Institute,
Beihang University, China
†School of Computer Science and Engineering,
Xi'an Jiaotong University, China
yangshu729,zhongzhi.luan,binyang@buaa.edu.cn
zdove1993@stu.xjtu.edu.cn
htmingbuaa,depeiq@buaa.edu.cn

***Abstract*—The value of run-time application power consumption in the full system is a prerequisite for energy optimization. Though past studies have demonstrated the use of performance events for building single component (CPU, Memory, Disk) power model, we propose a method to estimate application power consumption in the full system with performance events. In this paper, we identify different stress on components from CPU-intensive, memory-intensive and I/O-intensive applications. Combining these and the "trickle-down" effect of performance events, the feasibility of estimating application power consumption with a few performance events is demonstrated. Through measurement of actual systems running test sets, these models are shown to have an average error of less than six percent across the considered workloads. Besides, we demonstrated that these models can be applied to real-world applications. Moreover, power models proposed in this paper can provide software developers or system designers with visible power behaviors for the applications. Thus, software developers can make decisions or change code to develop more energy-efficient software.**

***Keywords*-performance events;application;power models;full system;**

I. Introduction

As power and energy are becoming one of the key challenges in the system and software designs, several researchers have focused on the energy efficiency of hardware and embedded systems [1], [2], the role of application software in IT energy consumption still needs investigation. The value of run-time application power consumption in the full system is a prerequisite for the research of software energy optimization. In this paper, we propose a method to estimate application power consumption in the full system with performance events, which directly links performance and energy.

Past studies about system power model focus on the cycle-level power simulation, and estimating component (CPU, memory, and Disk) power. These work does not apply to the estimated application energy consumption in the full system. Though work of building full-system power model based on the utilization of the main hardware components has been developed, these methods are coarse-grained and not accurate enough, but we propose a method to estimate the power consumption in the full system based on the performance events. The principle behind using performance events is that total system power consumption is the sum of individual hardware unit power consumption, and the performance counters can accurately reflect the use of the corresponding hardware unit, in turn, reflect its power consumption. Our work is facing two challenges: (1) The consideration of the simultaneous stress on the multiple components from running applications. (2) Since processors do not provide enough counters to monitor multiple performance events at the same time [3], it is necessary to find a cheap way to build full-system power model with only a few performance events.

In this paper, we identify different stress on components suffer from different types of applications. Combining these and the "trickle-down" effect of performance events [4], we simplify our problem by building different power models for different classes of applications. Then, we propose a methodology to predict the power consumption for applications in full system that follows a two-step approach: First, we use micro-benchmarks that individually stress the corresponding component of the system utilized by the classified applications and do the corresponding analysis. The first step provides us with alternative performance events which are strongly related. Second, the final selection of performance events is determined by the average error rate during regression analysis and the compared power traces between measured and modeled. Once the performance events determined, the power model also determined. Moreover, the full-system power model provide the first step in helping us refine our understanding of the power from the standpoint of performance events, which can profile the application power behaviors in the execution.

The remainder of this paper is organized as follows: Section II compares and contrasts the related work. Section III describes our methodology. Section IV describes two

978-1-5090-4297-5/16 $31.00 © 2016 IEEE
DOI 10.1109/HPCC-SmartCity-DSS.2016.154

power models and the validations. Section V presents how power model using performance events can be used for the research of the software-based optimization in energy consumption. Section VI discusses conclusion and directions for future work.

II. Related Work

In this section we discuss some of the prior research that is related to our work. Hardware measurement is expensive and cannot provide application power behavior information. Novel hardware designs such as LEAP [5] have been designed for embedded system, which have attracted most attentions. Architectural-Level power simulators have already been developed to estimate energy consumption, Brooks [6] presents Wattch, an architectural-level power simulator that estimates CPU power consumption, and discusses how Wattch can be used to perform microarchitectural tradeoff studies, compiler optimization, and hardware optimizations for low-power. Using simulators can be highly inefficient for estimating runtime application power (Wattch only performs 80k instructions per second) and his simulator is limit to performing architectural or compiler energy research on processor.

