Nikov, K., & Nunez-Yanez, J. (2020). Intra and inter-core power modelling for single-ISA heterogeneous processors. *International Journal of Embedded Systems*, 12(3), 324-340.
<https://doi.org/10.1504/IJES.2020.10021023>,
<https://doi.org/10.1504/IJES.2020.107046>

Peer reviewed version

Link to published version (if available):

[10.1504/IJES.2020.10021023](https://doi.org/10.1504/IJES.2020.10021023)
[10.1504/IJES.2020.107046](https://doi.org/10.1504/IJES.2020.107046)

[Link to publication record in Explore Bristol Research](#)
PDF-document

This is the author accepted manuscript (AAM). The final published version (version of record) is available online via INDERSCIENCE Publishers at <https://www.inderscience.com/info/inarticle.php?artid=107046> . Please refer to any applicable terms of use of the publisher.

University of Bristol - Explore Bristol Research

General rights

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available:
<http://www.bristol.ac.uk/red/research-policy/pure/user-guides/ebr-terms/>

Intra and Inter-Core Power Modelling for Single-ISA Heterogeneous Processors

Krastin Nikov

Department of Electrical and Electronic Engineering,
University of Bristol,
Bristol, UK
E-mail: kris.nikov@bristol.ac.uk

Jose Nunez-Yanez*

Department of Electrical and Electronic Engineering,
University of Bristol,
Bristol, UK
E-mail: j.l.nunez-yanez@bristol.ac.uk

*Corresponding author

Abstract: This research presents a systematic methodology for producing accurate power models for single Instruction Set Architecture (ISA) heterogeneous processors. We use the hardware event counters from the processor Performance Monitoring Unit (PMU) to accurately capture the CPU states and Ordinary Least Squares (OLS), assisted by automated event selection algorithms, to compute the power models. Several estimators for single-thread and multi-thread benchmarks are proposed capable of performing power predictions across different frequency levels for one processor as well as between the heterogeneous processors with less than 3% error. The models are compared to related work showing significant improvement in accuracy and good computational efficiency which makes them suitable for run-time deployment.

Keywords: big.LITTLE System-on-Chip, Linear Regression, Ordinary Least Squares, Hardware Performance Events, Automated Event Selection

Reference to this paper should be made as follows: Nikov, K. and Nunez-Yanez, J. (xxxx) 'Intra and Inter-Core Power Modelling for Single-ISA Heterogeneous Processors', *International Journal of Embedded Systems*, Vol. x, No. x, pp.xxx-xxx.

Biographical notes: Kris Nikov received his Ph.D. in Electrical and Electronic Engineering at the University of Bristol in 2018 with a thesis on Power Modelling and Analysis on Heterogeneous Embedded Systems. He is currently working as a Research Associate at the University of Bristol Department of Electrical and Electronic Engineering on the topic of ENergy Efficient Adaptive Computing with multi-grain heterogeneous architectures (ENEAC).

Jose Nunez-Yanez is a Reader (associate professor) in adaptive and energy efficient computing at the University of Bristol and member of the microelectronics group. He holds a PhD in hardware-based parallel data compression from the University of Loughborough, UK, with three patents awarded on the topic of high-speed parallel data compression. His main area of expertise is in the design of reconfigurable architectures for signal processing with a focus on run-time adaptation, parallelism and energy-efficiency. Previous to joining Bristol he was a Marie Curie research fellow at ST Microelectronics, Milan, Italy working on the automatic design of accelerators for video processing and a Royal Society research fellow at ARM Ltd, Cambridge, UK working on high-level modelling of the energy consumption of heterogeneous many-core systems.

This paper is a continuation of the work described in 'Evaluation of hybrid run-time power models for the ARM big. Little architecture.' published in IEEE/IFIP 13th International Conference on Embedded and Ubiquitous Computing, EUC 2015, pages 205-210. 2015

1 Introduction

The slowdown of Moore's law and the rapid increase in complexity of heterogeneous information processing systems [1] has resulted in the use of various techniques in order to satisfy consumer demand for performance. Research has

shown that multi-core heterogeneous systems seem to be the way forward to address the increase in energy usage in proportion to performance [2]. An example of commercially successful Heterogeneous CPUs are the *big.LITTLE* SoC [3] developed by ARM Ltd. These multicores were first announced in 2011 [4] and continuous to gain popularity

with a new generation called *DynamIQ* recently announced by ARM [5]. They combine *high-performance* and *energy efficient* processing cores in a configurable combination. The two processor types use the same ISA so they are able to execute the same compiled code. The aim is to achieve better power efficiency, while maintaining good levels of performance, by using the heterogeneity of the system to direct tasks towards the most suitable processor type. Due to the increased complexity of such systems and their broader energy usage variation, extra attention needs to be paid to the software side and particularly the energy management policies. This research investigates a power modelling approach suitable for heterogeneous processors with a common ISA. We have used our methodology to compute very accurate run-time models for the *big.LITTLE* system, while keeping it generic enough so that it can be adapted to other architectures and a different set of system events. In order to validate our approach we compare our models to other published work and show significantly reduced model error. Our research offers some key insights into predicting the behaviour of modern heterogeneous systems and can serve as a stepping stone for further advancements in intelligent advanced power-aware scheduling.

The key contributions in this article are as follows:

1. *Flexible and reconfigurable methodology with automatic event selection* - The methodology described in our previous work in Nikov et al. [6] is further developed and improved. Several different automated algorithms and optimisation criteria have been investigated and are shown to greatly outperform traditional *intuitive* methods for PMU event selection. Specific system tools are used to control the data collection process more closely and achieve less than 1% power and performance experiment overhead, resulting in a significant reduction in model error compared to other published methodologies.
2. *Intra and Inter-Core Power Models* - A specific technique is developed to scale PMU events, which allows the computation of average power between frequency levels on the same processor using the same data. An extension of this method allows the use of runtime hardware counter information to predict average power between any two frequency levels of the two processing clusters on the *LITTLE* platform. These types of models are named *intra* and *inter-core* respectively and show very high accuracy. The ability to use the PMU events of one processing cluster to predict the average power of another cluster is a feature of our methodology that we have not encountered elsewhere in literature and allows full characterisation of the power profile of the heterogeneous platform.
3. *Open-source methodology* - To facilitate further research in the field and model comparison, reuse and verification we have made the entire methodology open-source at the following github repositories [7] [8]

The rest of this article is organised as follows. Section 2 gives a comprehensive overview of related work in the area of power modelling and provides more information about the models used for reference and comparison. Section 3 details the development platform and Section 4 the benchmarks used in this research. Section 5 introduces the data collection methodology and the techniques to reduce experiment variability and overhead. Section 6 explains the model calculation method and the automatic event selection algorithms and optimisation criteria. Section 7 describes the specific model features for *intra* and *inter-core* modelling. Section 8 contains the main experiment results. The final Section 9 concludes this article, highlighting the achieved objectives, lists unresolved problems and other areas for future work.

2 Related Work

The optimisation of power and energy in a computing system can be done at different levels such as the cost of moving information [9] or processing information [10]. In all these cases it is very useful to be able to predict the impact that changes will have in energy/power consumption using some high-level activity measures without direct power measurements that could be unfeasible because the silicon is not yet available or no access to the power rails is possible. These predictions can be then be used by a scheduling algorithm to optimize overall energy requirements [11]. A very successful way to observe these fine changes in power and energy is using hardware system information available from the PMU on a CPU. Historically PMU's have been used to estimate performance, but lots of researchers have been successful in also estimating CPU/system power consumption using PMU hardware events. The main benefits of this approach is that PMU support is widespread so a good solution could be easily incorporated into existing systems. The work of Nunez-Yanez et al. [12] makes a case that system-level modelling is better than lower level modelling. The authors use a large number of PMU events collected with a simulator on an ARM Cortex-A9, to train a linear model using mathematical regression. Instead of using micro benchmarks they use cBench as a workload to stress the entire system as a whole and report an average of 5% estimation error. Similarly, Singh et al. [13] have developed a power model based on 4 PMU events on AMD Phenom 9500 CPU. They use micro benchmarks to train the model and events are collected every second. The model is computed using piece-wise linear regression with least squares estimator and is tested on NAS, SPEC-OMP, and SPEC 2006 with median errors of 5.8%, 3.9%, and 7.2% respectively. They further this work by using the model to guide a single-thread scheduler, which suspends processes to ensure a power budget. This shows how power models can be used effectively in dynamic schedulers to help improve the power efficiency of systems. In contrast our *intra* and *inter-core* models can be used for more advanced *never-idle* DVFS policies, which are shown to be more suitable for embedded and mobile processors [14]. Walker et al. [15] presents two different methodologies for 2

