Power-Aware Performance Adaptation of Concurrent Applications in Heterogeneous Many-Core Systems

Ali Aalsaud[1], Rishad Shafik[1], Ashur Rafiev[1], Fie Xia[1], Sheng Yang[2] and Alex Yakovlev[1]

[1]School of EEE, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK

[2]School of ECS, University of Southampton, Southampton, SO17 1BJ, UK

E-mail: {a.m.m.aalsaud, rishad.shafik, ashur.rafiev, fei.xia, alex.yakovlev}@newcastle.ac.uk

ABSTRACT

Modern embedded systems execute multiple applications, both sequentially and concurrently. These applications are exercised on heterogeneous platforms generating varying power consumption and system workloads (CPU or memory intensive or both). As a result, determining the most energy-efficient system configuration (i.e. the number of parallel threads, their core allocations and operating frequencies) tailored for each kind of workload and application scenario is extremely challenging. In this paper, we propose a novel runtime optimization approach with the aim of achieving maximized power normalized performance considering dynamic variation of workload and application scenarios. Fundamental to this approach is a comprehensive study to investigate the tradeoffs between inter-application concurrency with performance and power consumption under different system configurations. Using real experimental measurements on an Odroid XU-3 heterogeneous platform with a number of PARSEC benchmark applications, we model power normalized performance (in terms of IPS/Watt) underpinning analytical power and performance models, derived through multivariate linear regression (MLR). Using these models, we show that with increasing number of concurrent CPU intensive applications show variable gains in IPS/Watt compared to the memory intensive applications in both sequential and concurrent application scenarios. Furthermore, we demonstrate that it is possible to continuously adapt system configuration through a low-cost and linear-complexity runtime algorithm, which can improve the IPS/Watt by up to 125% compared to the existing approach.

1. INTRODUCTION

Running multiple software applications concurrently on the same platform is rapidly becoming the norm of modern computing, as are system platforms with higher complexity that cater to such uses. This increasing system complexity in both hardware and software emphasizes a major challenge for computing systems, especially mobile and embedded systems, namely the performance-energy tradeoff [1].

This has led to techniques for mitigating power consumption and performance degradation concerns. Power-aware platform design including many-core architectures with the provision for dynamic task mapping and voltage frequency scaling (DVFS) have been the preferred tools for system designers over the years [2, 3].

ISLPED '16, August 08-10, 2016, San Francisco Airport, CA, USA

© 2016 ACM. ISBN 978-1-4503-4185-1/16/08...$15.00

DOI: http://dx.doi.org/10.1145/2934583.2934612

Table 1: Features and limitations of the existing approaches.

Approach	Application	Validation	Key method
[3]-[7]	Single app	Simulation	Offline optimization, DVFS
[8]	Single app	Simulation	Runtime optimization, task mapping, DVFS
[9]	Single app	Simulation	Offline optimization, CPU only sched, DVFS
[10]	Single app	Simulation	Offline optimization, task mapping, DVFS
[11]	Single app	Implementation	Offline optimization, task mapping, DVFS
[12]	Single app	Implementation	Runtime optimization, FPGA only, DVFS
Proposed	Concurrent	Implementation	Runtime optimization, task mapping, DVFS

In the past few years, there have been various studies in energy efficiency in embedded systems, as shown in Table 1. A power control approach for many-cores processors executing single application has been proposed in [3]. This approach has three layers of design features also shown by other researchers: firstly, adjusting the clock frequency of the chip depending on the power budget; secondly, dynamically group cores to run-the same applications (as also shown in [4, 5]), and finally, modify the frequency of each core group (as also shown in [6, 7]).

Among others, Goraczko et al. [8] and Luo et al. [9] proposed DVFS approaches with software task partitioning and mapping of single applications using a linear programming (LP) based optimization during runtime to minimize the power consumption. Goh et al. [10] proposed a similar approach of task mapping and scheduling for single applications described by synthetic task graphs.

Several other works have also shown power minimization approaches using practical implementation of their approaches on heterogeneous platforms. For example, Sheng et al. [11] presented an adaptive power minimization approach using runtime linear regression-based modeling of the power and performance tradeoffs. Using the model, the task mapping and DVFS are suitably chosen to meet the specified performance requirements. Nabina and Nunez-Yanez [12] presented a similar DVFS approach for FPGA-based video motion compensation engines using runtime measurements of the underlying hardware.

A number of studies have also shown analytical studies using simulation tools like gem5, together with McPAT [13, 14] for single applications. These works have used DVFS, task mapping, and offline optimization approaches to minimize the power consumption for varying workloads.

