Technical Report UPC-DAC-RR-2010-2 Decomposable and Responsive Power Models for Multicore Processors using Performance Counters

Ramon Bertran, Marc Gonzàlez, Xavier Martorell, Nacho Navarro, and Eduard Ayguadé

Barcelona Supercomputing Center
Cr. Jordi Girona 29, 08034 Barcelona, Spain.

Department of Computer Architecture
Universitat Politècnica de Catalunya
Cr. Jordi Girona 1-3, 08034 Barcelona, Spain.

***Abstract*—Power modeling based on performance monitoring counters (PMCs) has attracted the interest of many researchers since it become a quick approach to understand and analyse power behavior on real systems. Moreover, several power aware policies use power models to guide their decisions and to trigger low-level mechanisms -e.g. manage processor frequency-. Hence, the information, the accuracy and the capacity for detecting power phases that a model provides is critical to increase the power-aware research chances and to improve the success of power savings techniques based on such models. In addition, the design of current processors have varied considerably with the inclusion of multiple cores with some resources shared on a single die. As a result, PMC-based power models warrant further investigation on current energy-efficient multicore processors.**

In this paper, we present a methodology to produce decomposable PMC-based power models on current multicore architectures. Besides from being able to estimate accurately the power consumption, the models provide per component power consumption, supplying extra information about power behavior. Moreover, we analyse and validate their *responsiveness* –the capacity to detect power phases–. We produce a set of power models for an Intel® Core™ 2 Duo, modeling one or two cores for a wide set of DVFS configurations. The models are empirically validated using the SPECcpu2006 and compared to other models built using existing approaches. Overall, we demonstrate that the proposed methodology produces more accurate and responsive power models, showing error ranges between [1.89-6]% and almost 100% accuracy in detecting phase variations above 0.5 watts.

I. Introduction

Energy, power density and power consumption have attracted the interest of researchers since they have become limiting factors in processor designs [22, 28]. Power density reduces the reliability and lifetime of processors [31] and limits the operating frequency [7]. Moreover, energy and power consumption are key factors in all market segments. For instance, they are important to extend the battery life of mobile devices or to reduce the need of power supply and energy bills of HPC data centers [8]. As a result, there are several techniques –working at different levels– addressing power consumption issues. All of them rely on accurate methods to gather information about power consumption. Specifically, for software-based solutions the need to estimate and predict power behavior has justified the research on power modeling methods.

Methods based on performance monitoring counters (PMCs) have been shown to be a good solution to estimate power consumption. As a result, their applicability has been demonstrated on several fields such as power management and application profiling. They are used to perform live predictions of power behavior in order to guide power aware policies [20, 30, 4]. Moreover, they are also used in research for quickly exploring new approaches since they allow to profile real systems and full executions of applications, avoiding the need to perform long-time and limited simulations [3, 14, 19, 6, 18, 5] . At the end, they have been crucial in the process of addressing power issues.

The simpler approaches to produce PMC-based power models select the PMCs that are most correlated with power and apply a multiple linear regression to derive the model [3, 19, 6, 5]. This leads to accurate power models, but they do not allow to know how the power is consumed within the processor. In addition, these models are less acceptable to experts and layman because PMCs

that should positively affect the final power consumption may have negative coefficients and vice versa. Figure 1 illustrates the importance of having a model capable of breaking down the power consumption. The figure shows two time slices of the `473.astar` application. During the first time slice on the left, the power consumption –at the top– remains constant. However, we see that the distribution of executed instruction –at the center– and the per component power breakdown[1] –at the bottom– show variations. The Front Side Bus (`FSB`) contribution increases, the contributions of the other components decrease and the overall power does not change. In this case, just reading power measurements or using a model unable to breakdown the power consumption would not detect such variations. Concretely, in this case the data locality changed and data had to be brought from main memory. The time slice on the right shows that the changes of contribution of each component are not always to the same direction as the changes of power. In this example, a model just based on the frontend (`FE`) component would fail to report the second change on power consumption, since its contribution increases but the overall power consumption decreases. These two examples demonstrate the advantages and the necessity of tracking as many components as possible in order to model correctly power behavior.

More complex PMC-based modeling methods solve these issues by splitting the model in microarchitectural components and then use heuristics -such as floorplan information- and manual tuning (not systematic) to calibrate the model. These last techniques have been successfully applied to previous-generation processors such as Pentium IV [14]. Since these processors were not designed taking into account power consumption as a main design constraint, they lack or do not use extensively several power related mechanisms [10, 17] which are common in today's processors. Moreover, the design of current processors have varied considerably with the inclusion of multiple cores with some resources shared on a single die. As a result, these decomposable power models warrant further investigation on current energy-efficient multicore processors. The proposal in this paper performs such investigation on such architectures and presents a new systematic methodology that reduces the complexity of generating decomposable power models for current multicore architectures, avoiding the need of using heuristics or manual tuning. Our proposal is a tradeoff between the component breakdown granularity of the model and the complexity of producing it. As a result, this approach addresses the problem of the lack of decomposable power models for current architectures.

[1]The power components are mapped to the microarchitectural components in the following manner. FE: frontend, INT: integer units, FP: floating point units, BPU: branch prediction unit, L1: L1 cache, L2: L2 cache and FSB: front side bus and main memory.

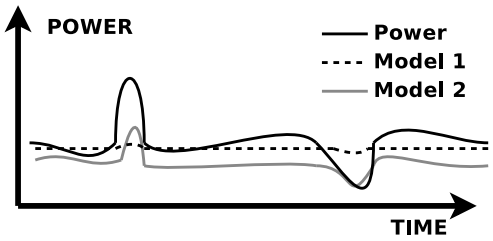


Figure 2. Power model examples.

Besides, the validation of power models have been only carried out via calculating the accuracy of their predictions. However, most of the power aware solutions that are built on top of the models trigger their actions when they detect some variation in power consumption, i.e. a change of power phase. Therefore, it is important to study the capacity of the models to detect power phase changes because it affects the final success of the solutions built on top of them. Concretely, it must be validated that models detect correctly the magnitude and the instant of phase variations. Figure 2 illustrates the importance of this problem. The two models shown have low average error in their predictions with respect to the real power. In fact, Model 1 on average is more accurate than Model 2. In this case,a validation based only on accuracy would say that the Model 1 is more suitable than the Model 2. However, the Model 1 would not predict any change in power behavior and therefore, any policy built on top of it would miss optimization opportunities. The capacity of the Model 2 to guide power policies is better since it reacts in a similar fashion as power. We call this property the *responsiveness* of the model. We conclude that the overall usefulness and applicability of power models rely on their accuracy and also on their responsiveness. Consequently, both metrics should be evaluated in order to validate power models.

In summary, the properties that power models models must have in order to improve their applicability are: accuracy, responsiveness and decomposability. In addition, the methodology for producing them should be applicable on new energy-efficient multicore architectures under different power configurations –i.e. different frequency configurations–. The main contributions of this paper are:

- A new systematic methodology for producing PMC-based power models on multicore architectures. The methodology ensures the decomposability, the accuracy and the responsiveness of the models generated.
- Concretely, we evaluate and validate the model *responsiveness* based on power phase detection ac-

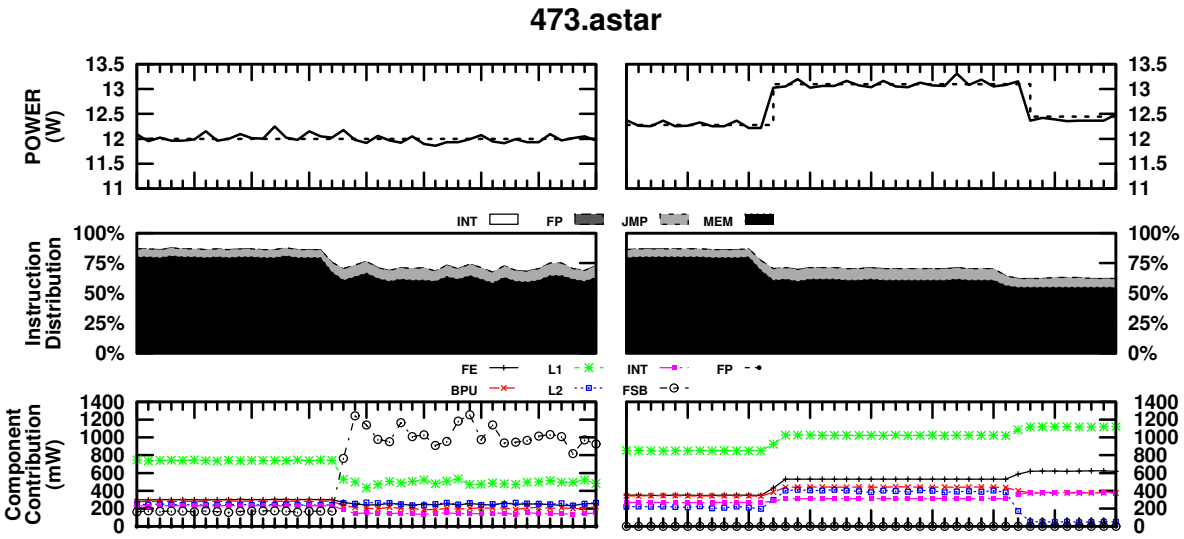


Figure 1. Power, instruction distribution and per component power breakdown for two time slices of the 473.astar SPECcpu2006 benchmark.

curacy.

- A case study is presented for an Intel® Core™ Core 2 Duo, a high performance processor designed with power efficiency as a main design constraint [12]. Single and multiple core models are presented for different DVFS configurations. An empirical validation of the produced models is performed using long executions of the SPECcpu2006 benchmarks. The accuracy validation shows average errors between [1.89-6]%. The responsiveness validation when operating at maximum frequency shows an overall 83.31% accuracy on phase change predictions and almost 100% accuracy for changes bigger than 0.5 watts. As a result, we show that the model produced is sensible to variations as small as 0.25 watts, which only represent a 2.24% variation of the average power consumption of the SPECcpu2006 suite.
- We present a power phase characterization of the SPECcpu2006 benchmarks.