Prior work about full-system modeling has also been developed [7], [8]. Economous [7] proposed full-system power model based on the system components utilization metrics. First, the maximum power of main components in the system are measured at full load. Then, the real-time component power is determined by the product of the utilization and maximum power. At last, a linear model with respect to full-system power consumption and the utilizations of the major components were developed. One of the disadvantages is that the prediction granularity of the model was limited by the speed at which the utilization metrics would update, and another is that it is not strictly linear relationship between CPU power and CPU utilization. In our experiments, for different applications of the same hardware and software systems, CPU utilizations for all applications are 100% while the CPU power consumption values can differ by more than 20%. Other work focuses on estimating OS power dissipation at routine or system-call level. Li [9] presented a strong correlation between OS power consumption and the instructions per cycle (IPC) during OS routine executions. The accuracy of this method is limited for the only consideration of IPC.

Processor performance events are increasingly being used to estimate and analyze energy. Bellosa [10] is the first to show strong correlation between performance events and specific energy consumption in Pentium II, but this correlation is not quantified. Bricher [4], [11] notes the "trickle-down" effect of processor performance events and proposed five component-level power models. Prior work using performance counters is limited to estimating the component-level power model, but we extend this work to the estimated energy of the full system, which is more challenging.

Table I
Description of Target System

System	*Description*
CPU	2.93GHz Intel Core i3
Memory	4GB DDR -1333
Storage	Seagate Barracuda 7200.12
Network	1000Mb/s Ethernet
Microarchitecture	Westmere

In the field of energy, the development of hardware power optimization is faster than software power optimization [12]. Existing studies neglect the facts that software developers play an important role in the software energy optimization process. Power models proposed in this paper can provide software developers or system designers with visible power behaviors for the applications with performance events. With it, software developers can make decisions or change code to develop more energy-efficient software.

III. Methodology

In this section, we describe our measurement environment, the different stress on components from the different types of applications, the principle behind the selections of workloads and performance events.

A. Hardware Power Measurement

In this work, we obtain the overall power consumption simply by connecting a power meter Voltech PM1000+ between the system and the AC outlet. The synchronization of the hardware power measurement and sampling of performance events is guaranteed. The power points collected from the hardware measurement are used to train and validate our model. Table I presents our target system environment.

B. Stress on Components from Applications

In this paper, we classify applications into three types: CPU-intensive applications, memory-intensive applications and I/O-intensive applications. Different applications stress components of a different degree. CPU-intensive applications use a lot of CPU when they are actively working to accomplish task. As a result, it contributes the increased CPU power in the full system. Memory-intensive applications utilize both CPU and memory and are speed limited by how fast the memory can feed data to the processor. We use an array reference synthetic program to analysis characteristics of different types of applications. In the experiment, we set the array reference step at a very large number to avoid the locality and then exponentially increase the size of array. The results of the experiment show two points: (1) I/O-intensive applications have high memory and disk stress while low CPU stress. CPU-intensive and memory-intensive applications have no I/O utilization. (2) With the increased size of array, the application from a CPU-intensive

application into a memory-intensive application, then into an I/O-intensive application. The types of applications are relevant with the stress of components.

It is noted in [4], last level cache misses are possible to approximate the number of memory accesses. But it does not apply to estimating the energy consumption in memory caused by I/O-intensive applications, because in fact there are few cache misses for the direct I/O transactions between memory and disk. Due to the DMA (Direct Memory Access), processors are aware of I/O transactions only at the beginning and end of the transmission. Thus it is necessary to distinguish the origin of the stress. Based on the fact that processor is unaware of I/O transactions, in this paper, we build different power models to estimate I/O-intensive applications and non I/O-intensive (CPU-intensive, memory-intensive) applications power consumption in full system.

C. Workloads selection

The selection of workloads is a key issue in training the power model. Our selection of workloads was driven by two factors: 1) a diverse set of behaviors across all workloads; 2) the sufficient variation in power. The first requirement can be met by a variety of benchmarks, but power variations of these benchmarks may be small. Our method first uses micro-benchmarks that are elaborately designed to sufficiently stress the target components. The use of micro-benchmarks to stress specific components like a control variate method, and goal is to derive the correlations between performance events and system power, excluding other confounding factors. But the final power model is quantified by the training set consist of synthetic benchmarks and industry-standard benchmarks, and the model is required to be validated by the test sets.