development platforms. They develop a model using 4 PMU events for a system featuring the ARM Cortex-A8. With that approach they report 1.9% average error while predicting power consumption using MiBench [16] as a workload. They also present a CPU frequency and utilisation based model for a *big.LITTLE* platform, which did not have the PMU enabled. They obtain information about CPU time spent in idle using information available from the Linux kernel running on the device. Tested on the same workload as the PMU model, the CPU frequency and idle time model achieves 10.4% and 8.5% error for the ARM Cortex-A7 and ARM Cortex-A15 respectively. We also explore using CPU State information alongside PMU events for accurate power modelling in our previous work [6]. There we used an intuitive approach to select the PMU events, based on observations from Nunez-Yanez et al. [12]. In this article we have focused our efforts into developing models purely from PMU events, since obtaining the CPU state information introduced large overhead, which we could not overcome. Despite this, our new methodology is capable of producing significantly more accurate models even without CPU state information, as evidenced in section 8.

In addition to our previous work, we use other published research to validate our model, namely the work of Pricopi et al. [17] Walker et al. [18] Rodrigues et al. [19] and Rethinagiri et al. [20]. The choice which models to compare against was guided by the fact that they are all PMU based models developed either on *big.LITTLE* or a similar embedded SoC. This made them reproducible on our development platform. Pricopi et al. [17] develop complex models for predicting performance by predicting the CPI stack. As part of their work, however, they have also built a mechanistic model for the Cortex-A15, which utilises CPU design experience and a deeper understanding of the architecture to select the list of PMU events used. They train the model with an average error of 2.6% when trained and tested on SPEC CPU2000 and SPEC CPU2006 benchmark suites. They have not produced a model for the Cortex-A7 on the justification that the processor does not exhibit much variation in its power dissipation and can be approximated by a single number. In our research we refute this assumption and show that the Cortex-A7 also exhibits significant power variation and dedicated power models are required to capture its behaviour. Their work is done on an experimental platform and on a single CPU frequency, hence the simplified power profile. Nevertheless this is one of the earliest PMU based models available for the ARMv7 architecture and provides great insight into the use of PMU events for power modelling. Walker et al. [18] have continued their work in [15] and have developed a model for *big.LITTLE* on the same platform, the ODROID-XU3. They use a simple SML method to traverse the list of available PMU events, but their methodology uses the SPEC 2006 workload and does not utilise some of our approaches to minimise overhead and event variability. They have developed individual models for the Cortex-A15 and Cortex-A7, though only the events list for the former is published. They have done a very thorough research into the statistical and mathematical drawbacks of OLS and presented a method for increasing model accuracy and

flexibility by addressing the problem of heteroskedasticity in power modelling reporting 2.8% and 3.8% average error for the Cortex-A15 and Cortex-A7. We manage to achieve a more effective strategy to ensure model accuracy and stability with our *per-frequency* level models combined with an extended analysis of different model event selection search algorithms. Rodrigues et al. [19] developed a model, designed to offer the most accuracy with a minimal set of events for both a *high-performance* and a *low-power* execution unit, represented by an unnamed Intel Nehalem and Atom processors in a simulation environment. We have successfully managed to implement the model for the *big.LITTLE* multi-core SoC. This still is an interesting comparison case, since the authors also have a comprehensive analysis of PMU events. They have compared several models, utilising different numbers of PMU events, in their research. The models are trained and validate on an extensive suit of benchmarks, consisting of SPEC 2000, MiBench and mediabench [21]. The final reported model error is less than 5% for both CPU types for a single-core set-up. The final work used in our comparison is Rethinagiri et al. [20]. They present a power-estimation tool for embedded systems, incorporating physical platform information and PMU events to predict power consumption, tested for ARM9, ARM Cortex-A8 and ARM Cortex-A9. They base their approach around accurate run-time system-level power models and use micro benchmarks to obtain cache information and *intuitively* selected PMU events and train a linear model using OLS regression. Their model has a small set of regressands, since they use just CPU frequency and 4 PMU events. Despite this they report around 4% for all three CPUs on a custom microbenchmark test set. The interesting thing about this model is the heavy emphasis on cache events. In our comparison, this model performs poorly, precisely due to the high variability of cache events in complex workloads. We show that by analysing the events with high variation and removing them from the event selection process, we are able to achieve much more stable and accurate models.

3 Platform description

Early research involving *big.LITTLE* SoCs involved using simulators due to the unavailability of suitable hardware platforms. Since then several companies have come up with various development boards for *big.LITTLE*. The ideal target for our approach to power modelling is a system which has both PMU events as well as sensors to collect the power of the desired component to be modelled. Our platform of choice is the Hardkernel ODROID-XU3 [22]. We use it for the majority of our experiments and we develop the methodology on it.

Our work is done on the first generation of the *big.LITTLE* platform so our models are built for the ARM Cortex-A15 and Cortex-A7 processors. A key feature of the SoC is the Cache Coherent Interconnect(CCI), which enables quick task migration between the two CPU islands. For that purpose ARM has developed patches for Linux and Android OSs, which support a custom scheduler for *big.LITTLE* [23]. The

scheduler is a natural extension of DVFS, which allows tasks to be migrated from one CPU cluster to the other. Thanks to the Cache Coherent Interconnect the overhead of migrating the task is kept low. The scheduler has 3 operating modes, depending on the particular implementation — Cluster Migration(CM), In-Kernel Switching(ICS) and Global Task Scheduling(GTS). The most sophisticated implementation is GTS since it allows migration between any two CPU cores, even on the same processing cluster. This also enables the full capabilities of the system with the ability to use all cores at the same time. GTS relies on migration thresholds to decide when it is time to migrate the task to a performance or a power-efficient CPU. A first scheduled task starts at the power-efficient cluster and if the CPU utilisation gets above a certain threshold the scheduler moves the task to the performance cluster. If the utilisation drops then it moves back to the power-efficient CPU. The threshold levels are dependent on implementation, but in all cases are chosen to be apart enough to prevent overzealous switching. Currently no existing solution involves taking the processor power usage into account. We believe this is a crucial step in improving the long-term viability of this technology, which is why our research is focused on power estimation. We design our models with the capability to be integrated into a power-aware scheduling solution. The platform selection is also motivated by the presence of four Texas Instruments INA231 sensors measuring the A15, A7, RAM and GPU power, current and voltage. This is a key feature of the platform, since it enables accurate power measurements to be made, which we need in order to train and validate the models. There are more modern solutions available with SoCs implementing ARMv8, but they lack the power sensors essential to this work. Market analysis done by Unity3d[24], which is popular mobile gaming engine, indicates that even in 2017, devices build on the ARMv7 architecture still dominate the mobile market at more than 90%. This means that the need for advancements in energy management for ARMv7-based devices is still important.

The platform was set up with a minimal Ubuntu 12.04 running the latest kernel available from the board support team. The platform OS is chosen to be small enough to avoid a big OS overhead, but with enough features to provide easy software development. Another key feature of the ODROID XU3 is that it has a broad DVFS range with the Cortex-A15 having 19 available frequency levels ranging from 0.2-2 GHz and 5 corresponding voltage levels and the Cortex-A7 with 13 available frequency levels from 0.2-1.4 GHz with 5 available voltage levels. The presence of the eMMC card slot in addition to the standard microSD slot is important since in our experiments we noticed a significant variation when comparing results obtained using an eMMC card with a microSD card. The sample data indicated that the eMMC is a much more stable card with consistent performance and variability below 5% for both CPU power as well as runtime for the two processor types in *big.LITTLE*. We use exclusively the eMMC card in our experiments.