Modern applications exercise underline hardware in different ways, generating varying tradeoffs between power and performance. For example, CPU intensive applications tend to have higher computing workloads with less memory bandwidth. On the other hand memory intensive applications typically exercise higher memory bandwidth with low CPU workloads [15]. When these applications run concurrently the workloads generated by the hardware may vary significantly compare to that of a single application. Hence, energy efficiency cannot be automatically guaranteed using the existing approaches that are agnostic of concurrent application workloads and behaviors (Table 1). In this work, we address the limitations of the above works and propose an adaptive approach, which monitors application scenarios at runtime. The aim is to determine the optimal system configuration such that power normalized performance can be maximized at all times. We adopt an experimental approach depending on profiling real power consumption and performance measurement for single and concurrent applications. For the first time, our study reveals the impact of parallelism in different types of heterogeneous cores on performance, power consumption and power efficiency in terms of IPS/Watt, which can be deduced as the number of instructions per unit of energy [16]. In our proposed approach, we make the following specific ***contributions***:
a) investigate the CPU performance in terms of instructions per cycle (IPC) and power tradeoffs using directly measured values from the performance counters,
b) use real application benchmarks, suitably chosen from a pool of available applications, including CPU intensive, memory intensive, and other combinations,
c) use multivariate linear regression (MLR) to model power and performance tradeoffs expressed as IPS/Watt, and
d) maximize IPS/Watt for single and concurrent application scenarios using low-cost runtime adaptation algorithm.

To the best of our knowledge, this is the first runtime optimization approach for concurrent applications, practically implemented and demonstrated on a heterogeneous many-core system. The rest of the paper is organized as follows. Section 2 shows the experimental environment, the configuration of system used in the experiment and the applications. The system approach is described in Section 3, which is the first step to produce power, performance and power normalized performance models for ARM heterogeneous processors using MLR. Section 4 presents experimental results. Section 5 concludes the paper.

2. SYSTEM ARCHITECTURE AND APPLICATIONS

The popularity of heterogeneous architectures, containing two or more types of different CPUs is growing [1]. These systems offer better performance and concurrency, however it is necessary to ensure optimal power and energy consumption. The Odroid-XU3 board supports techniques such as DVFS, affinity and core disabling, commonly used to optimize system operation in terms of performance and energy consumption [18] [19]. The Odroid-XU3 board is a small eight-core computing device implemented on energy-efficient hardware. The board can run Ubuntu 14.04 or Android 4.4 operating systems. The main component of Odroid-XU3 is the 28 nm Application Processor Exynos 5422. The architecture of the processor is shown in Figure 1. This System-on-Chip is based on the ARM big.LITTLE heterogeneous architecture and consists of a high performance Cortex-A15 quad core processor block, a low power Cortex-A7 quad core block, a Mali-T628 GPU and 2GB DRAM LPDDR3. The board contains four real time current sensors that give the possibility of the measurement of power consumption on the four separate power domains: big (A15) CPUs, little (A7) CPUs, GPU and DRAM. In addition, there are also four temperature sensors for each of the A15 CPUs and one sensor for the GPU.

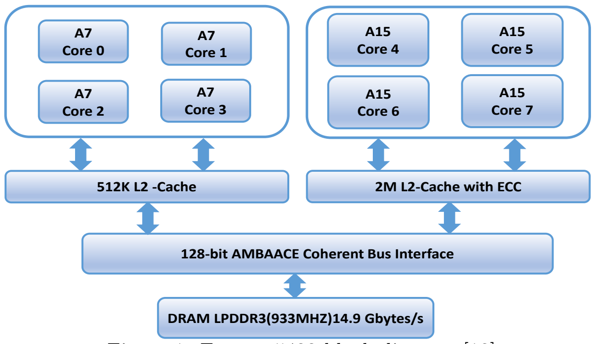


Figure 1: Exynos 5422 block diagram [18].

On the Odroid-XU3, for each power domain, the supply voltage (Vdd) and clock frequency can be tuned through a number of pre-set pairs of values. The performance-oriented Cortex-A15 block has a range of frequencies between 200 MHz and 2000 MHz with a 100 MHz step, whilst the low-power Cortex-A7 quad core block can scale its frequencies between 200 MHz and 1400 MHz with a 100 MHz step. The PARSEC [17] benchmark suite attempts to represent both current and emerging workloads for multiprocessing hardware. It is a commonly used benchmark suite for evaluating concurrency and parallel processing. We therefore use PARSEC on the Odroid-XU3 platform, whose heterogeneity can be representative of different design choices that can greatly affect workloads. PARSEC applications exhibit different memory behaviours, different data sharing patterns, and different workload partitions from most other benchmark suites in common use. The characteristics of applications, according to [17], which are used in this paper can be seen in Table 2. Three applications (*ferret*, *bodytrack* and *fluidanimate*) are selected to represent CPU intensive, memory intensive, and CPU with memory intensive respectively. Such a classification reduces the effort of model characterisation for combinations of concurrently running applications.