D. Performance Events Selection

During an application execution, we capture performance events with PAPI, a specification of a cross-platform interface to hardware performance counters on microprocessors [13]. Because performance events counted by the hardware performance counters cannot represent all kinds of events, performance events monitored in the system level are considered as a complement in this paper.

There are hundreds of performance events can be monitored in the system. A quick way is to select events that are most representative of the utilized components stressed by applications. Then we use micro-benchmarks to derive the correlation between the full-system power and the events selected in the previous step, only the strongly related metrics are selected as the inputs of linear and polynomial regression. Due to the similarity of many performance events such as *Instruction Retired* and *Instruction Executed*, the final selection of performance events is determined by the average error rate during regression analysis and the compared power traces between measured and modeled, and the process is repeated with alternate performance events as inputs. Once the final performance events determined, the power model also determined.

Table II
CPU 2006 CLUSTERING

Benchmark	*Cluster*	*Name*
SPEC CINT 2006	1	**gobmk**,hmmer,h264
	2	**omnetpp**
	3	**mcf**
	4	**libquantum**
	5	**astart**
	6	**xalancbmk**
SPEC CFP 2006	1	**lesilie3d**,bwaves,wrf,milc
	2	**sphinx3**,calculix,zeusmp,gromacs,namd
	3	**GemsFDTD**
	4	**cactusADM**
	5	**dealII**,games,tonto
	6	**soplex**
	7	**lbm**
	8	**povray**

IV. POWER MODEL

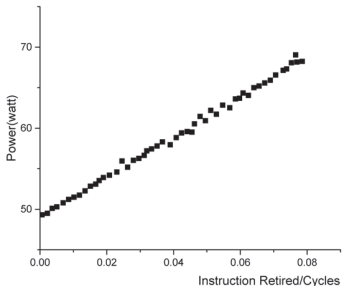
A. Full-system Power Model for CPU-intensive and Memory-intensive Applications

Both CPU-intensive applications and memory-intensive applications stress CPU and memory in the full system, and there is no explicit definition to distinguish them. According to the experiments, we found that a model can be used to estimate the power consumption of both types of applications. Thus, it is not necessary to classify the target application to CPU-intensive or memory-intensive, as long as there is no I/O transactions in these applications.

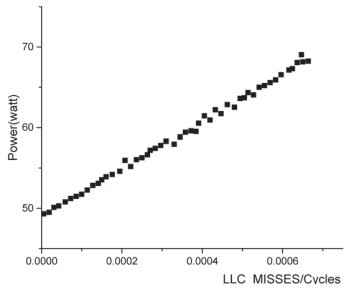
1) Workloads Selection: Since synthetic benchmark cannot represent various behaviors of applications, it is necessary to consider industry-standard benchmarks as a complement. We use a clustering approach demonstrated by Phansalkar [14] to select proper workloads in SPEC 2006 benchmarks [15]. The principle behind the selection are 1) A benchmark suit should have programs that are representative of a wide range of applications areas. 2) Avoid selecting too many similar programs. Redundancy of workloads will not only increase evaluation time without providing additional information but also badly impact the model. Phanasalkar first measured fundamental program characteristics related to their instruction locality, data locality, branch predictability, and instruction mix, using performance counters. In order to remove the correlation between these variables, this dataset is pre-processed using Principal Components Analysis (PCA) [16]. Clustering Analysis (CA) is then used to group programs with similar program characteristics. At last, Phanasalkar shows that SPEC 2006 benchmarks are balanced and a subset of 6 integer programs and 8 floating pint programs can yield most of the information. Our method is to use one member of cluster for model validation and the remaining for model training, if a cluster only contains one

Table III
LIST OF REPRESENTATIVE INPUT SETS FOR SPEC CPU2006 PROGRAMS

Benchmark	*Name*	*InputSetNo.*
SPEC CINT 2006	perlbench	1
	bzip2	4
	gcc	1
	gobmk	5
	hmmer	1
	h264	2
	astar	2
SPEC CFP 2006	games	3
	soplex	1