4 Benchmark selection

The ideal workloads should be exhaustive benchmarks with diverse behaviour in order to capture different scenarios and long runtime. We explored a few open-source options initially, ranging from simple performance benchmarks like Dhrystone [25], Whetstone [26], LINPACK [27] to complex test suites like MEVBench [28], Parboil [29], Rodinia [30], BBench [31] and the benchmarks available through the phoronix-test-suite [32]. Eventually cBench [33] was selected, because it consists of a large set of smaller benchmarks aimed to represent real-life workloads and it has long runtimes (to ensure we get enough samples from the energy monitors). cBench is also single-threaded so it is ideal for developing a single-thread model, which we believed was a first and necessary step in our research. We use 30 microbenchmarks from the cBench suite on the ODROID-XU3.

For the multi-thread case we considered NPB [34] as well, but we decided to use PARSEC [35], since it is more modern, well established in the research community and also consists of several smaller benchmarks. It is highly configurable and you can set the number of threads you want for most of the workloads in the suite. This makes it ideal for our 8 core system, since it allows us to explore all the possible multi-core configurations of the system. We consider the 1 Core, 2 Cores, 3 Cores and 4 Cores cases separately and collect data for them individually. When building the power models we concatenate data from all 4 cases into one big set and use that in our analysis. Table 3 shows the benchmarks from each suite, that were used in our experiments. Further details can be found in Section 6.1.

Figure 1 gives details about the benchmark energy consumption at each frequency level for the Cortex-A15 and Cortex-A7 processors on the ODROID XU3 board. For the PARSEC highlight we use data of the workload running on all 4 Cores per cluster. The total runtimes for one execution of the benchmark suites on *big.LITTLE* at the highest CPU frequencies are, on average, 480s for the Cortex-A15 and 720s for the Cortex-A7 for cBench and 95s for the Cortex-A15 and 230s for the Cortex-A7 for PARSEC using all 4 available cores per cluster. This gives a lower-bound of 190 samples to be used in model generation and validation, which we prove in our own experiments to be enough to ensure model accuracy and stability.

The convex curve show that the lowest energy point is not simply the smallest voltage/frequency level and therefore is not always predictable without knowing the workload. This observation coincides with DeVogeleer et al. [36], who observed a similar relationship on the Samsung Galaxy S2 running a part of the Fast Fourier Transform algorithm as workload. This further supports our claim that accurate power models could be extremely useful for dynamic energy management, since such behaviour is very difficult to derive empirically.

Figure 1: ODROID XU3 Workload E/F Curve for cBench (left) and PARSEC (right)

Figure 2: Example of Data Sampling and Model Validation on the ODROID XU3

5 Data Collection

This section details how we collect the experimental data from the ODROID XU3 development board and prepare it for later processing by the power model generation algorithms.

Data collection consists of 3 key components - system configuration, workload selection and finally program control. We start by setting up the platform for the experiment by loading the eMMC memory card with the OS and custom kernel patch. Afterwards we install the methodology tools that we use to control program execution and minimise overhead - *cset* and *cpufrequtils*. Downloading and compiling the workloads - cBench [33] for the single-thread case and PARSEC 3.0 [35] for the multi-thread. After the experiment data has been collected, we synchronise the different sensor and event samples and supporting scripts [8].

The PMU available in the Cortex-A15 provides six configurable registers with an additional seventh reserved just for CPU cycles. In contrast the PMU available for the Cortex-A7 only has four configurable registers, but still has a dedicated one for CPU cycles. We are able to collect a

total of 67 different hardware events for the Cortex-A15 and 42 for the Cortex-A7. Interestingly only 23 are common to both PMU's. A full list of the PMU events is available through the Technical Reference Manual available for both processors [37].

In order to communicate with the PMU without a debugger and configure the performance counters, we use the linux tool *perf*. *perf* allows communication with the linux kernel from userspace and can profile the entire system.

We use several tools in the Linux ecosystem to help control the execution of the workflow and to read the PMU events and on-board power sensors. The two four-core CPU clusters can be configured separately, which allows for a lot of options for the system set-up. We do this in three steps. First, we use */sys/devices/* to identify and *hotplug* the CPUs we do not want in our experiment. Second, we also use *cpufreq-set* from the *cpufrequtils* package to set the CPU cluster frequency. Finally we use *cset* to assign the workload application to the appropriate cores and isolate all the data collection scripts and OS threads on the opposite CPU cluster.

Figure 3: Methodology Steps

Core Type	Runtime Single/multi-thread	Avg. Power
C-A15	0.16/1.18	0.27/0.38
C-A7	0.12/0.95	0.40/0.62

Table 1: Experiment Overhead[%]

Core Type	Min. Up Single/multi-thread	Min. Down
C-A15	9.40/9.21	8.59/8.43
C-A7	15.61/14.63	13.50/12.77

Table 2: Target Accuracy[%]

Workload	Train Set	Test Set
cBench	telecom_CRC32	consumer_jpeg_d
	consumer_tiffdither	security_blowfish_e
	telecom_gsm	security_pgp_d
	bzip2d	office_ghostscript
	consumer_tiffmedian	network_dijkstra
	consumer_jpeg_c	security_blowfish_d
	office_stringsearch1	automotive_susan_e
	office_ispell	automotive_qsort1
	automotive_susan_s	automotive_bitcount
	security_pgp_e	security_rijndael_e
	telecom_adpcm_d	telecom_adpcm_c
	automotive_susan_c	bzip2e
	security_sha	network_patricia
	security_rijndael_d	office_rsynth
	consumer_tiff2rgba	consumer_tiff2bw
PARSEC	parsec.facesim	parsec.dedup
	splash2x.radiosity	parsec.freqmine
	splash2x.raytrace	parsec.streamcluster
	splash2x.water_nsquared	splash2x.barnes
		splash2x.fmm

Table 3: Workload Splits

This prevents the collection to interfere with the experiment and minimises overhead.

The data collection scripts read the on-board power sensors data and use the Linux *perf* tool to communicate with the PMU and collect the events. This is all done in real-time, in parallel to the workload execution at a 0.5s interval. The interval is imposed by the minimal refresh rate of the power sensors and since all measurements need to be synchronised, the PMU event collection is also done at the same interval. An example of the power sampling and modelling is shown in Figure 2. We present the first 20 samples collected from a single run of *parsec.dedup* at maximum CPU frequency on all 4 cluster cores for both processor types. We also show that for every power sample, we also sample the PMU events at that point and can make a prediction on the instantaneous power. It is because we use such fine granularity in our *per-frequency* models that we claim they can be used in real-time to aid DVFS.

Our final methodology explores the entire set of PMU events available. The number of events that can be collected in parallel from the PMU is limited. With 67 total events available for the Cortex-A15 and 42 for the Cortex-A7, this means that we need multiple runs and collections in order to capture all events for analysis. In order to facilitate data analysis we use precise timestamps for each measurement so that they can be concatenated later on. A diagram of the full set-up is presented in 3.

This set-up is necessary in order to minimise experiment interference and variability. Table 1 shows the low overhead of the off-cluster data collection. We can see that our methodology has very minimal impact on the measurements -

less than 1% extra resources used. This is an important reason for the high accuracy of our models, as shown in Section 8.

We also perform one additional step - namely removing any events which have high variability between platform runs and any events that are very specific and do not get triggered by the workload. This ensures we have a consistent and stable list of events, which improves model stability as well. After this operation we are left with 55 and 30 usable events for the Cortex-A15 and the Cortex-A7, respectively.

6 Methodology

This section describes the steps involved after the *Synchronize & Concatenate Collected Data* bloc in Figure 3. Subsection 6.1 explains the linear regression algorithm used to calculate the model coefficients from the training data and Subsection 6.2 details our custom model event selection and optimisation procedures.