The rest of the paper is organized as follows. Section II describes our modeling methodology including the power component definition, the microbenchmark design, the experimental infrastructure and the method for producing the power model. The validation of the power models is presented in Section III, including a responsiveness analysis. In Section V we compare and contrast related work. Finally, Section VI conclude by summarizing our results.

II. Methodology

In this section, we describe our methodology for producing power models. The discussion is guided through the presentation of an example for a particular architecture, an Intel® Core™ 2 Duo [12]. The final outcome of this section is a set of power models with three main characteristics. First, the model is able to breakdown the power consumption among several architectural components. Second, the model incurs in a small average error in power prediction. And last, the model is responsive in front of power variations. These last two characteristics are validated in Section III and the first one is ensured by the methodology.

In short, our methodology for producing decomposable power models follows the common modeling steps. First, we define the model inputs which in our case they are the power components activity ratios. Second, we define the training data which in our case will be generated using microbenchmarks. Third, we collect the required data in order to train and validate the model respectively. And fourth, the model is built using our method. The main differences of our methodology over previous works are three: the basic rules for defining power components; the microbenchmarks design which isolate or decouple power component activities allowing the generation of decomposable power models; and finally, the model generation which takes profit of the specifically generated training data to built accurate, responsive and decomposable power models.

It should be noted that even though the example presented is for a particular architecture, the methodology can be applicable to other architectures. Particularly, one should redefine and redesign the power components and the microbenchmarks in order to fulfill the requirements of our method. The rest of the section presents in detail the previous steps and justifies their rationale.

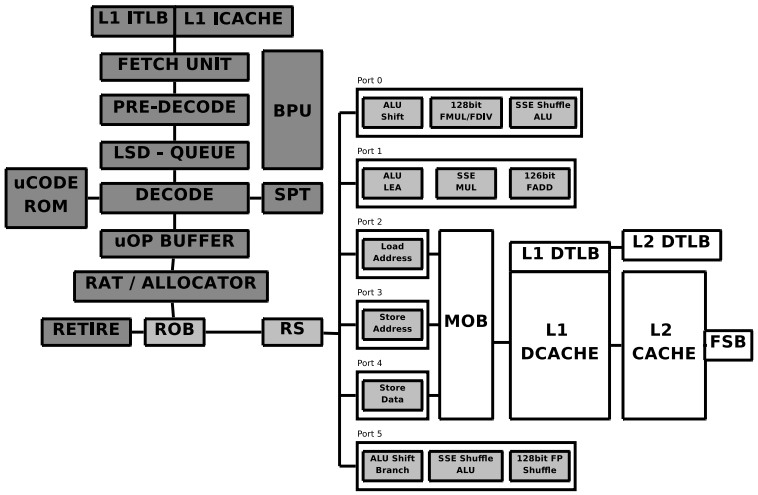


Figure 3. Single core view of Intel® Core™ 2 Duo microarhictectural components.

A. *Defining power components*

The inputs of our model are component activity ratios, which have been shown to be good proxies for power estimation [3, 14]. As a result, the first step towards the final power model is to define the power components and their associated activity ratio formula.

Our objective is to reach a component definition with as many as possible power components that represent a single microarchitectural component in order to have a more detailed power model. However, there are two main restrictions to this direct-mapping between microarchitectural components and power components. First, some microarchitectural components are tightly related in the sense that even if they are monitored by a different set of PMCs, they expose very similar levels of activity. For such components it is not possible to discern directly their power contribution because their activity can not be isolated or decoupled from the activity of other microarchitectural components. And second, there are also microarchitectural components that do not have any PMC that reports their activity. Therefore, due to these restrictions, in our approach a power component can represent one or a set of microarchitectural components. This approach is a reasonable tradeoff between the undecomposability of simpler approaches [30] and the complexity of the microarchitectural-level decomposable ones [14].

As a case study, we explain the power component definition for an Intel® Core™ 2 Duo. We have identified more than 25 microarchitectural components on this architecture. Figure 3 shows an schematic view of them classified into three categories in order to facilitate the explanation: the in-order engine (dark gray), the memory subsystem (white) and the out-of-order engine (light gray).

The in-order engine includes some microarchitectural components that do not have PMCs that report their activity. Moreover, the activity of each of these components can not be isolated since they are in the in-order part of the pipeline, i.e. activity in the `FETCH` unit, means activity in the `PRE-DECODE` unit. Therefore, we group the whole in-order engine components –except the branch prediction unit (`BPU`)– as one power component, namely, the frontend (`FE`) of the processor. The decision to define the `BPU` power component separately is based in the fact that it includes several predictors that can consume between 10% and 40% of the whole processor power budged [23].

For the memory subsystem, it is not possible to generate activity in one memory level without causing activity in the previous one due to its stacked configuration. However, it is possible to decouple their activity. As a result, the whole memory subsystem is divided in three power components: the `L1` cache, the `L2` cache and the Front Side Bus (`FSB`), which also represents the main memory.

The parts of the architecture in the out-of-order engine define the `INT`, `FP` and `SIMD` power components. At the end, eight power components are defined. Table I summarizes them in conjunction with their related microarhictectural components and PMC-based activity formulas.

B. *Designing microbenchmarks*

We need empirical data in order to train our power model. For that purpose, a set of microbenchmarks is designed. This is an important step in the methodology because not considering a significant set of possible

Power Component	Activity formula	Modeled components
FE	UOPS_RETIRED:ANY/ CPU_CLK_UNHALTED:CORE_P	L1_ITLB, L1_ICACHE, FETCH_UNIT, PREDECODE, LSD, DECODE, uCODE ROM, SPT, uOP BUFFER, RAT, ROB, RETIRE
INT	(RS_UOPS_DISPATCHED_CYCLES:PORT_0 + RS_UOPS_DISPATCHED_CYCLES:PORT_1 + RS_UOPS_DISPATCHED_CYCLES:PORT_5 - FP_COMP_OPS_EXE - SIMD_UOPS_EXEC - BR_INST_RETIRED:ANY)/ CPU_CLK_UNHALTED:CORE_P	Integer arithmetic units
FP	FP_COMP_OPS_EXE/ CPU_CLK_UNHALTED:CORE_P	Floating point arithmetic units
SIMD	SIMD_UOPS_EXEC/ CPU_CLK_UNHALTED:CORE_P	SIMD arithmetic units
BPU	BR_INST_DECODED/ CPU_CLK_UNHALTED:CORE_P	BPU and branch execution
L1	L1D_ALL_REF/ CPU_CLK_UNHALTED:CORE_P	LD/ST execution, MOB, L1, L1 DTLB, L2 DTLB
L2	L2_RQSTS:BOTH_CORES:ANY:MESI/ CPU_CLK_UNHALTED:CORE_P	L2
FSB	BUS_DRDY_CLOCKS:ALL_AGENTS/ CPU_CLK_UNHALTED:BUS	FSB and memory

Table I
POWER COMPONENTS DEFINED FOR A INTEL® CORE™ 2 DUO IN CONJUNCTION WITH THEIR PMCS-BASED ACTIVITY FORMULAS AND MICROARHICTECTURAL COMPONENTS THAT REPRESENT.

scenarios incurs in inaccuracy of the final model [32]. We do not use real applications to train the model due to two reasons. First, we want a model suitable for any workload; therefore, the training set should be workload independent. And second, in order to be able to derive the contribution of each power component to the total power consumption it is mandatory to be able to isolate or decouple their activity. This is required in order to apply multiple regression techniques without collinearity problems [24].

The microbenchmark design is always a source of inaccuracy since it is not possible to cover all possible scenarios of power consumption and activity. For example, for exercising the `INT` units we can use different instructions that will consume differently. Similarly, different inputs produce different power consumption. In conclusion, the microbenchmark suite should cover enough variety of power behavior in order to minimize all these sources of inaccuracy.

For our case study, we have designed 97 microbenchmarks. Their structure is an infinite loop containing a specific sequence of assembly instructions with distinct dependency chains in order to explore different activity ratios. However, we had to face the problem of the inherent correlation between the `FE` component and the rest of components. We solved it by introducing diverse combinations of `fxch`[2] instruction sequences. The instruction sequence within the infinite loop is long enough to minimize the effect of the branch instruction at the end of it. Table II summarizes the range of activity ratios that are explored for each power component. For example, for the `INT` component we have designed 13 benchmarks with `INT` activity ratios ranging from 1 to 3 and `FE` activity ratios from 1 to 3.45. Notice that we are able to have a `FE` activity of 3.45 uops/cycle with only 3 uops/cycle on the `INT` component and the rest of components unused. This decoupling of activity is the effect of the `fxch` instruction and it is mandatory to be able to form a decomposable power model afterwards.

The power consumption for entire set of microbenchmarks is low, between 5 and 12 watts. It is important to remark the difficulty of modeling such a power-efficient platform. In our case, 1 watt absolute error on the power prediction represents much more percentage error than the same absolute error on more power hungry platforms.

Next section describes the experimental framework and the data gathering process used to obtain the power consumption.

C. Collecting Data

All the experiments have been carried out on a workstation that features an Intel® Core™ 2 Duo T9400 processor [12] and two modules of 2GB of memory. The embedded controller firmware of the platform provides information about the power source, which fulfills the SMAPI specification [29]. This specification defines an interface to obtain power consumption measurements with a guaranteed granularity and accuracy. As a result, we gather power measurements in a granularity of milliwatts with a maximum error of 2%.

The platform runs a Linux kernel 2.6.28 with the required patches to allow PMC readings. The `tp_smapi` [1] module is loaded in order to be able to gather the power information. This module creates some entries in the `/sys` filesystem that provide information such as the current power consumption. We use a modified version of `pfmon` [2] to access to the PMCs and power consumption simultaneously. All measurements are obtained running the experiments in standalone mode to minimize interferences. We have switched off all sources of power consumption –i.e. the display– that we do not want to interfere to the power model generation. For platform components where it is not possible to switch off, we configured them at constant operation mode. For example, we found that the fan of the processor introduces up to 0.5W variations on power consumption. Hence, we configured it to always operate at full-speed. Moreover, in order to minimize external and unwanted variations, all the experiments were performed at a constant temperature of 20 grads Celsius and a 80% battery charge. This configuration allows us to ensure the same start conditions for all the empirical experiments without running out of battery. Under this conditions, we tracked the power consumption during five minutes

[2]As far as we know, the *fxch* is the only one instruction that does not dispatch any operation to the out-of-order engine. It only performs register renaming, therefore it allows to decouple the activity on the `FE` and the components in the out-of-order engine.