(a) System power vs. Instruction Retired



(b) System power vs. LLC_Misses

Figure 1. The correlation between system power and performance events.

member, then the workload is used for both training and validating. Table II lists workloads we select for training in normal text and validating in bold. Additionally, a synthetic benchmark and Stream (a memory-intensive benchmark) [17] are respectively added to the training set and test set. Many benchmarks in the SPEC 2006 have multiple input sets, for example, *403.gcc* has nine input sets, in order to further reduce the redundancy we only use the benchmark's input set selected as the most representative input set shown in [14]. We list the most representative input set in Table III. It shows that the fourth input set is the most representative dataset in *401.bzip2*. The training process based on the cluster of SPEC 2006 reduces training time without loss of accuracy.

2) Performance Events Selection: Since we are building power model used to estimate CPU-intensive and memory-intensive applications power, CPU and memory related events are considered. The initial selection is based on the "trickle-down" effect of events, and the product of *Core Frequency* and *Time* is used to create per cycle metrics. Figure 1 demonstrates the correlation between some of the performance events and full-system power, and we can see a nearly linear increase of the system power with the increased number of the given performance events. These events, but are not limited to these events have linear relationship with system power. Only highly corresponding performance events are set as the inputs of the linear and polynomial regression. the final set of selection in our model contains only contain processor active cycles, instruction retired and last level cache misses, which are explained in the next.

Active Cycles — *Cycles in which processor are active*. This metric reflect the active cycles of processor. A significant increase in power can be observed when the state of processor changes from idle to active and the significant increase can be reflected by this performance metric, while the power in idle state maintained as a constant.

Instruction Retired — *The instruction (micro-operation) leaves the* "Retirement Unit". The retirement unit writes the results of speculatively executed micro-operations into the user-visible registers and continuously checks the status of micro-operations in the reorder buffer, looking for ones that have been executed and no longer have any dependencies with other micro-operations in the instruction pool. This performance metric reflects how many instructions were really executed.

LLC_Misses — *Count each cache miss condition for references to the last level cache.* Since the number of memory accesses is directly proportional to the number of last level of cache misses, it is possible to approximate the number of memory accesses using this metric for non I/O-intensive applications.

3) Results: Within our training set, the form of the model is given as follows:

$$P_{system} = 47.675 + 23.834 * ActiveCycles + 2.093 * InstructionRetired + 72.113 * LLC\_Misses \quad (1)$$

The variables in the equation are normalized as the per cycle metrics.

We use the test set demonstrated in the section IV-A1 to validate our model and average errors are calculated by the equation 2, which takes errors in every sampling point into consideration. Figure 2 demonstrates three benchmarks' temporal power. It shows that our model can accurately reflect power variation in *sphinx3*, *calculix*, and *astar*. A high responsiveness is validated. The errors between measured and modeled power are below 5%. Table IV presents summaries of average errors for this model applied to ten workloads. The full system power model averaged below 6% error across all workloads, and the average errors among *gobmk*, *soplex*, and *povrary* are below 1%. Additionally,

Table IV
AVERAGE ERROR FOR THE FIRST POWER MODEL

	$Gobmk$	$omnetpp$	mcf	$astar$
AverageError(%)	0.41	2.02	2.76	2.63
	$leslie3d$	$sphinx3$	$GemsFDTD$	$soplex$
AverageError(%)	3.37	1.22	5.05	0.47
	$povray$	$Stream$		
AverageError(%)	0.70	2.97		

Stream, a memory-intensive benchmark not originates from SPEC 2006, has an average error below 3%. Since the simplicity, high accuracy and responsiveness of the model, it provides a valuable tool to estimate system power running both CPU-intensive and memory-intensive applications.

$$AverageError = \frac{\sum_{i=1}^{NumSamples} \frac{|Modeled_i - Measured_i|}{Measured_i}}{NumSamples} * 100\%. \quad (2)$$

B. Full-System Power Model for I/O-intensive Applications

In the section, we build a full system power model for I/O-intensive applications. Then the model is validated by two benchmarks: IOzone [18] and Stream (we turn the memory-intensive benchmark into an I/O-intensive benchmark through changing the size of array).