6.1 Linear Regression Method

After data collection we perform the mathematical analysis on the results off-line, on a supporting machine. First, we split the workload into two sets - one for training and one for model testing. For cBench we have an even split of 15 microbenchmarks for each set, while for PARSEC we have a 4 to 5 split for training and testing respectively. Details about the individual microbenchmarks selected for each set are given in Table 3. We retain and use the same split for the majority of the experiments involving PMU event selection and when comparing the models to other

related work. In addition to the randomised split we also validate the best model performance using a *n-fold cross-validation* [38] [39] to ensure the statistical rigorousness of the model. These results are presented in detail in section 8. Afterwards we compute the model using the *octave* mathematical environment. We use Ordinary Least Squares(OLS) [40], a well-know linear regression algorithm, to identify the events that best predict average power from the train set. The mathematical expression is shown in equation 1.

$$\alpha = (X^T X)^{-1} X^T y = \left(\sum_{i=1}^n x_i x_i^T \right)^{-1} \left(\sum_{i=1}^n x_i y_i \right) \quad (1)$$

Power is used as the dependent variable y in the above equation, also known as regressand. The events are expressed as the X vector of independent variables, a.k.a. regressors. The OLS method outputs a vector α , which holds coefficients extracted from the activity vectors. Then the equation 2 is used to estimate power usage using a new *test* set of events.

$$P_{CPU} = \alpha_0 + \alpha_1 \times event_1 + \dots + \alpha_n \times event_n \quad (2)$$

We evaluate the accuracy of the modelled equation by using data from the benchmark test set with a new set of power values and events. To do this, we measure the percentage difference or Mean Absolute Error(MAE) between the measured power and the estimated power by plugging the new events in the equation. We have tried other metrics like Root Mean Squared(RMS), but they proved to be very sensitive to outliers. In general, approaches like OLS are quite dependent on the inputs and equations used. If the model is too simple it might not give accurate predictions, because it does not use a sufficient number of characteristics/events/regressors to fit the data properly. On the other hand a too complicated model, using many events, might be hard to compute in real-time and can be prone to overfitting the training data and if the training set is not broad enough it might perform poorly on future types of work that have not been included in it. There is a fine balance between simplicity, real-time usability and good performance, but there is still a lot of evidence that linear regression models can produce accurate models and be used in power optimization techniques in embedded systems [41].

6.2 Automated Search

We have explored 3 different search algorithms and 3 optimisation metrics.

6.2.1 Custom Algorithms

We use intelligent search algorithm scripts to identify the best power models from the collected events. We have developed 3 different search algorithms - *bottom-up*, *top-down* and *exhaustive*. As their names suggest they traverse the PMU events search tree in different ways. For all three, we have the ability to choose an initial set of events to start from, as well as the maximum number of events we want in our model. For our platform we always use *CPU_CYCLES* as our

first event since the available PMU has a dedicated counter. Our experiment also show that *CPU_CYCLES* is also the highest correlated single event to CPU power, so it is essential to any PMU event based model. The maximum number of events used in the models depend on the amount of concurrent hardware events we can collect at the same time, which is 6+1 and 4+1 for the Cortex-A15 and Cortex-A7, respectively. This is done to ensure our models are responsive and can be used at run-time. Including more PMU events in the model than there are physical counters results in additional methodology overhead and reduced model accuracy, because the PMU has to multiplex and approximate the extra events.

The first method - *bottom-up* search, presented in Algorithm 1 - goes through the process PMU events data one by one and calculates model performance for each event combination with the starting events list. With each iteration of the algorithm it adds the event which helps improve the model the most. This is repeated until we reach the desired number of regressors or if we cannot improve the optimisation metric any more. The second automatic search method uses a *top-down* approach as shown in Algorithm 2. The algorithm starts off by first making a model using all the available events and then slowly removing the events, which cause this model to improve the most, one by one. This way we trim the search tree from the top, hence the name. Often this algorithm will get *stuck* on a bigger set of events than can physically be collected concurrently on the PMU. Therefore, the rest of the search is done using an *exhaustive* approach which identifies all the combinations of 7 or 5 events from the already pruned search tree and extracts only the best performing combination. The reason we do not just use the *exhaustive* method is because it takes a very long time to complete using the processed PMU events list. For example, for the Cortex-A15 this means a total of 25827165 combinations. This is a very large number of combinations to go through, which is why we have explored the other, *faster*, techniques to improve the probability of identifying the most optimal solution.

6.2.2 Optimisation Criteria

In addition to implementing several search algorithms we have also added in several options for model optimisation criteria. Instead of MAE we can also minimise event cross-correlation or the error standard deviation, two other approaches shown in related literature that could potentially improve model accuracy. They do this by allowing us to traverse a different search tree and thus overcome any possible local optima that the automated search algorithm might be *stuck* on.

Particularly, minimising the error standard deviation implies that the model has a consistent prediction error and is robust to variation in workload. This means that even if the model has a higher average relative error, we can at least expect the same performance across all stages of the workload. In some cases, where we can offset the predicted power, this might be a preferable choice. High event correlation in relation to linear regression, means we introduce relationships between the different events,

Algorithm 1: Bottom-Up Automatic Event Selection

```

Input: DataFile1 // Origin processor data
Input: DataFile2 // Target processor data samples
Input: BenchmarkSplit // The experiment workload train and test benchmark split
Input: EventsPool // The pool of PMU events, to search through
Input: EventsNum // The number of events desired for the model
Output: EventsList // Final optimal list of events

1 begin
2   EventsList  $\leftarrow$  NULL; // Initiate list
3   while EventsNum > 0 do // Search until desired number of events reached
4     EventAdd  $\leftarrow$  NULL; // Initialise helper variable
5     foreach TempEvent in EventsPool do // Try each available event
6       TempList  $\leftarrow$  List + TempEvent; // Build a model using good events together with the tested
          event
7       TempError  $\leftarrow$  MODEL(DataFile1, DataFile2, BenchmarkSplit, TempList); // Use Algorithm 3 to
          validate model.
8       if MinError = NULL then // Use first event metrics as baseline
9         EventAdd  $\leftarrow$  TempEvent;
10        MinError  $\leftarrow$  TempError;
11      else
12        if TempError < MinError then // Overwrite if event improves model
13          EventAdd  $\leftarrow$  TempEvent;
14          MinError  $\leftarrow$  TempError;
15        end
16      end
17    end
18    if EventAdd  $\neq$  NULL then // After searching through all events, check if model can be
        improved
19      EventsList  $\leftarrow$  EventsList + EventAdd; // Add improving event to list
20      EventsPool  $\leftarrow$  EventsPool - EventAdd; // Remove improving event from pool
21      EventsNum  $\leftarrow$  EventsNum - 1; // Reduce number of events to search for
22    else
23      RETURN EventsList; // If no improving event can be found, return list
24    end
25  end
26  RETURN EventsList; // Return list once desired number of events are found
27 end

```

Algorithm 2: Top-Down Automatic Event Selection

```

Input: DataFile1 // Origin processor data samples
Input: DataFile2 // Target processor data samples
Input: BenchmarkSplit // The experiment workload train and test benchmark split
Input: EventsPool // The pool of PMU events, to search through
Input: EventsNum // The number of events desired for the model
Output: EventsList // Final optimal list of events

1 begin
    // Build model from all the available events and use it as baseline for improvement
2   MinError ← MODEL(DataFile1,DataFile2,BenchmarkSplit,EventsPool); // Use Algorithm 3 to validate
    model
3   EventsList ← EventsPool; // Initiate list with all the events available
4   while TRUE do // Start searching, break conditions are inside the loop
5     EventRemove ← NULL; // Initialise helper variable
6     foreach TempEvent in EventsPool do // Try each available event
7       TempList ← Pool − TempEvent; // Build a model using all events but the tested event
8       TempError ← MODEL(DataFile1,DataFile2,BenchmarkSplit,TempList);
9       if TempError < MinError then // Overwrite if event improves model
10        EventRemove ← TempEvent;
11        MinError ← TempError;
12      end
13    end
14    if EventRemove ≠ NULL then // After searching through all events, check if model can be
        improved
15      EventsPool ← EventsPool − EventRemove; // Remove improving event from pool
16      SizePool ← SIZE(EventsPool); // Check how many events are left in pool
        // If desired number of events remain, return pool as list
17      if SizePool = EventsNum then
18        EventsList ← EventsPool;
19        RETURN EventsList;
20      end
21    else
        // If no improving event can be found, return pool as list
22      EventsList ← EventsPool;
23      RETURN EventsList;
24    end
25  end
26 end

```

which can make the model very sensitive to outliers [42]. Minimising event cross-correlation ensures we use events which contribute independently to the model performance and the model is less susceptible to run-time performance variation.