Microbench set	#	FE Activity	INT Activity	FP Activity	SIMD Activity	BPU Activity	L1 Activity	L2 Activity	FSB Activity	Power Range
FE	1	1	0	0	0	0	0	0	0	5587mW
INT	13	1-3.45	1-3	0	0	0	0	0	0	5792mW-8114mW
FP	9	0.2-1.98	0	0.2-1	0	0	0	0	0	5087mW-6887mW
SIMD	12	1.85-3.29	0	0	0.99-2.63	0	0	0	0	6966mW-9274mW
BPU	5	0.42-1.14	0-0.42	0	0	0.46-1	0	0	0	5761mW-8521mW
L1	16	1-2.97	0	0	0	0	0.66-2	0	0	6571mW-8160mW
L2	12	0.12-0.42	0	0	0	0	0.11-0.22	0.11-0.21	0	8600mW-10112mW
FSB	18	0.02-0.14	0	0	0	0	0.02-0.04	0.02-0.04	0.58-0.71	10976mW-12205mW
MIX	11	1.63-3.95	0-1	0-0.8	0-1.97	0-0.34	0-1.97	0-0.07	0-0.34	7318mW-11298mW
Total	97	0.02-3.95	0-3	0-1	0-2.63	0-1	0-2	0-0.21	0-0.71	5087mW-12205mW

Table II
MICROBENCHMARK CHARACTERISTICS. ACTIVITY RATIOS AND POWER CONSUMPTION FOR AN INTEL® CORE™ 2 DUO.

before each experiment and we found the baseline power consumption for the idle system to be 9.8W. The fact that for all the experiments carried out we found a fairly constant idle power consumption demonstrated that we nullified possible interferences. For the purpose of the model formation only, we assumed that baseline, 9.8W, to be the power consumption of the platform except the processor and the memory. This assumption and the accuracy ensured by the SMART specification avoids the need of more complicated measuring techniques [30].

The performance monitoring unit of the processor is not able to track all the selected PMCs simultaneously. Therefore, we grouped them in sets that can be sampled at the same time. We programmed `pfmon` to switch PMC sets every 10ms and print the PMCs values and power measurements every 2 seconds. Although it is widely accepted that the PMC sampling does not introduce unacceptable inaccuracies [16, 14], we measured its effect, resulting in an interference of less than 150mW. In summary, we executed `pfmon` configured to account events in user and system level, switch between counter sets every 10ms and write the values every 2 seconds.

To generate the input data for training the power model we have run every microbenchmark during three minutes with only one core enabled, collecting a trace of 90 samples. We have validated that each microbenchmark stress the component it was designed to do so by applying the formulas in Table I. Moreover, we checked that during the entire trace of each microbenchmark the PMC activities and power consumption remain nearly constant. We take the average over the 90 samples so that the possible errors and noise are minimized as much as possible [32]. Similarly, we have collected the data for 26 applications of the SPECcpu2006 [13] benchmark suite. In this case, every benchmark ran for 30 minutes or less if it ended earlier. This data is going to be used to validate the power models.

All gathered data allows for explaining the difficulties to relate the processor activity and the actual power consumption. We have plotted the evolution of all components for three applications of the SPECcpu2006 benchmark suite (473.astar, 482.sphinx and 416.games) for a time slice. We accompany this data with actual power measurements and the instruction distribution between load, store, integer, floating point and branch instructions. Figure 4 shows the plots. The case of 473.astar exposes two power variations. Power consumption is incremented for about 900mW due to a variation of the instruction distribution: the core starts executing more integer operations. We observe that the FE activity is incremented, which basically implies that more instructions are being executed per cycle. Naturally, the processor starts consuming more power. However, notice that at the power fall -also caused by a change of the instruction type distribution, in fact, even more integer instruction are being executed- the FE activity increases too. In these two points the power consumption varies differently in each case. If one analyzes the other components, it realizes that the main components that justify the power variations are the BPU and L2 cache. At the first inflection point, the BPU goes from an activity index of 0,16 to 0,24 and the L2 activity goes from 0,01 to 0,016. At the second inflection point these two indexes recover similar levels to the previous ones. The case of 482.sphinx shows one inflection point caused by an increment of the FSB activity, which means that the processor enters in a phase where it misses data in the cache hierarchy and starts accessing main memory. This causes an immediate fall on the activity of the FE, and less instructions are being executed per cycle. Hence, power consumption decreases. From the two examples, we conclude that any power aware policy must be able to detect the cause of a power variation. Finally, the case of 416.gamess shows that simultaneous changes on two different components might compensate one to each other and result in no power variation. In this case the application switches between integer activity to floating point activity.

In conclusion, two main observations have to guide the elaboration of power models. First, we have seen that power variations might have very different causes related to specific trends in the activity on every component. In order to implement power aware policies on top of power models, the models have to clearly identify these

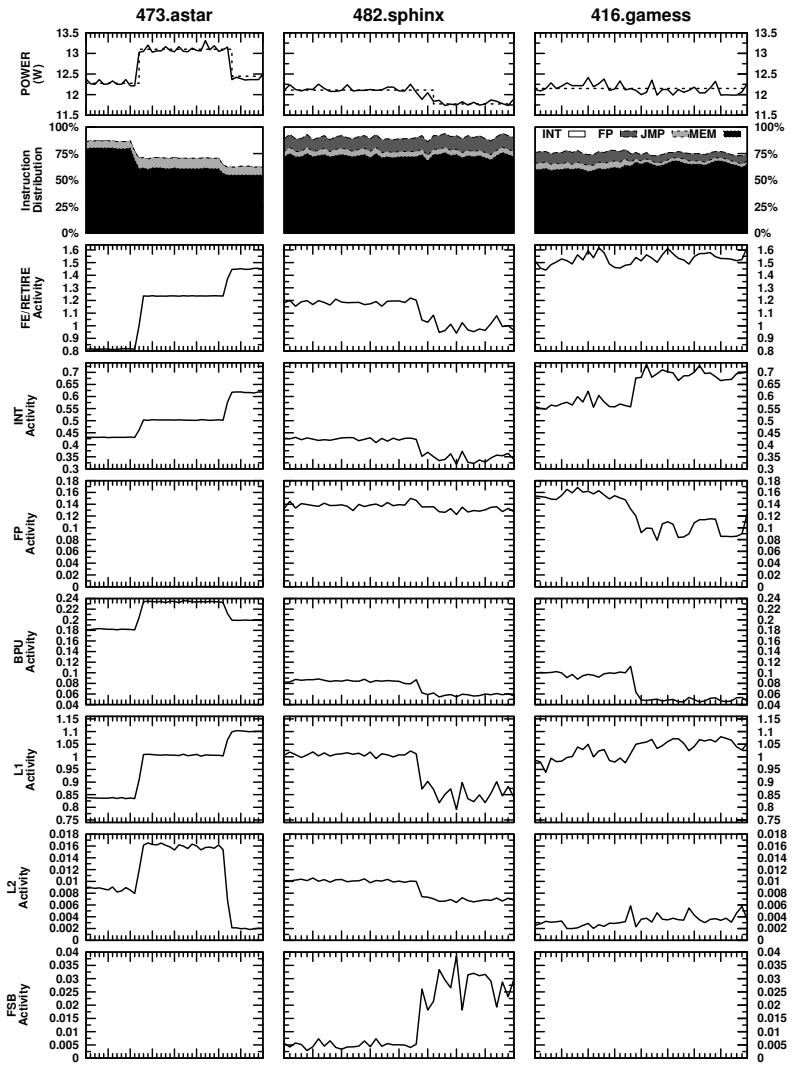


Figure 4. Power consumption, instruction distribution and component activity ratios during three time slices of the `473.astar`, `482.sphinx` and `416.gamess` SPECcpu2006 benchmarks.

causes and the main components of involved in power variations.

D. Modeling power

1) Single Core modeling: We model the power consumption using a multiple linear equation with seven input variables, one for each power component defined. The SIMD component is not used because the SPECcpu2006 suite do not present any activity in such component. Therefore, the total power consumption is expressed as:

$$P_{total} = \Big(\sum_{i=1}^{i=numcomp} AR_i \times P_i \Big) + P_{static} \quad (1)$$

where P_i is the weight of component i, which we need to solve, and AR_i is its activity ratio. The $AR_i \times P_i$ represents the dynamic power consumption of component i, and P_{static} represents the overall static power consumption of all components. This approach is commonly used for generating PMC based power models [14, 32, 30]. However, instead of applying directly linear regression techniques [11, 24], we derive the marginal effect of each component to the overall power consumption. Firstly, we compute the P_i of each component – except the `FE`– separately using a multiple linear regression with the component related microbenchmark set as input. For example, we computed P_{INT} using only the `INT` microbenchmark set, which only stresses the `INT` component. The components which activity is not completely isolated -i.e. `L2` and `FSB`- are calculated incrementally. For example, before deriving P_{L2} from the `L2` microbenchmark subset, we derive first P_{L1} and

use that value for the P_{L2} estimation. The rationale that justifies this methodology is that the collinearity that may exist between components that are not stressed together is canceled. Therefore, we get better estimates about what is the contribution of each component. The goodness of the estimates also relies on the collinearity between the exercised component and the FE, which we minimized when we designed the microbenchmarks. Finally, we use again a multiple linear regression using as input the MIX microbenchmark subset to derive P_{static} and P_{FE}. We use that set because it represents the common activity scenario, when there is activity across all processor components, canceling the known effect of under-estimate their contribution [32]. Moreover, this process have been automatized using a R script [27] in order to be applicable systematically. This is possible because no manual tuning is required, demonstrating that with an appropriate set of microbenchmarks it is possible to generate decomposable and accurate power models.

Table III summarizes the models generated. The first four columns indicate the model name, the methodology used to produce the model, the training dataset and the frequency used to generate it respectively. Using our methodology (inc. in Table III), we have generated one model for each frequency available in our experimental platform. Note that most of these frequency points are not recommended by the manufacturer, but we found to be reliable and we used them to obtain more data. We refer to our models as MICRO models and can be found at the top eleven rows in Table III.