1) Workloads Selection: Our training set contains data set generated by synthetic disk workload and half of the data set from IOzone, a filesystem benchmark that generates the highest sustained power in the memory and disk. Additionally, we reset the size of array in Stream and then turn it into an I/O-intensive benchmark. At last, the remaining data set from the IOzone and all of the data from Stream260 (the array is tuned to have 260M elements) are used to validate our model.

2) Performance Events Selection: Transactions between CPU and memory can be observed by processor performance counter. However, DMA accesses that originated from an I/O device whose destination is system main memory allows the independence of the CPU. I/O transactions are only visible to the processor at the beginning and end of the transmission. Since performance events originate from the PMC (performance monitoring counts) can't represent all transactions in the system. We can consider system-level utilization as a complement for performance events.

In this section, we use a synthetic disk workload to derive the correlation between I/O utilization and full-system power. A strong quadratic relationship was observed in the experiment.

3) Results: Since the synthetic benchmark does not represent a diversity of behaviors, the final power model is quantified in the training set explained in section IV-B1. The model is presented as follows:

$$P_{system} = 52.514 - 7.432 * (U_{io})^2 + 20.769 * U_{io}. \quad (3)$$

Table V
AVERAGE ERRORS FOR THE SECOND POWER MODEL

	$Measured(Watt)$	$Modeled(Watt)$	$Error(\%)$
$IOzone$	63.01	64.57	5.54
$Stream260$	61.86	65.42	5.99

and a trace of the total measured and modeled power is given in Figure 3. In the case of our test set, the metric of I/O utilization is used to achieve an average error about 5.5% presented in Table V. A model based only on the I/O utilization is a cheap method as it is a well understood metric. As it is used to estimate I/O-intensive application power, the inaccuracies are decreased.

C. Real-world applications

Prior work in section IV has proved our models are useful for applications which are classified into three types, namely CPU-intensive, Memory-intensive, and I/O-intensive. In this part, we apply our models to real-world applications, which are based on the machine learning algorithms mentioned next.

Supprot vector machines(SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

K-means clustering is a method of vector quantization, k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

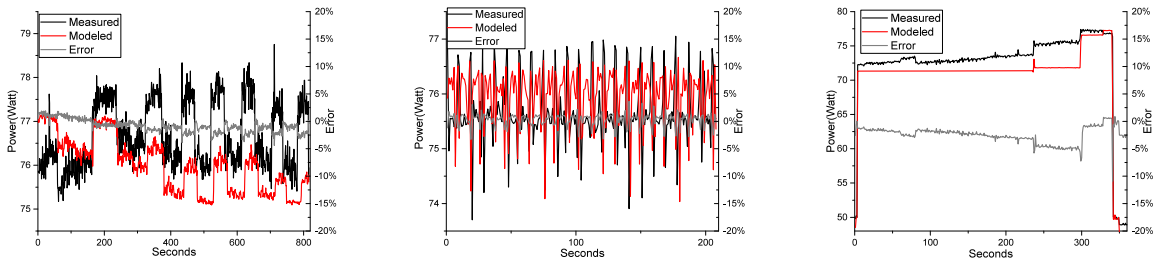
Logistic regression or *logit regression*, or *logit model* is a regression model where the dependent variable (DV) is categorical.

Elastic Net is a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge methods.

K-nearest neighbors is a non-parametric method used for classification and regression in pattern recognition.

We applied our experiments on Madelon and Arcene data sets. The Madelon data set, 4400 instances and 500 attributes, is an artificial data set, which was part of the NIPS 2003 feature selection challenge. This is a two-class classification problem with continuous input variables. The Arcene data set task is to distinguish cancer versus normal patterns from mass-spectrometric data. This dataset is one of 5 datasets of the NIPS 2003 feature selection challenge. Besides, we conducted our experiments on various machine learning libraries exposed in Python: MLPy, PyBrain, PyMVPA, MDP, Shogun and MiLK.