Our general findings suggests that minimising MAE yield the best performing models, even when validated on a separate test dataset. We also confirm that the *bottom-up* search method produces models with higher accuracy than the *top-down* approach. In addition to directly comparing the models we have compiled a set of target accuracy metrics to ensure the optimal models that have been computed can actually be useful for DVFS, presented in Table 2. These metrics are computed by taking into account the maximum difference of instantaneous sample power between two adjacent energy levels for each CPU type either going *up* a frequency or *down* one. The result is given as a percentage to the starting level power and the maximum between any two levels for each workload suite is given as the target accuracy. What this translates to is that the *per-frequency* power model error needs to be lower than this value to ensure the power predictions are for the same CPU frequency level. In order for the models to be used as a guide for DVFS they need to be able to distinguish between the different CPU energy levels, otherwise the scheduler might over or under-compensate, resulting in sub-optimal performance and energy consumption. In our experiments we show that the computed optimal models do satisfy these metrics, which means they can be used as the basis for a DVFS-based scheduler. Detailed breakdown of the experiment results are available in Subsection 8.2.

7 Novel Model Features

This section identifies the three key model features that deliver high model accuracy as well as extended use for DVFS and power prediction across the CPU clusters. These are the main features that differentiate our work from other published models and methodologies for the *big.LITTLE* SoC.

7.1 Per-Frequency Level Models

Our first unique observation is that we can capture the complex behaviour of the development platform much better, by calculating unique coefficients for the model for every frequency level available. The *per-frequency* level models allow us to use the linear regression algorithm on a much tighter dataset, which yield significant improvement in model accuracy. In our initial experiments we report more than 3x lower MAE compared to a single unified model for the full frequency range, while using the same set of events. The main reason that the models work so well is that using the OLS method on the data at each frequency level allows us to capture much more closely the full CPU power. With relation to the model vector, shown in Equation 2, the α_0 term represents the CPU *static* and *idle* power and $\alpha_1 \times event_1 + \dots + \alpha_n \times event_n$ represent the CPU *dynamic* power, so only

training on the data from a single frequency allows us to overcome the high variation between the 0.5s PMU event samples as well as capture implicitly the other technological and OS contributors to CPU power at that level. The downside is that instead of 1 model equation, we have a set of coefficients for every frequency level on each processor type. This means 19 equations for the Cortex-A15 and 13 for the Cortex-A7. The added complexity is still very manageable by a modern system, since the table of model coefficients can be easily loaded into the L1 Cache, thus switching-out the parameters for a new frequency level only takes a few cycles.

7.2 Intra-Core Models

The next stage is to extend the *per-frequency* models to capture CPU power between frequency levels on the same processor type. We call these *intra-core* models. The *per-frequency* models are used to calculate the instantaneous power at every sample but only for the frequency they are trained on, as shown on Figure 2. The *per-frequency* model is in contrast to the *intra-core* model which uses the average number of events from all samples at a frequency level to predict the average power at another level. Since this is done for one processor type within its own frequency range we label then *intra-core*. Another way to think about this is that the *per-frequency* models give a detailed analysis on the application at runtime at only one frequency, and the *intra-core* model gives an estimation of the average power on various frequency levels based on a window of PMU events. The reason we are unable to do this just by using the samples at one frequency and the *per-frequency* model coefficients for another is because the samples are 0.5s apart, therefore the average amount of events at each sample vary greatly between frequencies. This means that the calculated model coefficients for each event are very different for each frequency level. We have investigated an approach to solve this problem by scaling the event samples, instead of recomputing the coefficients. This method is the core component of the *intra-core* models.

The idea is to scale the sampled PMU events by using the property that the events of each data sample are approximately proportional to their averages for the whole runtime of the workload. This turns run-time information to average execution information. The technique to obtain the Event Scaling Factors(ESF) is detailed in Equation 3. During model training we perform this using the training sets of the two frequencies $f1$ and $f2$ that we want to predict between.

The ESF technique allows us to reuse the *per-frequency* model at a target frequency($f2$) with the scaled events from an origin frequency ($f1$). An example is shown in equation 4. After we have trained the *per-frequency* model for $f2$ (obtained coefficients $\alpha_0, \alpha_1, \dots$ etc.) and calculated the ESF for the events between $f1$ and $f2$, we can validate the model using the events information from the test set of $f1$ against the average power of the test set of $f2$. This special type of power model can be extended into a power-aware scheduler for DVFS by using the *per-frequency* models for every frequency and the calculated ESF to predict program

power usage for each frequency level and choosing the most energy efficient configuration.

$$\begin{aligned} \frac{event_{f1}}{event_{f2}} &= \frac{\overline{event_{f1}}}{\overline{event_{f2}}} \\ event_{f1} \frac{\overline{event_{f2}}}{\overline{event_{f1}}} &= event_{f2} \\ \frac{\overline{event_{f2}}}{\overline{event_{f1}}} &= \text{Event Scaling Factor} \end{aligned} \quad (3)$$

$$P_{CPU_{f2}} = \alpha_0 + \alpha_1 \times e_{1_{f1}} \times ESF_1 + \alpha_2 \times e_{2_{f1}} \times ESF_2 + \dots + \alpha_n \times e_{n_{f1}} \times ESF_n \quad (4)$$

7.3 Inter-Core Models

The final model variation involves further extending the *intra-core* models to be able to predict the average power between two processor types. We call these *inter-core* models. This is done by calculating the ESF between the events of the two CPU clusters. We extend Equation 4 into Equation 5 to give an example of how to use the ARM Cortex-A7(L) PMU events to predict the average power for the ARM Cortex-A15(b).

We do this in three steps. First we use the data from the train sets of a frequency level from both CPU clusters to compute the ESF. Then we use the target CPU cluster train set to fit the power model using the selected PMU events. Finally we use the scaled PMU events from the test set of the origin CPU cluster in the computed power model. In order to get the model accuracy we compare the model results using the scaled events against the average power from the test set for the target CPU cluster.

$$\begin{aligned} P_{b_{fb}} &= \alpha_0 + \alpha_1 \times e_{1L_{fL}} \times \frac{\overline{e_{1b_{fb}}}}{\overline{e_{1L_{fL}}}} + \alpha_2 \times e_{2L_{fL}} \times \frac{\overline{e_{2b_{fb}}}}{\overline{e_{2L_{fL}}}} + \\ &\dots + \alpha_n \times e_{nL_{fL}} \times \frac{\overline{e_{nb_{fb}}}}{\overline{e_{nL_{fL}}}} \end{aligned} \quad (5)$$

For our models we only use overlapping events available to the Cortex-A7 PMU and the Cortex-A15 PMU. This means we do not reuse the *per-frequency* model equations, but instead we use our methodology to retrain and validate specific dedicated *inter-core* models for both the single-thread and multi-thread cases. We used a narrow list of 17 common PMU events between the Cortex-A15 and the Cortex-A7, which are taken from the processed list of stable events.

Since we scale the PMU events with their averages, the events that are most suitable for this model are events that are consistent during the measurement intervals. An unseen benefit of the static measurement sampling interval of 0.5s is the ability to obtain the mean value of the PMU events, without having to explicitly calculate the workload runtime, by averaging the data samples.

Because of the nature of our *inter-core* model as an extension of the *intra-core* one, we are able to scale and therefore predict the events from any of the frequencies of

one CPU cluster to any frequencies of the other CPU cluster. This explores the entire transition space of the heterogeneous system, which makes this type of model suitable for a full system energy-aware scheduler. The majority of related publications only consider models built for the Cortex-A7 or Cortex-A15 separately and we have not encountered any related work which has developed such techniques that actually capture the full behaviour of the *big.LITTLE* SoC.

7.4 Model Generation Procedure

Algorithm 3 is a pseudocode representation of the steps involved in training and testing the *intra/inter-core* models. We begin by reading the origin and target datasets for model generation and validation. After all inputs are processed, the algorithms begins extracting the frequency lists for both processors. The main calculation is performed in two loops — the outer loop goes through each frequency of the target set and the inner — through each frequency of the origin. For each frequency of the origin set we extract both the train and test benchmark sets from the origin and target dataset. Then we calculate the ESF using the two train sets and the model coefficients for the target frequency given by the outer loop. Finally we validate the model by using the test benchmark data from the origin dataset of the inner loop frequency and the calculated ESF against the average power of the target frequency test set. We calculate this error for each origin frequency of the inner loop against the target frequency of the outer loop and present the average of those errors as the model performance for the target frequency. This is a many-to-one mapping since we calculate how the model will behave for one target frequency using all available input frequency data. The final step is to repeat this process for all frequencies of the target and present the final model performance metric as the average of all individual target frequency errors. This results in a many-to-one mapping where we have validated the model between any two available frequencies of the origin and target datasets.