3. PROPOSED APPROACH

We develop runtime power and performance models using MLR. Based on these models a runtime adaptation is derived to determine the most energy-efficient system configurations for different application scenarios.

3.1 Modeling Power/Performance Tradeoffs

3.1.1 Power Model

Power consumption can be divided into two parts: dynamic power and static power. Dynamic power is the switching overhead in transistors, so it is established by runtime events. Static power is mainly determined by circuit technology, chip layout and operating temperature [13].

Experiments with the Odroid-XU3 platform were carried out in order to examine its power consumption under different operation frequencies and voltages. The frequency of

Table 2: Qualitative summary of the inherent key characteristics of PARSEC benchmarks [17].

Program	Application Domian	Application Type	Parallelization Model	Granularity	Working Set	Data Usage Sharing	Exchange
bodytrack	Computer Vision	CPU and memory intensive	data-parallel	medium	medium	high	medium
ferret	Similarity Search	CPU intensive	pipeline	medium	unbounded	high	high
fluidanimate	Animation	memory intensive	data-parallel	fine	large	low	medium

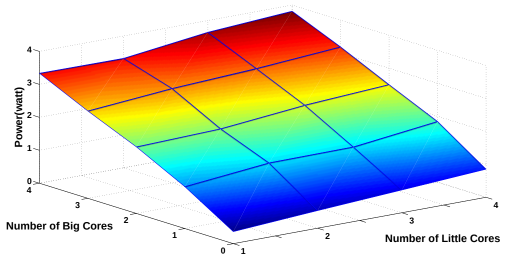


Figure 2: Total power for *ferret* application at 1400 MHz frequency.

each block can be changed independently using utility programs and the system scales the operating voltage of the block to fit the chosen frequency. The eight cores in the board are numbered as follows: core 0, core 1, core 2 and core 3 belong to the A7 (little) processor block, core 4, core 5, core 6 and core 7 belong to the A15 (big) processor block. Figure 2 presents the power consumption of *ferret* application for different thread-to-core allocations. As expected the power consumption increases as more cores are allocated for the given application.

Figure 3 depicts the power distribution between the cores and the memory for different application scenarios. The following three observations can be made from the figure. Firstly, it is clearly seen that the total power consumption for A15 and A7 for CPU intensive application (*ferret*) is higher than memory intensive application (*fluidanimate*). Secondly, in cases where threads are allocated to little cores only, the power of A15 cores is idle power and the total power dissipation for the big cores rise up from 0.39 Watt at 200 MHz to 2.22 Watt at 1400 MHz which shows the impact of DVFS on the total power. Finally as can be seen in Figure 3

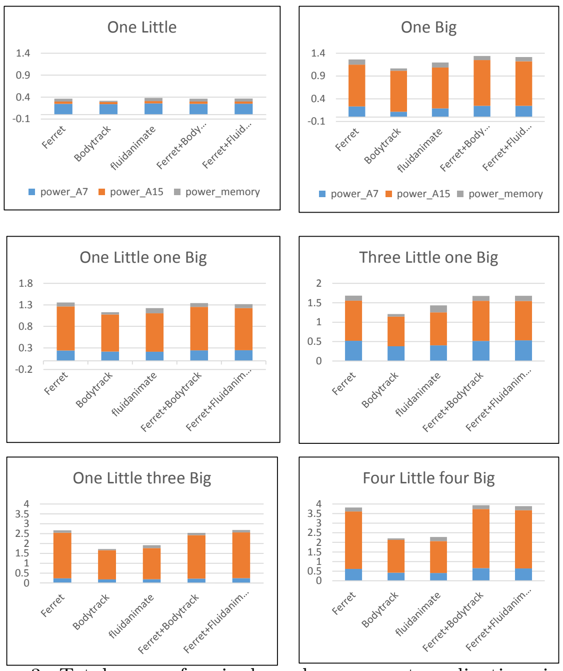


Figure 3: Total power for single and concurrent applications in different configuration running at 1400 MHz.

the memory power is much smaller than the companied A7 and A15 processors power. The variation of memory power depends on applications and the execution scenarios.

