ANN Predicted Apps-Usage Aware Linux Scheduler for Asymmetrical Multi Cluster SoC

Hou Zhao Qi Rex[1, 2*], Jong Ching Chuen[1], Andreas Herkersdorf[2]

[1] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

[2] Department of Electrical and Computer Engineering, Technical University of Munich, Munich, Germany.

[*] Corresponding author. Tel.: +65 91362032; email: houz0001@e.ntu.edu.sg
Manuscript submitted January 11, 2016; accepted April 1, 2016.
doi: 10.17706/jsw.11.7.623-630

Abstract: This research introduces an improved Linux scheduler that models and manages several power dissipation problems based on user application usage pattern identified in mobile computing platform. The scheduler is developed and simulated in Android 5.0 with the Linux kernel 3.10 and 3.14. The scheduler monitors multiple system performance metrics at runtime, predicts power dissipation and future workload with an ANN computation block. To exploit DVFS capabilities in ARM asymmetrical SoC designs a three clustered CPU setup had been introduced in this study and the scheduler interacts with the Linux Load Balancing mechanism to passively achieve optimized operating frequency for each application thread.

Key words: Linux, Android, power management, scheduling, thread interference power, ANN, GTS, IKS**.**

1. Introduction

Computing industry has been through many great revolutions since the beginning. Hardware size reduces with each generation of newer fabrication process and lead to the mobile computing landscape of today. But the term of being mobile has put so much demand in portable battery power like never before. Neither hardware based power saving mechanisms such as Power Gating and DVFS [1] nor the software based scheduling techniques such as smart thread migration [2] alone is enough to keep the technology going on its track. Hence we had aimed to develop an approach to remedy the problem with both software and hardware measures. Like many software based power management technique [3], different usage patterns of users are identified and specific usage pattern based scheduling schemes are applied to reduce power dissipation. The theory behind [3] is the observation that in the same group of Apps, the sum of power consumption of Apps executed individually is less than the power dissipated by the Apps executed concurrently. The excessive amount of the power dissipated amongst concurrently running Apps is named as the Thread Interference Power (TIP). The amount of TIP is unique to a group of Apps and the group is named as the AppSet. The existing load balancing and scheduling mechanism of Linux operating system does not focus on this power dissipation pattern. In order to further improve the power dissipation reduction, we aimed to achieve hardware incorporation with the previous software based power management approach.

The main contribution of this research includes:

- Designed a model to monitor the multiple TIP power dissipations patterns and update the AppSet Table with ANN at runtime.

- Proposed Machine learning based methods to identified TIP dissipation problems.
- Assessed the efficiency of the proposed scheduling scheme in asymmetrical multi clustered SoC.

2. Related Works

The main stream of big.LITTLE SoCs are now equipped with the In-Kernel Switcher (IKS), each little core is grouped with a big core to form a virtual CPU pair. Only one core will be activated at one time and application thread metrics are monitored for the scheduler to make thread migration decisions between two cores in a virtual pair. Given the fact that mobile computing platforms normally execute less computation intensive applications that require the entire processor to run at its full capacity, the IKS has demonstrate its ability in managing the power dissipation effectively. However, with only half of the cores activated at any given instance, it is indeed a wasteful approach to utilize the computing power of a system. Hence we had researched into the idea of accommodating several small CPU clusters in a SoC so that as many cores can be affordably powered up together. There are many researches aimed to apply machine learning in Linux scheduler. In [4] machine learning has been applied to calculate the optimized time slice length in CFS scheduler and had achieved 5% of overall improvement in the turnaround time. In [5] a Multilayered Perceptron (MLP) scheduler has been developed with the aid of M5 learning tool, and had achieved up to 7% of improvement compared to the state-of-the-art schedulers. However, the existing literatures have not tailored machine learning functions to suit their schedulers and require off-line learning for initialization. In contrast our scheduler developed dedicated machine learning function blocks to minimize computation overheads and kept all learning process on-line to maximize its compatibility.

3. DVFS and Asymmetrical Clusters

3.1 DVFS Incorporation

DVFS is used to assign each CPU core different voltage islands to operate at different frequencies for different workload. It is a common method to reduce the power dissipation of CPU cores at runtime. The preset voltage and frequency pairs for a CPU core to operate in a recommended condition is named as the operating points In the practice of conventional design flow under parallel computing environment, software are optimized to achieve maximum throughput at all cost with little consideration of scaling workload under different operating frequencies. While in fact different the optimized operating point of different application or even different threads of the same application can vary due to many factors (I/O constraint, Triggering Event, code bottle neck). Such design practice costs DVFS inefficiencies in energy reduction as threads from every application are likely to game for CPU resources for maximized computing throughput [6]. The ideal way to solve this problem is through the identification of optimum operating frequency factor individually for each thread. This approach may impact the system heavily in two aspects: 1. intensive computation due to the monitoring and tracing of each instruction dispatched is required to be logged for accurate identification; 2. excessive overheads created in the process to scale operating points and passive thread migration between CPU cores. To minimize the overhead, native Linux scheduler adopted the Coarse DVFS with governors adjusting the CPU operating points with limited preset values. In contrast to the Fine-grained DVFS methods had been developed [7] to save computation energy by allocating each CPU core a different operating point with finer frequency step size and time slice. However the resulted overhead impacting the system can be as high as 13% with an average of 10% energy reduction. In order to increase the DVFS efficiencies of the system without increasing the overhead drastically, we had worked towards implementing a passive DVFS approach with thread awareness. Our previously proposed power management schemes were implemented solely based on optimizing thread sequence in Linux run queue. To further reduce power dissipation we had implemented a passive DVFS