8 Results

We use our methodology to compute and validate power models on the ODROID XU3 development board. First we consider the single-thread case using the cBench workload and the *bottom-up* automatic event selection method. We compute the three types of models shown in Section 7 and validate the results against other published models, reporting significant improvements. Afterwards we move on to the multi-thread case using the PARSEC 3.0 workload. We also validate and compare the models against a larger number of published work and also report increased accuracy. In both cases we also include a comparison against a random set of PMU events, to ensure that the proposed search methods identify an optimal set. We also validate the best calculated *per-frequency* model using *n-fold cross-validation*, where we rotate and use 1 benchmark per set to test and all the others to train the model. The average of all iterations give us the final,

Algorithm 3: Advanced Model Training and Testing

```

Input: DataFile1 // Origin processor data samples
Input: DataFile2 // Target processor data samples
Input: BenchmarkSplit // The experiment workload train and test benchmark split
Input: EventsList // The PMU events list, to be used in the model
Output: FullError // Final model performance measurement
1 begin
  // Extract the processor frequency lists
2  READ FreqList1 from DataFile1;
3  READ FreqList2 from DataFile2;
4  foreach Freq2 in FreqList2 do // Model target frequencies
5    foreach Freq1 in FreqList1 do // Model origin frequencies
6      TrainSet1, TestSet1  $\leftarrow$  READ(BenchmarkSplit, Freq1) from DataFile1;
7      TrainSet2, TestSet2  $\leftarrow$  READ(BenchmarkSplit, Freq2) from DataFile2;
8      ESF  $\leftarrow$  CALC(TrainSet2, TrainSet1); // Equation 3
9      Model  $\leftarrow$  TRAIN with OLS(TrainSet2, EventsList); // Equation 1
10     Error1  $\leftarrow$  TEST(TestSet1, ESF, Model, TestSet2); // Equations 4 and 5
11   end
12   Error2  $\leftarrow$  AVG(Error1); // Many-to-one mapping
13 end
14   FullError  $\leftarrow$  AVG(Error2); // Many-to-many mapping
15 end

```

practical model error. Finally we conclude this Section with a discussion on the experiment results.

8.1 Single-Thread Models

After we complete the data collection experiment for the single-thread case and process the data, we proceed to use the *bottom-up* automatic search algorithm to compute the optimal *per-frequency* models. The final results when validating the models using the benchmark test set are shown in Figures 4a and 4c and Table 4. Each model is represented as *Model Code* (#) and the corresponding MAE for the Cortex-A15 and Cortex-A7 are given in column *b*, for *big*, and column *L*, for *LITTLE*.

In our first use of the automatic search algorithm we first compute the top events, excluding *CPU_CYCLES* and then add it to the final list. We see that with each algorithm iteration and added event to the list we observe a reduction in model MAE. Later on, when we consider the multi-thread case, we use multiple search algorithms and *CPU_CYCLES* as the starting list. This improves accuracy further by considering event relationships with *CPU_CYCLES* from the beginning of the algorithm. The final calculated models have an MAE of 1.76% and 2.99% for the Cortex-A15 and Cortex-A7, which satisfies the target requirement. It is interesting to note that we reach this high accuracy for the Cortex-A15 by only using 6 out of 7 available concurrent PMU events. This shows that the *bottom-up* method has not managed to find a 7th event that improves the model. It is also interesting to see that the model for the Cortex-A7 performs worse than the model for the Cortex-A15, which is surprising since the *LITTLE* processor is much simpler and has a smaller power range. This is due to the fact the Cortex-A15 PMU has access to a larger set of specialised hardware events, which can

capture the CPU behaviour better. The full set of events used in both models is given below:

Cortex-A15 *per-frequency* single-thread model events:
CPU_CYCLES, *L1I_CACHE_ACCESS*,
L1D_CACHE_ACCESS, *BUS_CYCLES*,
BUS_PERIPH_ACCESS, *BRANCH_SPEC_EXEC_RET*

Cortex-A7 *per-frequency* single-thread model events:
CPU_CYCLES, *BUS_READ_ACCESS*,
L2D_CACHE_REFILL, *UNALIGNED_LOAD_STORE*,
BUS_CYCLES

After computing and evaluating the *per-frequency* models, we proceed to calculate the *intra-core* models. For them, we reuse the *per-frequency* model equations and calculate the ESF according to Algorithm 3. The resulting model MAE for each processor type is presented in Figures 4b, 4d and Table 5 as *Model Code* (2). We see that the *intra-core* model has higher accuracy, because it only approximates the average power and cannot give detailed application profiling during execution like the standard *per-frequency* model can. The *intra-core* model MAE stands at 0.99% and 1.01% for the Cortex-A15 and Cortex-A7. This is definitely within target, so the models can accurately be used in energy-aware DVFS.

The final model computed for the single-thread case is the *inter-core* one. We use the *bottom-up* automatic event search and the list of common PMU events for both processor types to calculate the models using the methodology detailed in Subsection 7.3. We see the models have a very low MAE of 0.6% when the average power of the Cortex-A15 using the events from the Cortex-A7 and 0.69% MAE vice versa. We are able to achieve this with fewer than the 5 concurrent events limit. The full list of events for each model are presented below:

(a) Cortex-A15 Event Selection

(b) Cortex-A15 Model Comparison

(c) Cortex-A7 Event Selection

(d) Cortex-A7 Model Comparison

Figure 4: Single-Thread Results on ODROID XU3

Model Code	Model Type	b	L
(1)	CPU_CYCLES	5.83	4.72
(2)	+ 1 Event	3.47	4.99
(3)	+ 2 Events	2.04	3.39
(4)	+ 3 Events	2.04	3.10
(5)	+ 4 Events	1.89	2.99
(6)	+ 5 Events	1.76	-

Table 4: Event Selection Steps MAE

Model Code	Model Type	b	L
(1)	Per-Frequency	1.76	2.99
(2)	Intra-Core	0.99	1.01
(3)	Inter-Core	0.60	0.69
(4)	Nikov et al. [6]	3.41	4.81
(5)	Rodrigues et al.[19]	2.79	4.14
(6)	Pricopi et al.[17]	5.01	-
(7)	Walker et al.[18]	4.81	-
(8)	Per-Frequency Random	3.31	3.52
(9)	Per-Frequency N-Fold	2.42	4.56

Table 5: Single-Thread Models MAE

Cortex-A7 to Cortex-A15 *inter-core* single-thread model events: *CPU_CYCLES*, *EXCEPTION_RETURN*, *BRANCH_MISPRED*, *L2D_CACHE_WB*, *BUS_ACCESS*

Cortex-A15 to Cortex-A7 *inter-core* single-thread model events: *CPU_CYCLES*, *L1I_CACHE_ACCESS*, *BRANCH_PRED*

The final step in the single-thread case analysis is to compare our final models against other published work. We calculate and verify our published work Nikov et al. [6], which uses an *intuitive* set of PMU events and the works of Rodrigues et al. [19] Pricopi et al. [17] and Walker et al. [18]. The final model MAE are presented as *Model Codes* (4), (5), (6) and (7), respectively. More details

about the related work used the model comparison is given in Section 2. We have translated the models as closely to our platform and methodology as possible, but some lack a feasible representation for the Cortex-A7. The events used for each model are given below:

Nikov et al. [6] *per-frequency* model events:
`CPU_CYCLES`, `L1D_CACHE_ACCESS`,
`L1I_CACHE_ACCESS`, `INST_RETIRED`,
`DATA_MEM_ACCESS`

Rodrigues et al. [19] model events:
`L1I_CACHE_ACCESS`, `L1D_CACHE_ACCESS`,
`(EXCEPTION_TAKEN + BRANCH_MISPRED)`

Pricopi et al. [17] model events:
`INST_SPEC_EXEC`, `INST_SPEC_EXEC_INT`, `INST_SPEC_EXEC_VFP`,
`CPU_CYCLES`, `INST_SPEC_EXEC`, `INST_SPEC_EXEC`,
`L1D_CACHE_ACCESS`, `L2D_CACHE_ACCESS`, `L2D_CACHE_REFILL`,
`INST_SPEC_EXEC`, `INST_SPEC_EXEC`, `INST_SPEC_EXEC`

Walker et al. [18] model events:
`CPU_CYCLES`, `INST_SPEC_EXEC`, `L2D_READ_ACCESS`,
`UNALIGNED_ACCESS`,
`INST_SPEC_EXEC_INTEGER_INST`,
`L1I_CACHE_ACCESS`, `BUS_ACCESS`

Model Code (8) shows the MAE of a model using a random selection of events using the full amount of hardware counters available to the PMU. We can see that the search results produce a much more accurate model than the random set of events. An interesting thing to note is that the majority of the published work performs worse, which shows that mathematical and statistical approaches to event selection significantly outperform engineer intuition.