We use MLR to determine the relation between power and the number of cores, types of cores (big, little), and frequency. This relation is hypothesized to fit the following expression based on theory [13]:

$$P = \sum_{i=1}^{K} \alpha_i x_i V_i^2 f_i + \epsilon_1(x), \qquad (1)$$

where the first term($\sum_{i=1}^{K} \alpha_i x_i V_i^2 f_i$) of (1) represents the dynamic power, K is the number of group of cores, and the second term ($\epsilon(x)$) represents static and dynamic power of memory, leakage, and interconnect power. Coefficient α_i includes activity factor. In the case of Exynos 5422 (i.e. Odroid big.LITTLE) (1) can be approximated as:

$$P = \alpha_{A7} x_{A7} V_{A7}^2 f_{A7} + \alpha_{A15} x_{A15} V_{A15}^2 f_{A15} + \epsilon_1, \qquad (2)$$

where x_{A7} is the number of little cores, x_{A15} is the number of big cores, V_{A7} and V_{A15} are the voltages of A7 and A15 cores respectively, f_{A7} and f_{A15} represent the frequencies for A7 and A15 respectively; $\alpha_{A7}, \alpha_{A15}$ are MLR coefficients. Tables 3 and 4 show the result of MLR. All MLR procedures for these coefficients have returned mean-squared error (or MSE) coefficient of determination values of 0.92 or better, confirming the applicability of this hypothesis.

3.1.2 Performance Model

In this work, the experiments present the application behaviour on a given architecture and provides realistic values of the performance for the ARM processors. We use Ubuntu 14.04 kernel with our performance counter tool designed to gather processor performance events namely instructions retired in order to depict the behaviour of a thread execution in the heterogeneous system.

Figure 4 shows the real performance measurements in terms of instruction per second (IPS) for various thread-to-core allocations and frequencies. It can be observed that a memory intensive operation on its own has lower IPS than CPU heavy, as expected. However, when running these types together, the overall IPS is high. The clock-independent performance metric is instructions per cycle (IPC) that can be derived from IPS by knowing the clock frequency.

IPC is approximately a constant level in case of running

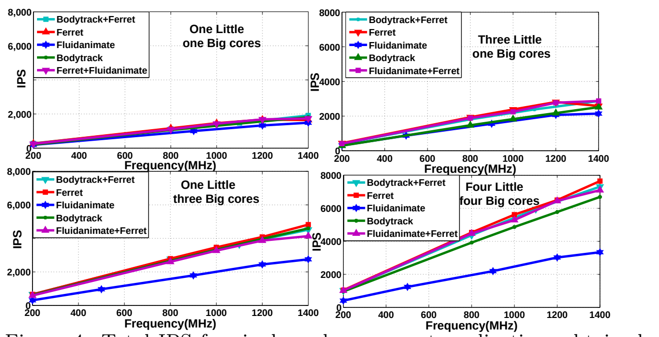


Figure 4: Total IPS for single and concurrent applications obtained from performance counters.

Table 3: Single Application Power Models.

Freq. (MHz)	ferret			**MSE**	bodytrack			**MSE**	fluidanimate			**MSE**	**Volt. A7**	**Volt. A15**
	α_{A7} **(e-11)**	α_{A15} **(e-10)**	ϵ_1		α_{A7} **(e-11)**	α_{A15} **(e-10)**	ϵ_1		α_{A7} **(e-11)**	α_{A15} **(e-10)**	ϵ_1			
200	10.02	5.2	0.112	0.97	12.6	3.66	0.08	0.98	2.51	3.87	0.14	0.94	0.91	0.91
800	8.17	4.79	0.182	0.98	3.21	2.97	0.22	0.94	2.22	3.43	0.20	0.93	1.00	0.91
1000	7.65	4.76	0.245	0.98	3.25	2.68	0.33	0.93	2.52	2.74	0.31	0.93	1.05	0.94
1200	7.9	4.8	0.27	0.98	3.01	2.55	0.43	0.92	1.86	2.92	0.48	0.91	1.13	0.99
1400	8.18	4.5	0.3	0.97	3.56	2.51	0.52	0.93	2.91	2.69	0.61	0.92	1.23	1.04

Table 4: Concurrent Application Power Models.