Analysing in detail the MICRO models in Table III, one can see that the weight of all the components increases exponentially with frequency. An exception is the FSB component which remain fairly constant on two values. The reason is that the frequency of the memory bus does not change with the frequency of the processor. Actually, it is 133GHz for core frequencies below 1.6GHz and 266GHz otherwise. The fact that the power weights have an exponential relation –like power–, provides an evidence about the effectiveness of the proposed methodology to model power behavior.

We have also produced other models using already known techniques to able to compare them, in Section III, against our MICRO models. To generate the MICRO-LIN model, we have applied a multiple linear regression using as input all the microbenchmarks. Moreover, we produced a set of over-trained models taking an incremental number of input variables –i.e. the FE model only uses as input the FE activity, the +FSB model uses the FE activity plus the FSB activity–. These models are also generated applying multiple linear regressions but using as input the SPECcpu2006 data. Notice that these last models do not fulfill our objectives because they do not track all the components, the component weights are not acceptable to decompose power –i.e. negative weights– or they are not workload independent.

2) Multiple Core modeling: We use an accumulative approach for modeling multiple cores assuming that each core behaves equally. This assumption have been already done in similar works [30]. Hence, we apply the single core model to each core in the architecture. Therefore, we express the total power consumption as:

$$P_{total} = \sum_{j=1}^{j=numcores} ((\sum_{i=1}^{i=numcomp} AR_{ij} \times P_i) + P_{static}) \quad (2)$$

where the P_i of each component is the same of the single core model but the formulas to calculate the AR_ij should be modified to perform per core accounting. This is straightforward since PMCs already support per core event masks. Besides, it is needed to redefine the P_{static} component of the model because that component represents the static power of the entire processor. As a result, the P_{static} value obtained from single-core model training set is not valid because it accounts for shared resources which static power should not be replicated –i.e. the L2 cache–. To recalculate it, we need an input data where all the cores modeled in the architecture are active. For that purpose, we re-executed the MIX subset of the microbenchmark suite on both cores at the same time in order to get a suitable training data. Then, a new P_{static} was generated by using a linear regression and dividing the obtained value by the number of cores.

3) Frequency based modeling: Analysing in detail the MICRO models in Table III, one can see that the weight of all the components increases exponentially with frequency. An exception is the FSB component which remain fairly constant on two values. The reason is that the frequency of the memory bus does not change with the frequency of the processor. Actually, it is 133GHz for core frequencies below 1.6GHz and 266GHz otherwise. In order to produce a unique model for all the frequencies, we have applied exponential regressions to each component, obtaining regression coefficients close to 1. Then, we have averaged the P_{FSB} weight on each range in order to build a pairwise model. We refer to it as the MICRO-FREQ model in Table III. For this model, the weight of each component is computed as $x \times y^{freq}$, where x and y are the first and the second value in the each box on the table. The weight of the FSB component is the second value in the box if the frequency is below 1.6GHz or the first value otherwise.

Moreover, we see for frequencies below 1.6GHz, the component weights differ. The reason is that at that point, the Intel SpeedStep technology reduces the frequency of the memory bus.

Taking into account that the power consumption can

Model	Method	training set	Frequency	P_{FE}	P_{INT}	P_{FP}	P_{BPU}	P_{L1}	P_{L2}	P_{FSB}	P_{STATIC}
MICRO	inc.	micro	2.54GHz	789	261	502	1908	856	24437	8852	8701
MICRO	inc.	micro	2.4GHz	709	237	424	1647	763	22078	8816	7830
MICRO	inc.	micro	2.26GHz	621	208	357	1463	693	19525	8580	7036
MICRO	inc.	micro	2.13GHz	566	193	357	1334	620	17676	8553	6272
MICRO	inc.	micro	2.0GHz	506	174	253	1108	542	15911	8572	5590
MICRO	inc.	micro	1.86GHz	433	148	261	995	476	13770	8232	4896
MICRO	inc.	micro	1.6GHz	336	113	184	769	360	10613	8680	3508
MICRO	inc.	micro	1.2GHz	239	80	148	582	266	7848	10268	2581
MICRO	inc.	micro	1.06GHz	208	72	131	500	238	6896	10273	2176
MICRO	inc.	micro	0.93GHz	177	60	103	404	198	5944	10270	1792
MICRO	inc.	micro	0.8GHz	148	55	106	362	169	5069	10815	1309
MICRO-FREQ	inc.+exp	micro	all	73.65/2.58	26.47/2.5	48/2.46	176.45/2.55	85/2.51	2590/2.45	8612/10406	675.61/2.82
MICRO-LIN	linear	micro	2.54GHz	1044	-373	-305	528	89	20795	7965	9674
FE	linear	spec	2.54GHz	623	-	-	-	-	-	-	10444
+FSB	linear	spec	2.54GHz	1318	-	-	-	-	-	9725	9430
+BPU	linear	spec	2.54GHz	1378	-	-	3834	-	-	12155	8859
+CACHES	linear	spec	2.54GHz	491	0	0	1597	3300	41284	16702	6836
ALL	linear	spec	2.54GHz	-621	1325	2809	3416	3750	36983	16892	6845

Table III
MODELS GENERATED FOR AN INTEL® CORE™ 2 DUO.

be expressed as a function of capacitance, voltage and frequency with the well-known equation:

$$P = C \times \ V^2 \times f \tag{3}$$

we validated that the power weights of each component behave as the above equation. For that purpose we tested the following condition:

$$\frac{Weight_{freq_1}}{Weight_{freq_2}} = \frac{(V^2_{freq_1} \times freq_1)}{(V^2_{freq_2} \times freq_2)} \tag{4}$$

to be true for each component and pair of frequencies. The C variable can be simplified because is constant, and the voltage range of our processor is [1.0500-1.1625] Volts. We found average errors between 2.7% to 9% depending on the distance between frequencies. Actually, the error increases with frequency distance. These lower error rates demonstrates that our methodology turns out to be quite effective to model power behavior. Next section presents the empirical validation of our `MICRO` and `MICRO-FREQ` models.

The average error For all the components and adjacent frequencies we get a total average error of 2.7%. For longer distances between frequencies we get a total average error of 9%.

III. VALIDATION AND EVALUATION

The model evaluation is organized in three main sections. The first one covers the `MICRO` model validation for one core, analysing their accuracy and responsiveness compared to other models. The second part validates the `MICRO` model for the whole chip, when two cores are active. And the third part validate the `MICRO` models for all the frequencies studied.

A. One Core Validation

1) Model Error: In this section we compare the `FE`, `+FSB`, `ALL`, `MICRO-LIN` and `MICRO` models in terms of accuracy (average error). In general, the all models behave similarly, showing an average errors in the below of a 3%. However, their standard deviation differs showing that the more components tracked, the lesser the deviation is. This remarks the importance of taking into account as many power components as possible.

One interesting observation is how the error evolves taking into account that there is an incremental relation between the `FE`, the `+FSB` and the `ALL` and `MICRO-LIN` models. Depending on the modeled application, not all components affect the power modeling in the same manner. Table IV shows the average error and standard deviation for every application and every model. For the case of `+FSB`, several applications are modeled more accurately, showing high error reductions. Examples of this behavior are the `435.gromacs`, `453.povray`, `470.lbm` and `471.omnetpp` applications. But other applications like the `434.zeusmp` and `437.leslie3d` do not expose any improvement, showing higher error rates for the `+FSB` model. This suggests that not every application is equally conditioned by the components in the architecture. This supports the idea of having a per component based model in order to never discard any component in the architecture, that for a particular application it has a significant power contribution or correlation. This is the case of the `ALL` and `MICRO-LIN` which in overall present lower error rates.

Table IV also includes the average error for every model. Notice that for all of them the average error is always below a 3%. Initially, this result suggests that accuracy is not significantly improved by the component decomposition. However, the deviation is reduced when more components are taken into account recommending the usage of as much components as possible. Besides, there is one qualitative difference between them: the `MICRO` model can account for the power contribution

Benchmark	Model FE		Model +FSB		Model ALL		Model MICRO-LIN		Model MICRO	
	%error	stdev	%error	stdev	%error	stdev	%error	stdev	%error	stdev
400.perlbench	3.31	1.26	4.09	1.56	1.12	2.66	2.78	1.7	1.48	1.77
401.bzip2	1.2	1.61	1.22	1.7	3.47	2.91	1.21	1.65	1.26	1.59
403.gcc	2.93	3.25	3.69	3.93	3.52	5.16	3.65	4.24	3.4	3.52
410.bwaves	6.07	1.54	8.24	2.58	4.01	1.79	7	2.15	4.89	1.62
416.gamess	0.98	0.84	1.1	1.04	1.11	1.21	0.93	1.13	0.88	1.03
429.mcf	1	1.66	1.59	2.82	1.15	3.98	4.19	2.97	1.28	2.6
433.milc	1.28	1.09	2.48	1.91	1.25	2.25	1.51	1.67	1.21	1.28
434.zeusmp	1.5	1.96	3.47	3.53	2.4	3.03	2.21	2.74	1.64	2.21
435.gromacs	12.12	1.54	7.7	1.96	5.96	2.69	8.39	1.88	5.69	2
436.cactusADM	4.14	2.02	3.56	2.16	3.91	3.59	4.16	2.14	2.18	1.98
437.leslie3d	0.56	0.53	3.4	2.48	1.79	1.92	1.51	1.84	1.14	1.1
444.namd	1.34	1.07	1.91	1.23	2.39	2.19	2.08	1.32	2.11	1.39
445.gobmk	3.49	1.05	5.37	1.64	1.32	3.77	3.35	1.42	2.76	1.58
450.soplex	1.24	1.05	1.88	1.91	2.03	2.49	2.31	1.76	0.99	1.22
453.povray	0.6	0.36	0.42	0.65	0.56	0.48	0.34	0.6	0.32	0.53
454.calculix	0.5	2.54	4.85	2.59	3.24	2.7	2.75	1.89	0.87	2.71
456.hmmer	0.44	1.06	0.43	0.93	0.56	1.03	3.08	1.1	1.65	1.08
458.sjeng	3.33	0.78	5.04	0.98	0.5	1.32	4.73	0.93	3.34	0.92
459.GemsFDTD	6.32	1.48	8.03	2.05	1.33	1.6	7.07	1.77	2.56	1.78
462.libquantum	6.37	4.41	9.94	5	3	3.94	10	4.73	3.99	4.31
464.h264ref	3.14	1.03	5.13	1.58	6	1.8	1.79	1.32	3.47	1.38
465.tonto	1.2	2.28	1.48	2.57	2.31	4.25	1.49	2.55	1.41	2.52
470.lbm	18.95	0.82	3.95	1.4	2.85	1.99	5.8	1.37	6.87	1.48
471.omnetpp	4.74	0.75	2.7	0.9	2.8	1.42	2.6	0.78	3.6	0.82
473.astar	2.51	2.18	2.57	2.96	1.67	1.92	3.03	1.85	1.9	2.21
482.sphinx3	1.05	0.88	0.77	1.06	2.99	1.66	0.8	0.99	0.56	0.57
TOTAL	**2.15**	**4.11**	**2.77**	**2.57**	**2.02**	**1.48**	**2.65**	**2.45**	**1.89**	**1.64**

Table IV
AVERAGE ERROR OF THE SINGLE CORE FE, FSB, ALL, MICRO-LIN AND MICRO MODELS FOR THE SPECCPU2006 SUITE.

of every component and the rest of the models can not. In any case, if one just validates the models only using prediction accuracy, all models are valid and expose a similar error levels. Next section proves this assumption to be false on the basis of power phase detection and analysis.