When we firstly run these machine learning applications, we find there are no I/O transactions, which can be easily observed by system-level monitoring. So we apply the first model to estimating power consumption. Similar to our prior method, we demonstrate the temporal power graph and compare the average error. Figure 4 respectively describe svm, kmeans, logistic and elasticnet power variation, the



(a) Power model—sphinx3. (b) Power model—calculix. (c) Power model—astar.

Figure 2. Power traces for CPU2006 benchmark

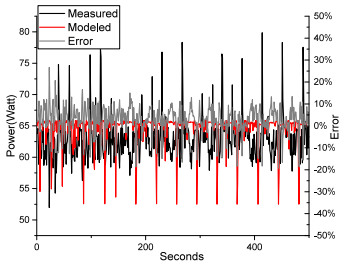


Figure 3. Power traces for IOzone.

Table VI
AVERAGE ERROR FOR THE MACHINE LEARNING APPLICATIONS

	SVM	$Kmeans$	$Logistc$
AverageError(%)	2.44	3.02	4.41
	$ElasticNet$	KNN	
AverageError(%)	3.76	1.13	

curve fitting responses to the measured curve. Besides, table VI demonstrates that average errors are about one percent in the best case, and still below five percent in the worst case.

The experiments in this part demonstrate that our models are not limit to classified applications. According to the application runtime information such as I/O transactions, which can be monitored by operation system, we can decide which model equation should be applied. Because our models are based on performance events and every application is running on the hardware components (such as CPU and memory) in the final analysis, models proposed in this paper are independent of applications scenarios.

V. THE APPLICATIONS OF SYSTEM POWER MODEL USING PERFORMANCE EVENTS

In this section, we show how our full-system power model using performance events can be used for the research of the software-based optimization in energy consumption.

First, our model can provide applications real-time performance and power information through the following three metrics, which can be calculated by the performance events used in the model.

IPC — Instruction per cycle, which describes the performance of the target applications.

Energy-Delay — The energy-delay product, proposed by Gonzalez and Horowitz [19], combines performance and power consumption into a single metric in which lower values are considered better both from energy and performance standpoint.

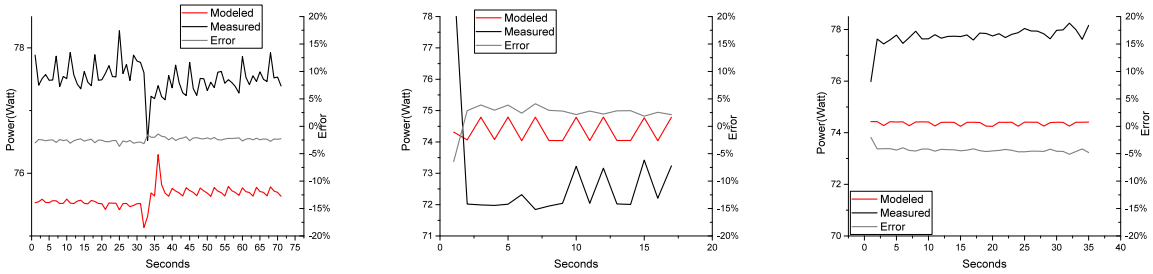
Energy-Efficiency — A generic metric defined as equation 4 in [20] to evaluate the energy efficiency, depending on the useful work completed and energy consumed.

$$EnergyEfficiency = \frac{UsefulWorkDone}{UsedEnergy}. \quad (4)$$

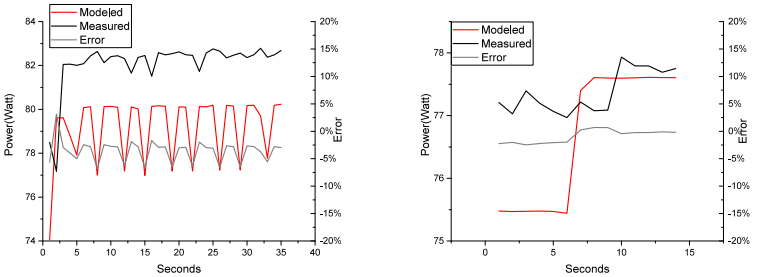
Second, our full-system power model can present application run-time power behavior in a visual method. With this tool, software engineers can realize the impacts of program behavior on energy. Thus, it supports to help software engineers make decisions or change their code to pursue the tradeoff between performance and power.