Freq. (MHz)	**Ferret+Bodytrack**			**MSE**	**Ferret+Fluidanimate**			**MSE**	**Volt. A7**	**Volt. A15**
	α_{A7}	α_{A15}	ϵ_1		α_{A7}	α_{A15}	ϵ_1			
200	1.331e-10	5.508e-10	0.088	0.99	7.041e-11	4.719e-10	0.133	0.99	0.91	0.91
800	3.635e-10	1.934e-09	0.1564	0.99	5.078e-11	4.719e-10	0.25	0.98	1.00	0.91
1000	4.73e-10	2.475e-09	0.1176	0.95	7.342e-11	4.632e-10	0.79	0.98	1.05	0.94
1200	4.587e-10	2.675e-09	0.1286	0.96	7.652e-11	4.632e-10	0.80	0.99	1.13	0.99
1400	7.646e-10	3.843e-09	0.1954	0.98	3.305e-12	4.632e-10	0.80	0.95	1.23	1.04

more big cores than little cores when increasing the frequency for the memory intensive applications. However, the experimental results of CPU intensive applications show that performance slightly decreases at 1400 MHz when we use more little cores. This shows that expected trends from theory may not be confirmed with experimental data in every case. The importance of the experimental approach must be recognized. From the data in the above figures, it is apparent that the performance increases linearly with the number of big and little cores. Considering that models for runtime use should in principle be as simple as possible, we hypothesize that the relation between IPC and the numbers of group of cores in heterogeneous systems can be approximated by the following expression:

$$IPC = \sum_{i=1}^{K} \alpha_i x_i + \epsilon_2(x), \tag{3}$$

where the first term ($\sum_{i=1}^{K} \alpha_i x_i$) represents the frequency-independent performance components that depend on architectural configuration, K is the total number of voltage islands (i.e. group of cores), and $\epsilon_2(x)$ is the error term. In the case of Exynos 5422 (i.e. Odroid big.LITTLE) (3) can be expressed as:

$$IPC = \alpha_1 x_1 + \alpha_2 x_2 + \epsilon_2, \tag{4}$$

where α_1, α_2, and ϵ_2 are coefficients which need to be determined for each operating frequency. Using MLR, their values have been obtained as shown in Table 5 and 6 .

3.1.3 Power Normalized Performance Model

Based on the power and performance outcomes, the experiments indicate that the optimal system configuration corresponds to the highest IPS/Watt value. Figure 5 presents the experimental data of IPS/Watt for different applications scenarios and architectural configurations. In the case of a single applications, *bodytrack* (CPU and memory intensive) exhibited the highest IPS/Watt. This can be explained by its high IPS (Figure 4) and the lowest power (Figure 3). The IPS/Watt of *bodytrack* shows an increasing trend with higher number of cores allocated; in the case of four little four big it has the maximum value of 3.8×10^9 IPS/Watt, when operating at 800 MHz. However, with higher frequencies the power consumption increases, which reduces its IPS/Watt. Similar observations can be made for the other single application scenarios.

In the case of concurrent applications the *bodytrack+ferret* (which is dominated by CPU intensive routines) shows higher IPS/Watt when compared with *fluidanimate+ferret* (which is a combination of CPU and memory intensive routines). To explain this further, Figure 6 depicts the individual and concurrent applications scenarios of *bodytrack* and *ferret*. As can be seen, when these two similar workloads are combined as a concurrent application it shows higher IPS/Watt with increasing number of cores.

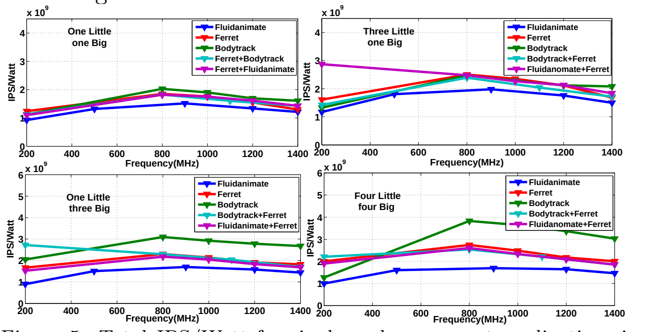


Figure 5: Total IPS/Watt for single and concurrent applications in different frequencies.

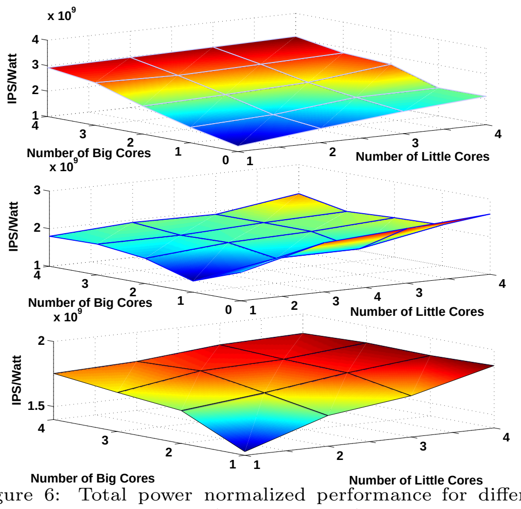


Figure 6: Total power normalized performance for different core-allocations at 1400 MHz for *bodytrack*, *ferret* and *bodytrack+ferret* applications.