Finally, we have measured the error of the ALL model if applied to the microbenchmarks. The aim of this experiment is to prove that this model is totally limited to the SPECcpu2006 benchmark suite, with no valid applicability to model the processor's power consumption independently of workload and input data set. The average error was 54.77% with a 381% standard deviation. Notice that the MICRO model succeeds in modeling the SPECcpu2006 applications, but the model derived from these applications can not model the microbenchmarks.

2) Model Responsiveness: As we stated earlier, the *responsiveness* of a model is its capacity to react in a similar fashion as power, having the similar inflection points with similar magnitudes. Therefore, to evaluate the responsiveness of a model, we apply a phase detection algorithm to both, the modeled and the real power, in order to compare afterwards their results similarity. The phase detection algorithm used is a modified version of the first pivot clustering algorithm presented in [15]. In our method, the power of a new sample is compared to the power of the current pivot sample, the starter of a phase. If the power of the new sample is within a specified threshold distance of the pivot power, in our case $\pm$250mW, it is assigned to that phase. Otherwise, a new phase is added with the current sample as the pivot. We choose a $\pm$250mW threshold because it represents 2.24% of the average power consumption of the SPECcpu2006 suite. Lowering that threshold would introduce inaccuracies because it will be below the 2% error of the measurement device. The characteristics of each phase are its starting time (timestamp of the pivot sample), its duration, its variation with respect to the prior power phase and the average power consumption for all the samples in the phase.

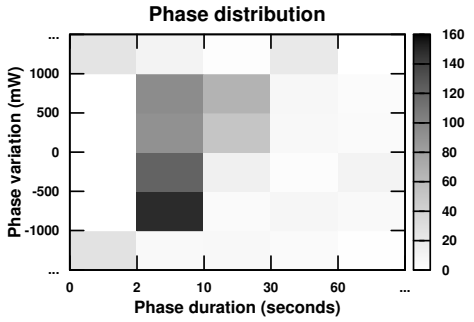


Figure 5. Power phases distribution of the SPECcpu2006.

We classify the power phases in intervals of duration and variation. This allows us to perform analysis by duration and variation, while still being able to easily summarize the power trends. Figure 5 shows the distribution of the phases by its duration and its variation. From the duration point of view, we can observe that most of the phases last from 2 to 10 seconds. From the other point of view, we realize that most phases are defined after inflection points where power varies between 500mW and 1000mW. However, it is important to remark that power variations of more than 1000mW

exist. These power variations correspond to significant targets of power aware policies. We observe that they are concentrated in the 2-or-less-seconds category, although they also appear for phases of 30-60 seconds. Next, we show that there are several phases which have a high variation and sufficient time to be detected and potentially treated by power aware policies.

Table V details the phase characterization for each benchmark sorted by number of phases. Columns 2 to 7 show the number of phases classified by the its variation. It also shows that most of the phases correspond to inflection points of more than 500mW of variation. The next 5 columns show the number of phases classified by its duration. We observe that most phases last from 2 to 10 seconds. The last column is the average power consumption. We see that the average power consumption is about 12W for the entire benchmark suite. The `470.lbm` consumes the maximum power on average -about 14.5 watts- and `435.gromacs` the minimum, about 10.5 watts. The first one, shows high activity ratios in all the components and the second one has low activity ratios in the memory hierarchy and the BPU. At first sight, we see that half of the benchmarks are regular in power consumption, and that few ones concentrate the majority of inflection points. If we analyze the variability, we see that about a 14% of the inflections points have a variation bigger than 1 watt. The `454.calculix`, `403.gcc` and `410.bwaves` centralize the majority of such points with 44 (22+22), 19 (10+9) and 25 (7+18) respectively. The percentage of points with a variation within 0.5 and 1 watts is about 46% of the total. The benchmarks that contribute more to this category are `444.namd` (52+48), `434.zeus` (41+42), `462.libquantum` (19+25) and `465.tonto` (19+19). These benchmarks are also the ones that contribute more to the inflection points with a variation less than 0.5 watt, which account for the 40% of the total. Taking into account phase duration, we see that a 63% of phases are within the (2,10] range and that only a 6.9% of them are shorter than two seconds. This distribution shows that there is room to apply power aware policies since most of the phases are enough long to overcome the overhead of applying them. Moreover, we see that the set of aforementioned benchmarks are also the ones which experiment more shorter power phases. A special case is `454.calculix`, which shows an high number of very short and long phases. In particular, it presents 21 phases less or equal to 2 seconds and 25 between half and one minute long. We have checked that its power consumption is periodic, switching between a very short phase and a long phase with a variation bigger than 1 watt.

After analyzing the insights of the SPECcpu2006 power behavior, we use these results to validate that our power models predict a similar number of phases, with similar variations and durations. For this purpose we use two metrics. The main metric is the percentage of accuracy, which indicates the percentage of times that a model predicts correctly a real power phase. The other metric is the percentage of false positives (the model predicts an nonexistent change of phase). Again, the analysis are summarized taking into account the variation and the duration of the phases.

Figure 6 shows the percentage of accuracy for the model `FE`, model `+FSB` and model `MICRO`. The x and y axes correspond to the phase duration and the phase variation respectively. The z axis is the percentage of accuracy. The surface is colored to facilitate the readability. The darker the color, the more accurate the model is for that duration and variation. The three models show a similar trend. The bigger the variation, the bigger the accuracy is. These means that important changes in power consumption are detected even though all the components are not modeled. However, the final accuracy varies considerably among the different models. The `MICRO` model is almost 100% accurate for changes bigger than 1 watt. However, the `FE` models shows a 58% accuracy on average and the `+FSB` model about a 80%. This difference is important because our model is almost 100% accurate on detecting most important power variations where potentially power aware policies need to be reevaluated. If we take into account the rest of the phases we also see that the `MICRO` model is also the most accurate. In overall, the `MICRO` model shows a 83.31% of accuracy. The `FE` and the `+FSB` models show a 15.22% and a 27.9% of accuracy, respectively. This revokes the assumption that a power model with a low average error is valid for any applicability. We have shown that for phase detection analysis, a model should include as many components as possible.

If we analyze the other metric, the `MICRO` model shows a 34% of false positives and the `FE` and `+FSB` models show 10.81% and a 23.77% respectively. These last two models are less responsive, and therefore they are likely to not have false positives at the expenses of having plenty of false negatives.

We have also studied the distribution of false positives and negatives. Figure 7 shows it for the `MICRO` model. In the figure, the false positives are concentrated in the phases with a low variation or short duration, demonstrating that the false positive ratio is not an issue for power aware policies. Actually, the model performs fairly well for the phases and variations which are interesting for power policies. The rest of the models are not depicted due to space limitations but we point out some their trends. The `+FSB` model and the `FE` model show less false positives in overall, but have some of them in phases with huge variations. The reason is that

	Phase variation(mW)						Phase duration (seconds)					
BENCHMARK	< = -1000	(-1000,-500]	(-500,0]	(0,500]	(500,1000]	>1000	<=2	(2,10]	(10,30]	(30,60]	>60	POWER(mW)
434.zeusmp	0	41	57	48	42	1	1	170	18	0	0	11778.8
444.namd	0	52	17	23	48	0	0	75	65	0	0	11264.9
454.calculix	22	5	4	4	6	22	21	17	0	25	0	13157.1
462.libquantum	2	19	19	24	25	2	1	83	6	1	0	10636.5
403.gcc	10	9	10	9	9	9	6	35	14	1	0	11679
410.bwaves	7	5	0	0	6	18	16	9	3	8	0	11082.1
465.tonto	2	19	13	19	19	0	2	55	11	4	0	12131
416.gamess	0	7	4	1	8	0	0	2	9	4	5	12124.9
429.mcf	2	4	2	2	0	3	2	2	2	5	2	11453
400.perlbench	1	1	7	6	1	2	2	9	7	0	0	12628.4
433.milc	0	3	1	0	3	1	1	3	0	1	3	11665.7
401.bzip2	0	1	5	6	3	0	0	4	5	1	5	11888.2
473.astar	0	1	2	3	1	0	0	0	1	2	4	12413.9
456.hmmer	0	1	2	2	1	0	0	3	2	0	1	12425.9
450.soplex	0	1	1	1	1	0	0	1	1	0	2	11659.2
436.cactusADM	0	0	1	2	0	0	0	3	0	0	0	11507.2
482.sphinx3	0	0	1	2	0	0	0	0	0	0	3	11959.1
459.GemsFDTD	0	0	1	1	0	0	0	1	0	0	1	10999.4
470.lbm	0	0	1	1	0	0	0	1	1	0	0	14410.6
471.omnetpp	0	0	1	0	0	0	0	0	0	0	1	12445.8
435.gromacs	0	0	0	0	0	0	0	0	0	0	0	10573.8
437.leslie3d	0	0	0	0	0	0	0	0	0	0	0	10863.6
445.gobmk	0	0	0	0	0	0	0	0	0	0	0	12453.6
453.povray	0	0	0	0	0	0	0	0	0	0	0	12266.8
458.sjeng	0	0	0	0	0	0	0	0	0	0	0	12510.7
464.h264ref	0	0	0	0	0	0	0	0	0	0	0	12109.6
TOTAL	46	169	149	154	173	58	52	473	145	52	27	

Table V
POWER PHASES CHARACTERISTICS OF THE SPECCPU2006.

sometimes there is a huge variation in the activity of the components modeled, but the activity of the non-modeled components cancel (example 3 of Figure 4) the effect on power. As a result, such models predict non-existent huge changes.