Since last level cache misses have an effect on the full-system power model which can be seen in equation 1, we consider a case study which examines the effects of the cache misses on full-system power consumption. We developed our experiments on the well-known program matrix multiplication. The experiments were based on the three functionally equivalent versions *kji*, *ijk* and *ikj* [21] (the order of the cycle is identified by the index of *i, j, k*) of the matrix multiplication. The three versions of matrix multiplication have a gradually decreased cache misses while completed the same task. It is noted that the version of *ikj* has more cache-friendly code than *ijk*, and *ijk* is more cache-friendly than *kji*.

Figure 5 shows the *performance* (instructions per cycle), *breakdown of average power*, *energy-delay*, and *energy efficiency* for the different versions of matrix multiplication. The IPC graph shows that the drop of cache misses has a great benefit for performance. However, it does not apply to the power. Due to the decreased cache misses, the

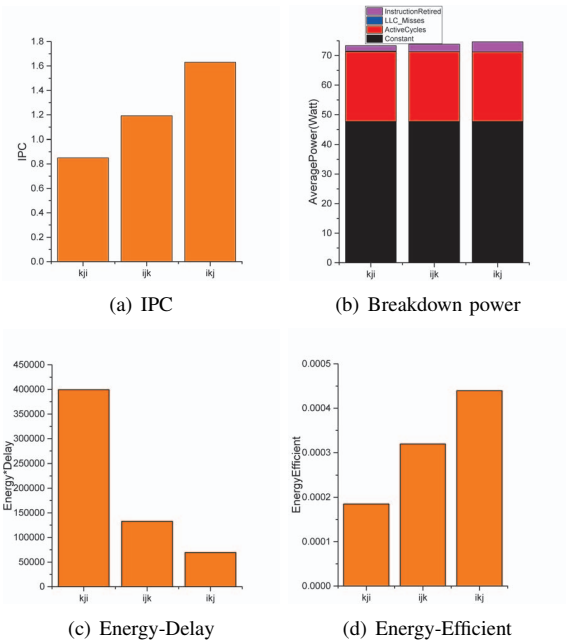


(a) Power traces — SVM. (b) Power traces — Kmeans. (c) Power traces — Logistic.



(d) Power traces — Elasticnet. (e) Power traces — KNN.

Figure 4. Power traces for machine learning applications



(a) IPC (b) Breakdown power

(c) Energy-Delay (d) Energy-Efficient

Figure 5. The correlation between system power and performance events.

efficiency of processor is improved. As it can be seen in subfigure(b), the value of *Instruction Retired* and *Active Cycles* are increased. Since they have greater impact on power than cache misses,which are decreased, the power in fact increased. The comparison of performance and power shows the tradeoff between them. In general, we sacrifice high power in order to obtain better performance, or pursue lower power suffer from the loss of performance in the power limit environment. The overall energy consumption of a program is equal to power dissipation multiplied by the execution. In this experiment, the version of the least cache misses *(ikj)* has the minimal energy, which can be easily understood by reducing processor idle energy consumption in completing the same task. For subfigure(c), energy-delay decreases monotonically with the decreased cache misses, reflecting the fact that the reduction of unnecessary cache misses offers benefit for the matrix multiplication program from both performance and energy standpoint. As the three versions have completed the same useful work, the energy efficiency is inversely proportional to energy. Subfigure(d) shows the *ikj* has the best energy efficiency as we expect. Overall, the point of this case study is to demonstrate how the system power model, the resulting metrics, graphs shown can help explore tradeoff points taking into account both power and performance related metrics. Moreover, with the visible breakdown of power, programmers can make better decision or change their code to obtain low power or better performance, which is imperative in the power-constrained server environments. The full-system power model using performance events provide the first step in helping us refine our understanding of the power and energy from the standpoint of performance events, which can profile the

application power behaviors in the execution.

VI. Conclusion

In this paper, the feasibility of predicting application power consumption in full system with a few performance events is demonstrated. Since the performance events used in our models are available for most platforms and can be monitored simultaneously, our models have a strong practicality. Across the considered benchmarks, the full-system power models can accurately reflect temporal power and are shown to have an average error of less than 6%. Moreover, These models can be applied to the real-world applications. Last but not the least, the full-system power model using performance events helps software developers understand how the performance events affect the energy consumption of their applications and motivate them to develop energy-efficient software.