The above models have been derived from the offline characterization data. In order to achieve the optimal mode during normal device operation, runtime adaptation has to be used instead. To simplify runtime adaptation MLR is used, which exploits the same set of runtime observations to derive/predict power and performances with reasonable accuracy. The power and performance expressions in (2) and (4) can be combined into a MLR expression as shown in the following:

$$\mathbf{D}^T = \mathbf{X}^T \mathbf{A} + \mathbf{E}^T, \tag{5}$$

Table 5: Single Application Performance Models.

Freq. (MHz)	ferret			MSE	bodytrack			MSE	fluidanimate			MSE
	α_{A7}	α_{A15}	ϵ_2		α_{A7}	α_{A15}	ϵ_2		α_{A7}	α_{A15}	ϵ_2	
200	0.43	0.87	0.15	0.99	0.23	0.97	0.14	0.99	0.20	3.87	0.6	0.88
800	0.46	0.94	0.14	0.99	0.23	0.97	0.14	0.99	0.21	3.43	0.52	0.91
1000	0.37	0.87	0.37	0.96	0.23	0.97	0.12	0.99	0.21	2.74	0.54	0.89
1200	0.43	0.91	0.16	0.99	0.25	0.98	0.15	0.99	0.21	2.92	0.52	0.89
1400	0.40	0.96	0.004	0.99	0.24	0.91	0.21	0.99	0.17	2.69	0.52	0.90

Table 6: Concurrent Application Performance Models.

Freq. (MHz)	Ferret+Bodytrack			MSE	Ferret+Fluidanimate			MSE
	α_{A7}	α_{A15}	ϵ_2		α_{A7}	α_{A15}	ϵ_2	
200	0.35	0.86	0.26	0.99	0.45	0.80	0.06	0.98
800	0.40	0.93	0.16	0.99	0.49	0.89	0.01	0.99
1000	0.38	0.93	0.132	0.98	0.43	0.86	0.13	0.99
1200	0.38	0.93	0.132	0.99	0.45	0.87	0.07	0.99
1400	0.34	0.93	0.11	0.99	0.41	0.84	0.10	0.99

where **D** is determinants vector formed of power and performance as shown in (2) and (4); **X** is a predictor vector for the given determinant; **A** is MLR coefficient vector for the given determinant ($a_i \in A$ where a_i is a coefficient expression in (2) including $V_i^2 f_i$ terms, i.e. $a_i = \alpha_i V_i^2 f_i$) and finally **E** is the error terms vector. Table 3-6 show the MLR determinant values for different operating configuration(including both single and concurrent applications scenarios).

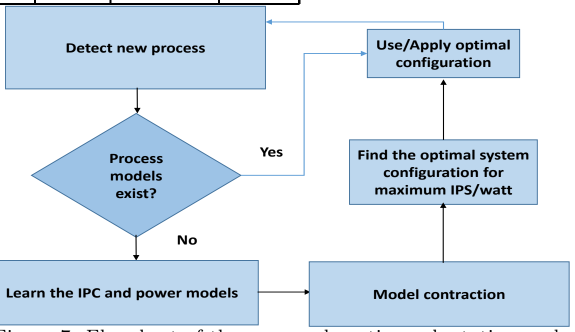
Algorithm 1 Runtime system adaptation algorithm to generate maximum IPS/Watt.

```
procedure MAXIMIZE IPS/WATT
    Input: Application scenario
    Output: System configuration (number of A15 cores, number of A7 cores, operating frequencies)
    Initialize: $\rho_{max} \leftarrow 0$
    for all $f \in [f_{min}, f_{max}]$ do
        Collect power/perf. readings for 200 ms
        Use MLR and (1) to find Power model for $f$
        Use MLR and (2) to find IPC model for $f$
        $IPS \leftarrow IPC \cdot f$
        $\rho \leftarrow IPS/P$
        if $\rho > \rho_{max}$ then $\rho_{max} \leftarrow \rho$
        end if
    end for
    Return: system config. corresponding to $\rho_{max}$
end procedure
```

3.2 Runtime Adaptation

The runtime adaptation is aimed at finding optimal task mapping and DVFS control in the presence of various types of concurrent applications (CPU or memory intensive). It is based on gradually building a library of the power and performance models, obtained during a normal device operation. The flowchart of the proposed approach is shown in Figure 7. The algorithm detects the arrival of a new application process and looks up its model in the library. If the model already exists in the library, it can be immediately used to compute the optimal point of operation, otherwise the learning process activates. As shown in Algorithm 1, the procedure collects power readings and performance counters for 200 ms time intervals swiping across the entire range of frequencies and uses MLR to derive the power and performance models according to Section 3.1. Once the models are obtained for all DVFS points, we can find an optimal system configuration by maximizing the IPS/Watt metric.