B. Two Core Validation

We have selected a subset of the SPECcpu2006 benchmarks to evaluate the power models when two cores are active and one application runs on each core. The applications have been chosen according to the classification and characterization of the SPECcpu2006 for multi-programming evaluation [26, 25]. We have selected a total number of 6 applications and for every pair of applications (15 experiments), we ran each for ten minutes on one dedicated core.

Table VI shows the power and model average errors on two cores. The models were extended as explained in Section II-D2. In general the MICRO model outperforms the other two and shows an average error of 4.63% while the other two models suffer from average errors of 5.12% and 6.09%, with higher standard deviations. This magnitudes are mainly caused by the pairs generated in conjunction with the 462.libquantum application. For these cases, all models produce inaccurate estimations. We suspect that the reason is related to the hardware pre-fetcher because we it affects heavily the application performance, compared to the rest of benchmarks. The rest of pairs generally expose an acceptable error (below the 5% border).

In conclusion, the extended MICRO model succeeds in modeling the entire multi-core chip. Table VII shows a direct application of the it: per core/per application power consumption. Notice that this model is the only one that can deliver this data.

C. Frequency validation

Table VIII summarizes the average error of the MICRO models. For all the frequencies, the MICRO models obtain low percentages of error, which decreases with frequency. The absolute error remains in the same magnitude for all the frequencies, showing that the methodology used is valid for any given power configuration. The MICRO-FREQ model follows the same trends, but with higher error rates due to the exponential and P_{FSB} approximation.

To sum up, we have applied the proposed methodology on a particular architecture and we have validated that the power models generated are valid in terms of accuracy for different DVFS configurations. Moreover, a case study of responsiveness analysis was presented

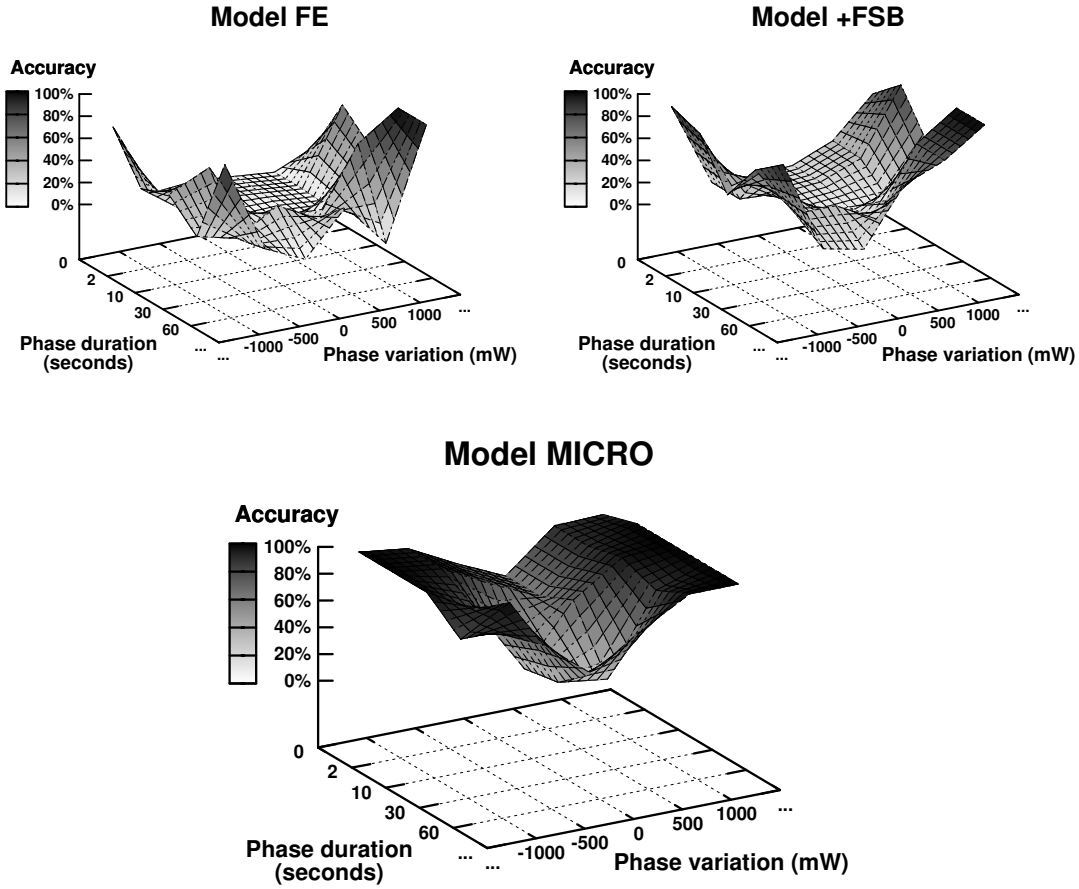
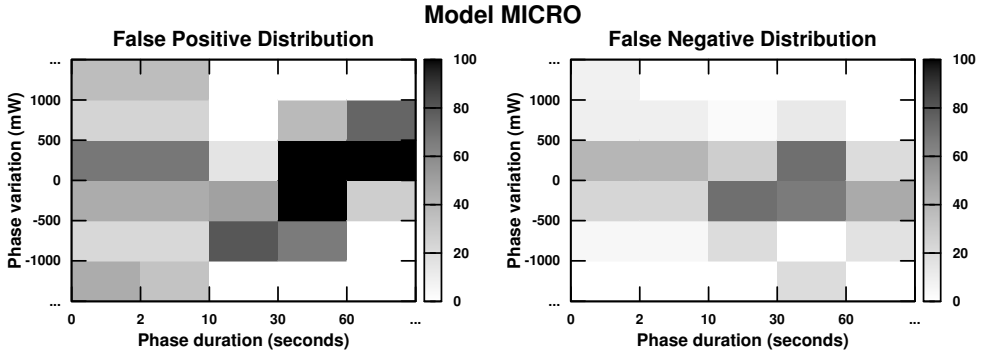


Figure 6. Accuracy distribution for three of the models studied.

Benchmark	Power (mW)	Model MICRO %error	Model MICRO stdev	Model FE %error	Model FE stdev	Model +FSB %error	Model +FSB stdev
400.perlbench-436.cactusADM	18540.20	4.10	1.85	4.16	1.38	3.56	1.41
400.perlbench-453.povray	19686.18	6.08	1.09	0.52	0.53	0.60	0.58
400.perlbench-454.calculix	20462.40	4.38	1.52	2.33	0.97	5.10	1.94
400.perlbench-462.libquantum	24034.74	6.80	1.78	17.46	1.21	13.54	1.77
400.perlbench-473.astar	19377.92	5.62	3.11	2.71	1.89	1.91	1.33
462.libquantum-436.cactusADM	22661.80	9.22	2.08	13.78	1.54	8.96	2.12
462.libquantum-453.povray	23896.88	9.02	1.38	16.37	1.25	13.34	1.55
462.libquantum-454.calculix	24716.67	10.15	1.45	15.12	1.73	9.31	2.01
462.libquantum-473.astar	23353.39	9.26	2.00	17.01	1.69	11.70	2.97
473.astar-436.cactusADM	18964.17	4.40	3.76	3.01	1.80	3.04	2.15
473.astar-453.povray	20049.61	3.12	2.87	3.21	3.35	3.62	3.40
473.astar-454.calculix	20798.19	3.04	3.59	3.01	2.73	3.78	1.79
436.cactusADM-453.povray	18477.38	1.47	1.09	5.09	1.50	4.36	1.50
436.cactusADM-454.calculix	19269.42	1.68	1.17	6.06	2.41	8.81	3.08
453.povray-454.calculix	20163.70	2.84	0.89	3.46	1.30	6.78	2.52
AVERAGE		**4.63**	**2.9**	**5.12**	**6.31**	**6.09**	**4.15**

Table VI
AVERAGE ERRORS FOR THE TWO CORE MODELING.



Phase Duration	Phase variation (mW)											
	< = -1000		(-1000,-500]		(-500,0]		(0,500]		(500,1000]		>1000	
	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP
<=2	0	12	0	0	0	0	0	0	0	0	2	9
(2,10]	0	2	7	33	28	54	36	62	9	23	0	4
(10,30]	0	0	1	4	10	7	14	8	2	0	0	0
(30,60]	1	0	0	6	2	11	5	10	1	3	0	0
>60	0	0	1	0	5	3	1	5	0	3	0	0

Figure 7. Distribution of False Positives and False Negatives for the model MICRO.

Benchmark	Model MICRO (mW)	CPU0 1st Benchmark (mW)	CPU1 2nd Benchmark (mW)
400.perlbench-436.cactusADM	19298.02	10707.80(55.49%)	8590.22(44.51%)
400.perlbench-453.povray	20882.35	10762.56(51.54%)	10119.79(48.46%)
400.perlbench-454.calculix	21352.66	10768.96(50.43%)	10583.71(49.57%)
400.perlbench-462.libquantum	22398	10954.22(48.91%)	11443.79(51.09%)
400.perlbench-473.astar	20444.1	10776.56(52.71%)	9667.53(47.29%)
462.libquantum-436.cactusADM	20569.68	11432.35(55.58%)	9137.33(44.42%)
462.libquantum-453.povray	21739.6	11508.96(52.94%)	10230.64(47.06%)
462.libquantum-454.calculix	22206.72	11495.94(51.77%)	10710.77(48.23%)
462.libquantum-473.astar	21188.89	11399.75(53.80%)	9789.13(46.20%)
473.astar-436.cactusADM	18131.4	9523.93(52.53%)	8607.47(47.47%)
473.astar-453.povray	19776.85	9619.77(48.64%)	10157.08(51.36%)
473.astar-454.calculix	20206.38	9628.47(47.65%)	10577.91(52.35%)
436.cactusADM-453.povray	18708.42	8590.34(45.92%)	10118.09(54.08%)
436.cactusADM-454.calculix	19143.36	8569.32(44.76%)	10574.04(55.24%)
453.povray-454.calculix	20732.2	10142.99(48.92%)	10589.20(51.08%)

Table VII
ACCOUNTING FOR THE POWER CONTRIBUTION OF EACH CORE.