The final model, identified by *Model Code* (9) shows the practical error of the *per-frequency* power model using the optimal set of events. We see that even after thorough validation across the entire workload suit, the model is well within the target accuracy metrics. We only produce cross-validation results for the *per-frequency* model, since the *intra* and *inter-core* models do not exhibit such degrees of variability, making them very stable and in no need of any further validation.

We show that our models have a significantly lower MAE, showing that our methodology and particularly our event selection technique are beneficial. We also see that the improved experiment set-up results in increased model accuracy, when comparing against our previous work. We show our refined methodology can produce state of the art power models for both processor types individually and also produce accurate heterogeneous models for the *big.LITTLE* SoC on the ODROID XU3 development platform.

8.2 Multi-Thread Models

After we investigate the single-thread case we move onto the multi-thread case using the PARSEC 3.0 workload. The experimental set-up and execution details are given in Section 5. After data collection and processing we proceed to build and evaluate the power models using our methodology. For this case we include the *top-down*, assisted by the *exhaustive* search algorithm for event selection and the MAE standard deviation (MAESD) and the event cross-correlation(ECC) optimisation criteria. Details about these techniques are given in Section 6. The results from the

comparison between the different search algorithms and optimisation criteria for the *per-frequency* models is given in Figures 5a, 5c and Table 6. First we compare the *bottom-up* and the *top-down + exhaustive* search, shown as *Model Code* (1) and (2). We see that the *bottom-up* method is not only faster, but produces better models both for the Cortex-A15 and Cortex-A7. We then compare the three optimisation criteria using the same search algorithm, as seen in *Model Code* (1),(4) and (5). Our conclusion is that our initial approach used in the single-thread case, namely to use the *bottom-up* search with minimising model MAE produces the best models for the multi-thread case as well. Our final model MAE is 7.12% for the Cortex-A15 and 5.46% Cortex-A7. The model events are given below:

Cortex-A15 *per-frequency* multi-thread model events:
`Cores(#)`, `CPU_CYCLES`, `L1D_READ_ACCESS`,
`BRANCH_MISPRED`, `BARRIER_SPEC_EXEC_DMB`,
`L2D_INVALIDATE`,
`BRANCH_SPEC_EXEC_IMM_BRANCH`, `BUS_CYCLES`

Cortex-A7 *per-frequency* multi-thread model events:
`Cores(#)`, `CPU_CYCLES`, `L1I_CACHE_ACCESS`,
`L1D_CACHE_EVICTION`, `DATA_READS`,
`IMMEDIATE_BRANCHES`

The multi-thread models use a completely different set of events compared to the single-thread *per-frequency* ones and have a higher MAE. In order to investigate if the decreased accuracy is an issue with the limited amount of concurrent model events we use the methodology to compute a theoretical model without a limit on the events list, shown in *Model Code*

(3). We manage to go down to 6.92% and 5.35% MAE for the Cortex-A15 and Cortex-A7. The final theoretical model has 1 additional event `L1D_TLB_REFILL` for the Cortex-A15 and 3 additional events `SW_CHANGE_PC`, `UNALIGNED_LOAD_STORE`, `L1D_CACHE_REFILL` for the Cortex-A7. The results show that the multi-thread case is more complex to model, which is expected. Our next step is to compute the *intra-core* and *inter-core* models. We use the techniques from Subsections 7.2 and 7.3. The final results are given as *Model Code* (1),(2) and (3) in Figures 5b, 5d and Table 7. The *intra* and *inter-core* models have a very low MAE, less than 2.5% for all cases, which is more than enough accuracy to satisfy the target. The PMU events used in the dedicated *inter-core* models are given below:

Cortex-A7 to Cortex-A15 *inter-core* multi-thread model events: `Cores(#)`, `CPU_CYCLES`, `EXCEPTION_TAKEN`,
`L2D_CACHE_WB`, `BRANCH_MISPRED`,
`EXCEPTION_RETURN`

Cortex-A15 to Cortex-A7 *inter-core* multi-thread model events: `Cores(#)`, `CPU_CYCLES`, `EXCEPTION_TAKEN`,
`EXCEPTION_RETURN`

In order to complete our evaluation, we compare the models against published work. This includes our previously calculated *per-frequency* single-thread model events as well as the events from Rodrigues et al. [19] Pricopi et al. [17] and Walker et al. [18], which we used for the single-thread comparison. These models are presented as *Model Code* (4),(6),(7) and (8) in the results graphs

(a) Cortex-A15 Search Methods

(b) Cortex-A15 Model Comparison

(c) Cortex-A7 Search Methods

(d) Cortex-A7 Model Comparison

Figure 5: Multi-Thread Results on ODROID XU3

Model Code	Model Type	b	L
(1)	Bot-Up MAE	7.12	5.46
(2)	Top-Down + E. MAE	8.37	5.91
(3)	Bot-Up MAE MAX	6.92	5.35
(4)	Bot-Up MAESD	14.68	9.22
(5)	Bot-Up ECC	20.35	9.86

Table 6: Search Method MAE

Model Code	Model Type	b	L
(1)	Per-Frequency	7.12	5.46
(2)	Intra-Core	0.65	1.85
(3)	Inter-Core	2.39	1.62
(4)	Single-Thread	13.93	13.23
(5)	Rodrigues et al. [19]	14.91	9.10
(6)	Rethinagiri et al. [20]	27.74	37.51
(7)	Pricopi et al. [17]	30.56	-
(8)	Walker et al. [18]	9.79	-
(9)	Per-Frequency Random	10.74	7.53
(10)	Per-Frequency N-Fold	8.28	6.41

Table 7: multi-thread Models MAE

and table. We also include an additional model in our comparison Rethinagiri et al. [20], in order to use more models in the Cortex-A7 evaluation. That model is labelled as *Model Code* (5) in the results table and graphs and the full events list is given below:

Rethinagiri et al. [20] model events: $\frac{INST_RETIRED}{CPU_CYCLES}$,
(*L1I_CACHE_REFILL* + *L1D_CACHE_REFILL*),
L2D_CACHE_REFILL

We can see from the results that our *per-frequency* models are significantly better than the related work, achieving higher than 2.5% and 3.5% reduction in MAE compared to the next best model for the Cortex-A15 and Cortex-A7, respectively. We also see that reusing the optimal events for the single-thread case results in a high model MAE. We thought this could be due to the different workload, but when we tried the optimal multi-thread events on the cBench single-thread workload we obtained 3.23% and 3.88% MAE for the Cortex-A15 and Cortex-A7. This shows how different and complex the multi-thread scenario is and optimising the multi-thread events also results in producing good models for a single-thread workload.

Finally, in Figures 5b and 5d and Table 7 we also present a *per-frequency* model computed from a random set of hardware events utilising all available PMU registers as well as a *n-fold cross-evaluation* of the *per-frequency* model, as done with the single-thread model results in the previous Section 8.1. These models are presented by *Model Code* (9) and (10) respectively. We see again that the custom search methods produce an optimal set of events and outperform the random selection, with only one other published model achieving lower error as well. We also note that the cross-validated model error is still within the target accuracy, by a small margin. This highlights a need to investigate other methods of computing the model, not just linear regression, in order to further reduce the error and ensure model stability. This is planned as future work.

Overall we show that our methodology can produce accurate run-time power models. After our investigation into different event selection algorithms and criteria, we show that, for our set-up, the *bottom-up* search method minimising MAE, described in Subsection 6.2, generates the best performing models. Our *per-frequency* model, explained in Subsection 7.1, achieves less than 3% and 7.5% MAE for the single-thread and multi-thread cases, for both processor types. We also show that our coarse-grained *intra* and *inter-core* models, detailed in Subsections 7.2 and 7.3, also achieve very high prediction accuracy with less than 2.5% MAE for all model variations. Our *per-frequency* models show at least 1% and 2.5% lower MAE for the single-thread and multi-thread cases, compared to the next best model.