In the current version of the runtime control, the combined behaviour of concurrently running applications is captured by separate models. We are aware of the exponential growth when all combinations of applications are eventually stored in the library. This problem is addressed by compacting the models by similarity. The applications of the similar type tend to have similar values of $\alpha_1, \alpha_2, \epsilon_1, \epsilon_2$, hence these coefficients can be checked against a given threshold ε. The



Detect new process
Use/Apply optimal configuration
Process models exist?
Yes
Find the optimal system configuration for maximum IPS/watt
No
Learn the IPC and power models
Model contraction

Figure 7: Flowchart of the proposed runtime adaptation cycle.

models within the threshold are combined and stored as a single model.

4. RESULTS

The proposed runtime adaptation algorithm has been used in a number of examples with PARSEC benchmark programs running on the Odroid-XU3 platform, and the resulting IPS/Watt metric has been compared with common scheduling approaches such as those used by the Linux ondemand governor [20]. The example execution cases include running the three benchmark programs on their own in a sequential way and running *ferret* with either *bodytrack* and *fluidanimate* in two-app concurrent situations. In all these cases, using the proposed algorithm resulted in IPS/Watt improvements (up to 125% in the case of *ferret* application) as shown in Figure 8. From Figure 8 the following observations can be made. Firstly, it is clearly seen that the improvements in IPS/Watt for CPU and memory intensive applications are approximately same in the case of single applications. Secondly, in the case of concurrent applications the improvement in IPS/Watt is less than of the single application. Finally, the lowest improvement is recorded in the *bodytrack*-only case, where, compared to ondemand, an improvement of 26% can be observed.

The proposed runtime adaptation approach outperforms the ondemand governor with default Linux scheduler for the given applications. This can be explained as follows. The default Linux scheduler sets the number of parallel threads of these applications equal to the number of available cores. With the allocated cores the ondemand governor downscales the operating frequency when the CPU usage is lower than pre-defined thresholds points. However, when the CPU usage starts to increase it sets the operating frequency to the maximum points. Our approach allocates the number of parallel cores and their DVFS points based on the application workload types to ensure energy efficiency for all application scenarios.

The IPS/Watt improvements are recorded when the op-

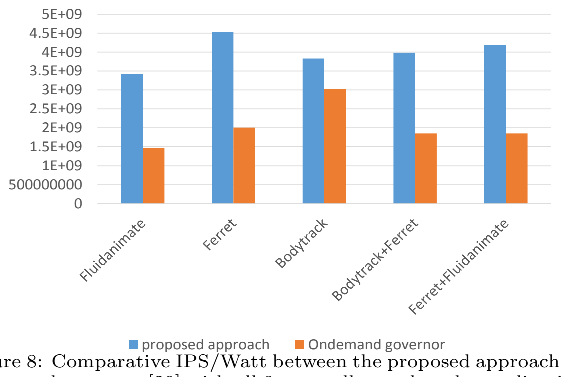


Figure 8: Comparative IPS/Watt between the proposed approach and ondemand governor [20] with all 8 cores allocated to the applications.

timal runtime configuration obtained with our adaptation approach is running. However, the adaptation process itself may cause overheads in two ways. Firstly, the frequency needs to be swept through a number of different values. During the sweep each frequency value is held 200 ms (see Algorithm 1) so that our performance counters can be used to collect the relevant data and different thread-to-core allocations tested. Whilst the application is not idle during this time, it is not always running at the optimal frequency and/or core allocation. Secondly, the application needs to be truly paused during the following operations: 1. when performance counters are extracted - this cost is very low, around 210 ns for each round of extraction (30 ns per performance counter and seven performance counters); 2. when the model is established - using floating point calculations, modeling should complete within 2 ms and using fixed-point, which we have not tried, should in principle reduce this cost further; 3. when the frequency is tuned - each frequency change costs 300 ns for the ARM platform experimented here; 4. when changing thread to core allocations - this is the highest application-pausing delay with experiments showing an average latency of 6.7 ms per change. Considering all this, the true application pausing intervals add up to very little time during each round of runtime adaptation and the main issue is the total time needed to sweep through enough frequencies and core allocations during which the application is mostly not run optimally. This adaptation, however, is only carried out once per new application execution state, i.e. when an application starts or ends. In most real-world situations such events happen relatively sparsely. For instance, if these events happen every 100 seconds and we sweep through five frequencies, only during 1% of the time the system may not be optimal. The total true application pause time (in the tens of milliseconds at most) is negligible.The presented approach focuses on power-aware performance optimization, the absolute value for performance is a major concern.