Frequency (Ghz)	Avg.Power (mW)	Model MICRO-FREQ			Model MICRO		
		%error	stdev	error (mW)	%error	stdev	error (mW)
0.800	2113.46	9.07	5.65	191.63	6.07	6.96	128.28
0.933	2654.6	4.75	5.24	126.07	4.76	5.22	126.42
1.066	3120.04	6.25	5.38	194.9	4.1	4.33	127.96
1.200	3584.31	7.45	4.86	266.93	3.56	3.73	127.61
1.600	4795.22	3.64	3.51	174.74	3.27	3.73	156.59
1.866	6407.61	3.11	3.01	199.15	2.47	2.59	158.03
2.000	7240.7	2.79	2.61	202.13	2.23	2.36	161.79
2.133	8100.77	2.11	2.27	171.14	2	2.34	161.85
2.266	9023.32	1.99	2.15	179.14	1.89	2.12	170.61
2.400	10006.21	3	2.63	299.75	1.9	1.96	189.64
2.533	11143.27	5.29	2.94	589.31	1.89	1.64	210.38
AVERAGE		**4**	**2.31**	**216.5**	**2.85**	**1.4**	**154.15**

Table VIII
AVERAGE ERROR SUMMARY OF THE MICRO MODELS FOR ALL THE FREQUENCIES.

for a particular frequency, 2.53GHz. Overall, the models present fairly good results, outperforming the ones generated using previous modeling techniques.

IV. Applicability

This section describes two applicability examples of the power models based on a per component basis and the hardware counters. The first one corresponds to the ability to attach the power consumption to a particular application, moreover, to aprticular time period of an application's execution. The second corresponds to the new capabilities that the power models enhance in terms of the analysis of the power evolution of an application.

1) Component impact analysis: This section studies how the power components contribute at every inflection point where power changes between two phases. Figure 8 shows how every power component contributes at every inflection point. Each chart shows statistics for a particular type of power change, depending on the power increment/decrement that is observed at the moment of the inflection point. For instance, chart (a) shows statistics for all inflection points where power is decremented by 1000mW or more. In each chart, the power components are placed in the x-axis, and the power contributions are placed in the y-axis as percentages. For example, the category 10%-L1 informs about how often it happens that the power contribution of the L1 cache accounts for a 10% or less of the total power change. The color of the rectangle represents a percentage over the total number of power variations. Therefore, very light color rectangles indicate that something rarely happens and really darker rectangles indicate something that frequently happens. The values can only be added by column and in all cases always account for the 100% of the number of power variations.

The rectangle columns in the charts give a trend on how a particular component contributes in all power variations. For instance, in chart (a) the L1 power contributions never correspond to more than 50% of a total power variation of -1000mW. It can be observed that the darkest rectangle for this component is placed between the 20% and 30% bounds. According to the rectangle's color, in 50% of all power variations of -1000mW, the L1 decreases its power consumption for something in the range of [300-200] mW.

In general, the charts pretend to detect if there are cases where just one component is the sole responsible for a power variation, or if it acts as one significant contributor. One initial observation is that only the FE and the FSB present some cases where they justify 80% or more of the total power variation (charts (a), (b), (c) and (f)). But these cases do not account never for more than 10% of the total number of power variations. For the rest of cases we need to define a threshold to identify significant contributors. We have set this threshold to a 40% of contribution because is the threshold in which most components change their trend. Then, the upper half part of every chart is telling us what components have significant intensity on the rectangles in this area. For variations of less than 500mW (charts (c) and (d), only the FE, and the FSB present cases where they appear as the main contributors. These cases account approximately for 15% of the power variations. For variations between 1000mW and 500mW (charts (b) and (e)) similar results are observed. For variations of 1000mW (charts (a) and (f)), the main contributors are the FE, the FSB and the L2. The rest of components are usually contributing for less than the 30% of the power variations. Even in the case of the execution units (INT and FP components), their contribution is always below a 10%. Only the BPU exposes in some cases (chart (a)) contributions that go above 40%.

2) Per component power analysis: Figure 1 shows the power evolution for the 473.astar application during two different phases. The first one corresponds to the case where no power variation is observed. However, the components expose significant variations that when added altogether result in a constant power curve. Notice how the FSB makes a considerable increment in power consumption, going from 200 mW to 1100mW. The L1 presents a decrement in power consumption, going down from 700mW to 500mW. All other components show marginal decrements that altogether hide the increment of the FSB component. This example is significant due to two reasons. First, the power consumption stays constant, and any power aware policy just based on power variations would not detect the FSB increment. Second, in this situations there is a potential opportunity for power saving. We have checked that FE decrements its activity due to the fact that the program has entered in a phase where data is missing in the cache hierarchy and it has to be brought from main memory. It is a good point to evaluate whether or not it is worth to activate DFVS-based power policy. It is widely accepted that in a situation where memory is the element that limits the actual processor's throughput, decreasing the operating frequency might result in power savings without incurring in performance degradation.

The second situation corresponds to the case when the program enters in a phase where power is incremented for about 700mW, and afterward the power returns to its original value. The trend in each component is quite different. At the initial power increment, the BPU, the FE, the L1 and the L2 expose increments between 150mW and 200mW, but at the power fall, only the L2 decrease its power consumption. Most of the remaining of the components increase their power consumption. This case shows an example of two points of power

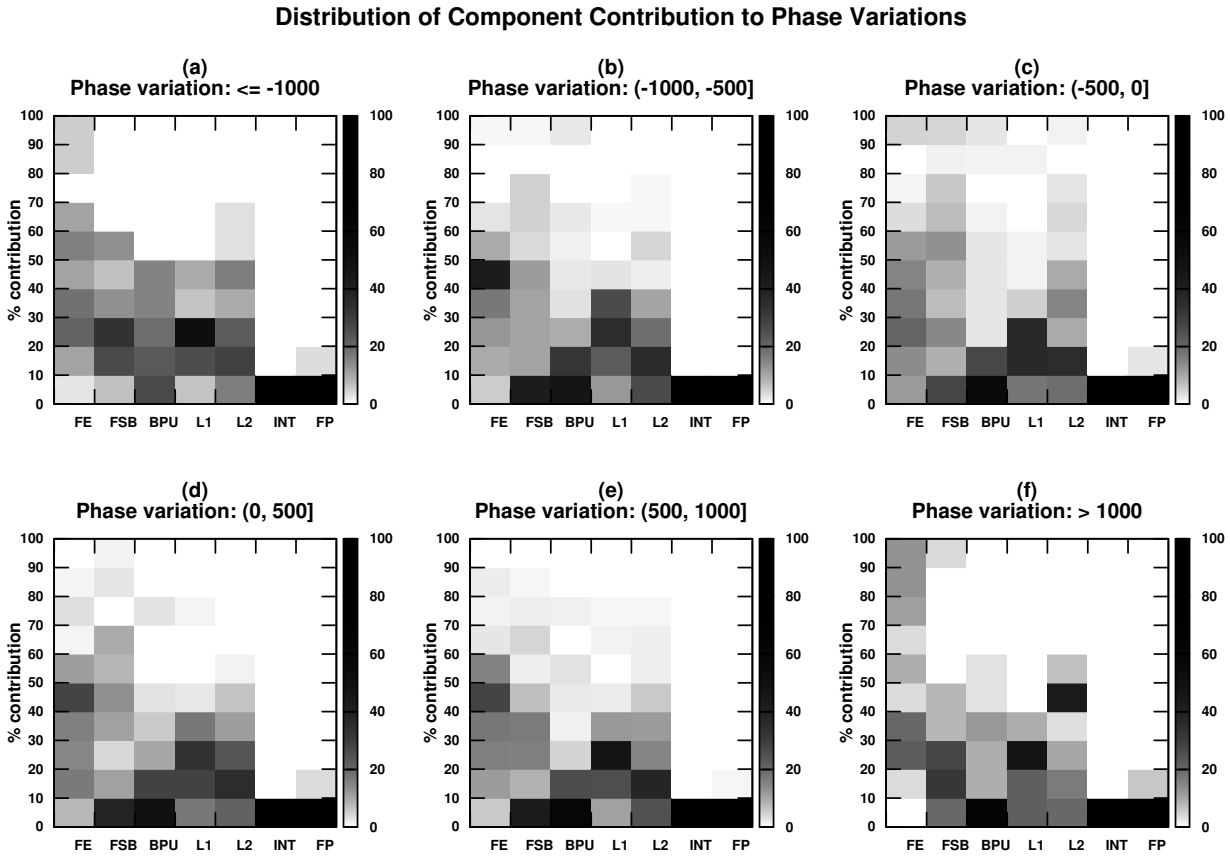


Figure 8. Contribution per components to phase variation.

variations where just looking at power variations is not enough to decide about applying any power saving policy. For instance, DFVS would not have much space to be applied here, but if just power variations are taken in to account with no understanding what caused them, it might happen that we decide to activate a policy that ends up with both performance and power efficiency degradation.

3) Power consumption per application and time period: One immediate application of our power model is that of measuring the impact on both performance and power consumption caused by the interaction of the two cores when two different applications execute, one on each core. We define two running configurations, namely the shared configuration where applications run in one core while another application runs in the other core, and the isolated configuration when an application runs in one core while the other is deactivated. We expect the applications to run slower under the shared configuration than in the isolated configuration.

For every tested application, we collect the number of executed instructions in a time slice of ten minutes running under the shared configuration. Then we compute the time it took on the isolated configuration to execute the same number of instructions. This defines a time window where both configurations can be compared in terms of average power consumption.