9 Conclusions

In this research we develop a methodology for training and validation of PMU based power models using a *big.LITTLE* platform. We can easily reconfigure the workloads, the model generation algorithms or even the experiment data collection scripts. We have used the methodology to develop intra-core

fine-grained *per-frequency* level models for single-thread and multi-thread workload, with reported MAE of less than 3% for the former and 7.5% for the latter. In addition to these models we have developed inter-core coarse-grained models specifically for use in DVFS and advanced scheduling strategies. We show we can predict the average power between different CPU energy levels with more than 97% accuracy between the two processor types. To our knowledge this is the first work, which has utilised specific techniques to generate heterogeneous PMU based power models for the *big.LITTLE* SoC, which capture workload power usage when transitioning between the two processor types. In order to validate the methodology we have thoroughly compared our power models against published work, with our final models achieving greater than 2% lower MAE compared to the next best model. As seen in the paper the initial phase of power model creation requires direct access to power measurements but once in the field the model can be used directly to obtain power estimations in the production SoC with no access to the power rails. As a continuation of our work, we plan to migrate the methodology to current 64-bit *big.LITTLE* and *DynamiQ* platforms and investigate other methods to compute the model coefficients to further increase model accuracy, specifically for the *per-frequency* models.

Acknowledgements

This work is supported by ARM Research funding, through an EPSRC iCASE studentship and the University of Bristol and by the EPSRC ENEAC grant number EP/N002539/1.

References

- [1] Jos Antonio Esparza Isasa, Peter Larsen, and Finn Hansen. A holistic approach to energy-aware design of cyber-physical systems. *International Journal of Embedded Systems*, 9:283, 01 2017.
- [2] Dong Hyuk Woo and Hsien-Hsin S. Lee. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer* 12, pages 24–31, 2008.
- [3] ARM. big.little technologies. <http://www.arm.com/products/processors/technologies/biglittleprocessing.php>, 2017. [Online; accessed 10-Oct-2013].
- [4] ARM. Arm unveils its most energy efficient application processor ever; redefines traditional power and performance relationship with big.little processing. https://www.arm.com/about/newsroom/arm-unveils-its-most-energy-efficient-*.php, 2011. [Online; accessed 21-Oct-2014].
- [5] ARM. Arm dynamiq: Technology for the next era of compute. https://community.arm.com/processors/b/blog/posts/arm-dynamiq-technology-for-the-next-era-of*, 2017. [Online; accessed 10-Feb-2018].

- [6] Krastin Nikov, Jose L. Nunez-Yanez, and Matthew Horsnell. Evaluation of hybrid run-time power models for the ARM big. Little architecture. *Proceedings - IEEE/IFIP 13th International Conference on Embedded and Ubiquitous Computing, EUC 2015*, pages 205–210, 2015.
- [7] Krastin Nikov. Datacollect. https://github.com/kranik/DATACOLLECT/tree/master/ARMPM_datacollect/ODROID_XU3, 2017. [Online; accessed 29-Jul-2017].
- [8] Krastin Nikov. Buildmodel. https://github.com/kranik/BUILDMODEL/tree/master/ARMPM_buildmodel, 2017. [Online; accessed 01-Aug-2017].
- [9] Xiao Jun Wang, Feng Shi, Yi Zhuo Wang, Hong Zhang, Xu Chen, and Wen Fei Fu. Power-aware high level evaluation model of interconnect length of on-chip memory network topology. *International Journal of Computational Science and Engineering*, 17:422, 01 2018.
- [10] Christian Poellabauer, Dinesh Rajan, and Russell Zuck. Ld-dvs: Load-aware dual-speed dynamic voltage scaling. *IJES*, 4:112–126, 01 2009.
- [11] Mayuri Digalwar, Pravin Gahukar, Biju Raveendran, and Sudeept Mohan. Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors. *International Journal of Embedded Systems*, 9:523, 01 2017.
- [12] Jose Nunez-Yanez and Geza Lore. Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip. *Microprocessors and Microsystems*, 37(3):319–332, may 2013.
- [13] Karan Singh, Major Bhadauria, and Sally a. McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46, 2009.
- [14] Connor Imes and Henry Hoffmann. Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores. *ACM SIGBED Review*, 11(4):49–54, 2015.
- [15] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-hashimi, and Geoff V Merrett. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *Ieee Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 1–14, 2015.
- [16] MR Guthaus and JS Ringenberg. MiBench: A free, commercially representative embedded benchmark suite. . . ., 2001. *WWC-4*. . . ., pages 3–14, 2001.
- [17] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Power-performance modeling on asymmetric multi-cores. *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 1–10, sep 2013.
- [18] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-hashimi, and Geoff V Merrett. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *Ieee Transactions on Computer Aided Design of Integrated Circuits and Systems*, 36(1):1–14, 2017.
- [19] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, 2013.
- [20] Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atillah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. System-level power estimation tool for embedded processor based platforms. *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation Methods and Tools - RAPIDO '14*, pages 1–8, 2014.
- [21] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335. IEEE Computer Society, 1997.
- [22] Hardkernel. Odroid-xu3. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127, 2013. [Online; accessed 12-March-2015].
- [23] Robin Randhawa. Software Techniques for ARM big.LITTLE Systems, 2013.
- [24] Unity Technologies. Mobile (android) hardware stats 2017-03. <https://web.archive.org/web/20170808222202/http://hwstats.unity3d.com:80/mobile/cpu-android.html>, 2017. [Online; accessed 29-Sep-2018].
- [25] Reinhold P. Weicker. "dhrystone" benchmark program. <http://www.netlib.org/benchmark/dhry-c>, 1988. [Online; accessed 20-Oct-2013].
- [26] Rich Painter. An update of the original 1987 c version of the whetstone benchmark. <http://www.netlib.org/benchmark/whetstone.c>, 1998. [Online; accessed 20-Oct-2013].
- [27] Jack Dongarra, Jim Bunch, Cleve Moler, and Pete Stewart. Linpack. <http://www.netlib.org/linpack/>, 1984. [Online; accessed 20-Oct-2013].
- [28] Jason Clemons, Haishan Zhu, Silvio Savarese, Todd Austin, and Ann Arbor. MEVBench : A Mobile Computer Vision Benchmarking Suite. pages 91–102, 2011.

- [29] JA Stratton and Christopher Rodrigues. Parboil: A revised benchmark suite for scientific and commercial throughput computing. ... *Computing*, 2012.
- [30] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, oct 2009.
- [31] Anthony Gutierrez, Ronald G Dreslinski, Thomas F Wenisch, Trevor Mudge, Ali Saidi, Chris Emmons, and Nigel Paver. Full-system analysis and characterization of interactive smartphone applications. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 81–90. IEEE, 2011.
- [32] Phoronix. Phoronix test suite. <http://www.phoronix-test-suite.com/>, 2017. [Online; accessed 20-Oct-2013].
- [33] cTuning. Collective benchmark. <http://ctuning.org/wiki/index.php/CTools:CBench>, 2015. [Online; accessed 19-Oct-2014].
- [34] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrisnan, and S.K. Weeratunga. The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.
- [35] Christian Bienia. BENCHMARKING MODERN MULTIPROCESSORS. 2011.
- [36] Karel De Vogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. *Parallel Processing and ...*, pages 793–803, 2014.
- [37] ARM. Cortex-a15 revision: r2p0 technical reference manual. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438c/DDI0438C_cortex_a15_r2p0_trm.pdf, 2011. [Online; accessed 10-Dec-2013].
- [38] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.
- [39] Tadayoshi Fushiki. Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2):137–146, 2011.
- [40] Michael H Kutner, Chris Nachtsheim, John Neter, and William Li. *Applied linear statistical models*. McGraw-Hill Irwin, 2005.
- [41] Hans Jacobson and Alper Buyuktosunoglu. Abstraction and microarchitecture scaling in early-stage power modeling. ... *(HPCA), 2011 IEEE ...*, pages 394–405, 2011.
- [42] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*, volume 821. John Wiley & Sons, 2012.