5. CONCLUSIONS

We demonstrated a novel runtime approach, capable of power-aware performance adaptation under sequential and concurrent application scenarios in heterogeneous many-core systems. The approach is based on power and performance models that can be obtained during runtime by multivariate linear regression based on low-complexity hypotheses of power and performance for a given operating frequency. The approach is extensively evaluated using PARSEC-3.0 benchmark suite running on the Odroid-XU3 heterogeneous platform. A selection of experimental results was presented to illustrate the kinds of tradeoffs in a variety of concurrent application scenarios, core allocations, and DVFS points, highlighting an improvement of power normalized performance which produced IPS/Watt improvements between 26% and 125% for a range of applications. It is expected that modern embedded and high-performance system designers will benefit from the proposed approach in terms of a systematic power-aware performance optimization under variable workload and application scenarios. Our future work will include investigating the scalability of the approach to more complex platforms and higher levels of concurrency.

Acknowledgement

This work is supported by the EPSRC (project PRiME, grant EP/K034448/1). The first author is also supported by studentship funding from the Ministry of Iraqi Higher Education and Scientific Research.

6. REFERENCES

[1] S. Kumar *et al.*, “Low overhead message passing for high performance many-core processors,” in *CANDAR*, pp. 345–351, Dec 2013.

[2] S. Borkar, “Thousand core chips: A technology perspective,” in *DAC*, pp. 746–749, 2007.

[3] K. Ma *et al.*, “Scalable power control for many-core architectures running multi-threaded applications,” in *Computer Architecture News*, vol. 39, ACM, 2011.

[4] Z. Xu *et al.*, “Exploring power-performance tradeoffs in database systems,” in *ICDE*, pp. 485–496, 2010.

[5] A. Rafiev *et al.*, “Studying the interplay of concurrency, performance, energy and reliability with archon – an architecture-open resource-driven cross-layer modelling framework,” in *ACSD*, pp. 122–131, 2014.

[6] H. Wong *et al.*, “The performance potential for single application heterogeneous systems,” in *8th Workshop on Duplicating, Deconstructing, and Debunking*, 2009.

[7] R. Shafik *et al.*, “Soft error-aware voltage scaling technique for power minimization in application-specific multiprocessor system-on-chip,” *JOLPE*, vol. 5, no. 2, pp. 145–156, 2009.

[8] M. Goraczko *et al.*, “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems,” in *DAC*, ACM, 2008.

[9] J. Luo *et al.*, “Power-efficient scheduling for heterogeneous distributed real-time embedded systems,” *IEEE TCAD*, vol. 26(6), pp. 1161–1170, 2007.

[10] L. K. Goh *et al.*, “Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems,” *IEEE TPD*, vol. 20, no. 1, pp. 1–12, 2009.

[11] S. Yang *et al.*, “Adaptive energy minimization of embedded heterogeneous systems using regression-based learning,” in *PATMOS*, pp. 103–110, IEEE, 2015.

[12] A. Nabina *et al.*, “Adaptive voltage scaling in a dynamically reconfigurable fpga-based platform,” *ACM TRETS*, vol. 5, no. 4, p. 20, 2012.

[13] V. Petrucci *et al.*, “Lucky scheduling for energy-efficient heterogeneous multi-core systems,” in *Proc. of USENIX HotPower*, pp. 7–7, 2012.

[14] B. Atitallah *et al.*, “An efficient framework for power-aware design of heterogeneous mpsoc,” *IEEE TII*, vol. 9, no. 1, pp. 487–501, 2013.

[15] M. Travers *et al.*, “Power-aware performance optimisation for for concurrent many-core applications,” in *ACSD*, 2016. (accepted).

[16] A. Wang and A. Chandrakasan, “A 180-mv subthreshold fft processor using a minimum energy design methodology,” *IEEE JSSC*, vol. 40, no. 1, pp. 310–319, 2005.

[17] C. Bienia and K. Li, “Parsec 2.0: A new benchmark suite for chip-multiprocessors,” in *PMBS*, June 2009.

[18] “Odroid XU3.” http://www.hardkernel.com/main/products.

[19] S. Skalicky *et al.*, “A parallelizing matlab compiler framework and run time for heterogeneous systems,” in *HPCC-CSS-ICESS*, pp. 232–237, IEEE, 2015.

[20] V. Pallipadi and A. Starikovskiy, “The ondemand governor,” in *Proc. of the Linux Symposium*, vol. 2, pp. 215–230, sn, 2006.