Table IX shows the slowdown factor for every pair of applications that we have tested, after computing the time window previously mentioned. Table X shows the interference between the pair of applications in terms of average power consumption within the time window. In general, we observe that the slowdown factor range between 0.97 and 0.77, depending on the applications. The power interferences show that on average, most of the applications require less power, but require more time to execute the same amount of work. We observe that the power interferences range between 0.76 and 0.87. Notice that in order to generate this kind of data, the power model has a main role, as it allows to generate power estimates for every core, separately. Thus, it allows us to attach the power consumption to a particular application, moreover to a particular time period within

Benchmark	400.perlbench	462.libquantum	473.astar	436.cactusADM	453.povray	454.calculix
400.perlbench	-	0.89	0.91	0.91	0.94	0.94
462.libquantum	3.14	-	3.02	3.09	3.17	3.17
473.astar	0.91	0.77	-	0.89	0.92	0.92
436.cactusADM	0.98	0.93	0.97	-	1	0.99
453.povray	0.95	0.95	0.95	0.95	-	0.95
454.calculix	0.97	0.96	0.97	0.97	0.97	-

Table IX
NORMALIZED EXECUTED INSTRUCTIONS (WITH RESPECT TO STANDALONE RUN).10 MINUTES EXECUTION

Benchmark	400.perlbench	462.libquantum	473.astar	436.cactusADM	453.povray	454.calculix
400.perlbench	-	0.87	0.85	0.85	0.85	0.85
462.libquantum	1.11	-	1.1	1.11	1.11	1.11
473.astar	0.78	0.79	-	0.77	0.78	0.78
436.cactusADM	0.76	0.81	0.76	-	0.76	0.76
453.povray	0.83	0.83	0.83	0.83	-	0.83
454.calculix	0.81	0.82	0.8	0.8	0.81	-

Table X
NORMALIZED AVERAGE POWER (WITH RESPECT TO THE STANDALONE, FOR THE SAME NUMBER OF INSTRUCTIONS RETIRED)

an application's execution.

In conclusion, we have validated our power model for both the average error and the model responsiveness. Besides, we have shown two important applicability examples that clearly expose that the models based on hardware counters and build in a per-component basis are suitable candidates for implementing power aware policies or energy accounting for applications.

V. RELATED WORK

In general, previous works on power modeling focus in two main aspects: power model generation and its usage to guide power aware policies. For instance, Tao Li et al. [19] study the power consumption at the OS level. For this purpose, an IPC based model is built based on the broadly accepted observation that there is a linear relation between IPC and power consumption. In [3], F. Bellosa implements an OS power aware policy based on data collected at runtime through the PMCs. In this work, overall power consumption is being modeled using a reduced set of PMCs. K. Singh et al. [30] describe a methodology based on a set of microbenchmarks that stress particular components of the processor architecture being modeled. As a result, a model suitable for any workload -generic-. Our main difference with all these approaches is that they use the PMCs –one or several– to predict the power consumption of the whole processor, treating it as a black-box.

A. Miyoshi et al. [21] introduce the concept of critical power slope. This new metric is used to determine the effectiveness of activating power saving techniques on scheduling points. A contribution of this work is that depending on the workload characteristics lowering the processor frequency results in power savings. Our proposal agrees with this result, and points out the fact that power models have to be validated at inflection points of power in order to be used by power aware policies.

The works of R.Joseph, C.Isci, G.Contreras and M.Martonosi [16, 14, 9] present many similarities to our proposal because we follow the same objective: generate an accurate power model capable to breakdown the power consumption. The most similar work to our is [14]. In that work, the Pentium IV architecture is decomposed in microarchitectural components in a similar fashion. But the model generation methodology differs: they use component area as the main heuristic to derive each component power consumption. Some microbenchmarks are also used to guide the manual tuning of the model. In contrast, we derive the marginal effect of each component on power from data gathered using a specifically designed set of microbenchmarks. Thus, we avoid their test-and-evaluate manual tuning and the need of the floorplan information. In [9] they also use statistical linear regression to generate the model but they validate it against a similar set of applications. In overall, these works are able to predict power consumption in the same order of accuracy than the ones that use less PMCs but, they can breakdown the power consumption. These works proved that the usage of sampling techniques to read a bigger set of PMCs does not incur in inaccuracies. In [15], C.Isci and M.Martonosi present a study comparing power phase classification techniques based on PMCs and control flow information, concluding that PMCs detect more power phases.

There is not doubt about the applicability of PMC-based power models since they provide accurate insight into processor power consumption and that they can be

used to breakdown the power consumption of a given platform. While this work is based on the same idea of using the PMCs for power estimation, there are some key characteristics that differentiate our proposal from all previous research:

- We present a new methodology to model current multicore architectures. The key difference is that our methodology is strictly based on the appropriate definition of power components and the specifically designed microbenchmarks that isolate and decouple each component activity. This allows us to produce models that are accurate, responsive and decomposable.
- Besides the validation of the model based on average error, we *validate and evaluate the responsiveness of the model*, comparing it against other PMC-based power models.
- We present a set of power models for the *Intel® Core™ 2 processor*, a high performance processor designed with power efficiency as a main design constraint, for difference DVFS configurations.

VI. Conclusion

In this paper we present a novel systematic power modeling methodology based on performance monitoring counters (PMCs) to derive per component power breakdowns on current multicore architectures. Moreover, besides validating the accuracy of power estimates, we validate the models by evaluating their *responsiveness*, their capacity to detect power variations. We generated a set of models for the Intel® Core™ 2 Duo architecture, modeling one or two cores for different DVFS configurations. The models are validated using empirical measurements of long executions of the SPECcpu2006 benchmarks, and despite the energy-efficient characteristics and the complexity of the modeled architecture, the models shown fairly good accuracy and responsiveness. Concretely, the accuracy validation shows average errors between [1.89-6]%, and the responsiveness validation when operating at maximum frequency shows an overall 83.31% accuracy on phase change predictions and almost 100% accuracy for changes bigger than 0.5 watts. As a result, we show that the model produced is sensible to variations as small as 0.25 watts, which only represent a 2.24% variation of the average power consumption of the SPECcpu2006 suite. Moreover, the comparison performed against models built using existing methods shows that our methodology produces more accurate and responsive power models. In overall, this work addresses the problem of the lack of decomposable power models for current architectures.

References

[1] Thinkpad SMAPI kernel module version 0.40. http://tpctl.sourceforge.net/ . (page 5).

[2] Perfmon2. http://perfmon2.sourceforge.net/ . (page 5).

[3] F. Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 37–42, New York, NY, USA, 2000. ACM. (pages 1, 4, 17).

[4] A. Bhattacharjee and M. Martonosi. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 290–301, New York, NY, USA, 2009. ACM. (page 1).

[5] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. *Performance Analysis of Systems and Software, IEEE International Symmposium on*, 0:158–168, 2007. (page 1).

[6] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 275–280, New York, NY, USA, 2005. ACM. (page 1).

[7] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 338–342, New York, NY, USA, 2003. ACM. (page 1).

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM. (page 1).

[9] G. Contreras and M. Martonosi. Power prediction for Ontel XScale®processors using performance monitoring unit events. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 221–226, New York, NY, USA, 2005. ACM. (page 17).

[10] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 78–88, Washington, DC, USA, 2006. IEEE Computer Society. (page 2).

[11] N. Draper and H. Smith. *Applied Regression*

Analysis. Wiley, New York, NY, second edition, 1981. (page 7).

[12] V. George, S. Jahagirdar, C. Tong, K. Smits, S. Damaraju, S. Siers, V. Naydenov, T. Khondker, S. Sarkar, and P. Singh. Penryn: 45-nm next generation Intel® Core™ 2 processor. In *ASSCC'07 IEEE Asian Solid-State Circuits Conference*, 2007. (pages 3, 5).

[13] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006. (page 6).

[14] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93, Washington, DC, USA, 2003. IEEE Computer Society. (pages 1, 2, 4, 6, 7, 17).

[15] C. Isci and M. Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA-12*. Princeton University, February 2006. (pages 10, 17).

[16] R. Joseph and M. Martonosi. Run-time power estimation in high performance microprocessors. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2001. ACM. (pages 6, 17).

[17] Y.-M. Kuo, S.-H. Weng, and S.-C. Chang. A novel sequential circuit optimization with clock gating logic. In *ICCAD '08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 230–233, Piscataway, NJ, USA, 2008. IEEE Press. (page 2).

[18] K.-J. Lee and K. Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11*, page 232.1, Washington, DC, USA, 2005. IEEE Computer Society. (page 1).

[19] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, 2003. (pages 1, 17).

[20] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40(4):403–414, 2006. (page 1).

[21] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 35–44, New York, NY, USA, 2002. ACM. (page 17).

[22] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001. (page 1).

[23] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. R. Stan. Power issues related to branch prediction. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, pages 233–, Washington, DC, USA, 2002. IEEE Computer Society. (page 4).

[24] M. J. Pazzani and S. D. Bay. The independent sign bias: Gaining insight from multiple linear regression. In *In Proceedings of the Twenty First Annual Conference of the Cognitive Science Society*, pages 525–530, 1999. (pages 5, 7).

[25] A. Phansalkar, A. Joshi, and L. K. John. Analysis of redundancy and application balance in the spec cpu2006 benchmark suite. *SIGARCH Comput. Archit. News*, 35(2):412–423, 2007. (page 12).

[26] A. Phansalkar, A. Joshi, and L. K. John. Subsetting the spec cpu2006 benchmark suite. *SIGARCH Comput. Archit. News*, 35(1):69–76, 2007. (page 12).

[27] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0. (page 8).

[28] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 66–77, Washington, DC, USA, 2006. IEEE Computer Society. (page 1).

[29] SBSIF. SMART specifiaction rev.1.1 Dec 11, 1998. [Online] Available: http://sbs-forum.org. (page 5).

[30] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, 2008. (pages 1, 4, 6, 7, 8, 17).

[31] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2003. ACM. (page 1).

[32] W. Wu, L. Jin, J. Yang, P. Liu, and S. X.-D. Tan. A systematic method for functional unit power estimation in microprocessors. In *Proceedings of the 43rd annual Design Automation Conference*, pages 554–557, New York, NY, USA, 2006. ACM. (pages 5, 6, 7, 8).