Our research is subject to some limitations that can be addressed in future work. First, the current category to decide which model equation should be applied for the target applications is based on the I/O transactions. Further work will come up with a classifier, trained with the data set collected by emerging real-world applications (big data, ML, VR). Thus, a model will be auto determined to be applied for the target application. Second, we are devoted into developing energy debugging tool based on power models proposed in this paper. Future work will focus more on how to assist programmers to develop more energy-efficient software.

Acknowledgment

This research is supported by the National Key R&D Program (Grant No. 2016YFB0200100) and the National Natural Science Foundation of China (Grant No. 61361126011, 61133004, 61502019).

References

[1] Capra E, Francalanci C, Slaughter S A. Is software green? Application development environments and energy efficiency in open source applications [J]. Information and Software Technology, 2012, 54(1): 60-71.

[2] Manotas I, Pollock L, Clause J. SEEDS: a software engineer's energy-optimization decision support framework[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 503-514.

[3] Azimi R, Stumm M, Wisniewski R W. Online performance analysis by statistical sampling of microprocessor performance counters[C]//Proceedings of the 19th annual international conference on Supercomputing. ACM, 2005: 101-110.

[4] Bircher W L, John L K. Complete system power estimation using processor performance events [J]. Computers, IEEE Transactions on, 2012, 61(4): 563-577.

[5] Peterson P, Singh D, Kaiser W J, et al. Investigating Energy and Security Trade-offs in the Classroom with the Atom LEAP Testbed[C]//CSET. 2011.

[6] Brooks D, Tiwari V, Martonosi M. Wattch: a framework for architectural-level power analysis and optimizations [M]. ACM, 2000.

[7] Economou D, Rivoire S, Kozyrakis C, et al. Full-system power analysis and modeling for server environments[C]. International Symposium on Computer Architecture-IEEE, 2006.

[8] Rivoire S, Ranganathan P, Kozyrakis C. A Comparison of High-Level Full-System Power Models[J]. HotPower, 2008, 8: 3-3.

[9] Li T, John L K. Run-time modeling and estimation of operating system power consumption [J]. ACM SIGMETRICS Performance Evaluation Review, 2003, 31(1): 160-171.

[10] Bellosa F. The benefits of event: driven energy accounting in power-sensitive systems[C]//Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system. ACM, 2000: 37-42.

[11] Bircher W L, John L K. Complete system power estimation: A trickle-down approach based on performance events[C]//Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on. IEEE, 2007: 158-168.

[12] Sun Y, Zhao Y, Song Y, et al. Green challenges to system software in data centers [J]. Frontiers of Computer Science in China, 2011, 5(3): 353-368.

[13] Lively C, Taylor V, Wu X, et al. E-AMOM: an energy-aware modeling and optimization methodology for scientific applications [J]. Computer Science-Research and Development, 2014, 29(3-4): 197-210.

[14] Phansalkar A, Joshi A, John L K. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite[C]//ACM SIGARCH Computer Architecture News. ACM, 2007, 35(2): 412-423.

[15] H.McGhan, SPEC CPU2006 Benchmark Suite, Microprocessor Report, October 10, 2006

[16] G. Dunteman, Principal Components Analysis, Sage Publications, 1989

[17] STREAM VERSION 5, http://www.cs.virginia.edu/stream, Jan.2013

[18] IOzone, http://www.iozone.org

[19] R. Gonzalez and M. Horowitz. Energy Dissipation in General Purpose Microprocessors. IEEE Journal of Solid-State Circuits,31 (9) : 1277-84,1996

[20] Johann T, Dick M, Naumann S, et al. How to measure energy-efficiency of software: Metrics and measurement results[C]//Proceedings of the First International Workshop on Green and Sustainable Software. IEEE Press, 2012: 51-54.

[21] Bryant R E, David Richard O H, David Richard O H. Computer systems: a programmer's perspective [M]. Upper Saddle River: Prentice Hall, 2003.