incorporation by modifying the Linux Load Balancing to have thread awareness. In native Linux Load Balancing in CFS, threads are allocated to the run queue in a round robin manner without analyzing TIP dissipated in the switching of different threads. Which resulted in thread migration due to unwanted bottleneck and cache coherence problems. The proposed re_scheduler monitors the operating point of each CPU core and calculate its Core Capacity (CC). Computation Budget (CB) of each thread is computed based on their historical values recorded in the AppSet table. Threads are then allocated by matching the CC of each core and the CB of each thread. Asymmetrical CPU architectures has proven to be effective in reducing the DVFS overheads [8] by fixing different frequencies to each cluster and allocating threads to different clusters to satisfy the operating point requirements. The big.LITTLE architecture has shown the competence of asymmetrical over symmetrical CPU in terms of computation throughput at lower power consumption. And in the mobile platforms such as the Android, there are only a handful of parallel threads per application due to its UI thread and AsyncTask mechanism. Hence, we explored the possibility to extend the capacity of asymmetrical designs by having more than two different core types and grouped into smaller cluster that consists only two core as shown in Fig. 1. Such a design is aimed to grant the system with more choices of operating points at runtime, which is equivalent to have a higher degree of freedom in choosing the suitable cluster and without generating overheads from adjusting operating points. The computation intensive threads can be allocated to the faster cores and less demanding threads are allocated on low frequency clusters to achieve better power saving. As shown in Fig. 1. the proposed scheduler is tested in the GEM5 [9] simulated environment where an asymmetrical CPU with Cortex A35, A53 and A57 core clusters with two core each was constructed, the operating system is ICS 5.0 Android built with Linux kernel 3.10 and 3.14.

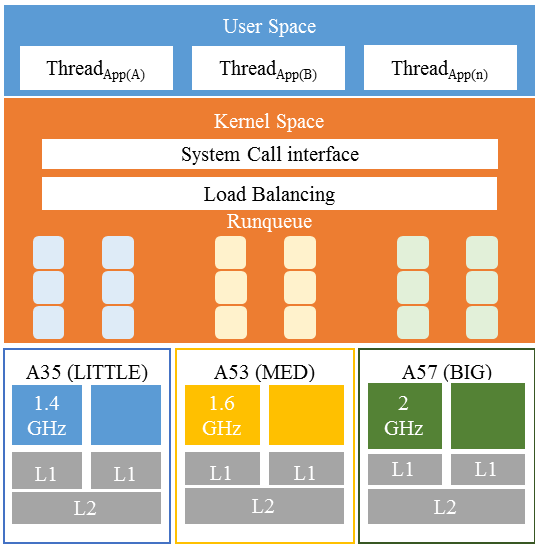

Fig. 1. Big. Med.Little core cluster setup.

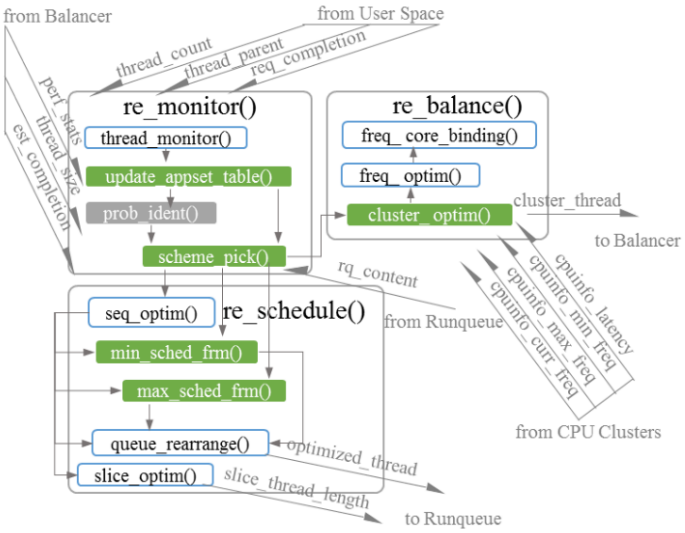

Fig. 2. Control blocks of re_scheduler.

3.2. Scheduler Function Blocks and Power Performance Curve

As shown in Fig. 2, the re_scheduler is constructed with three blocks, *re_monitor()*, *re_balance()* and *re_schedule()*. The *re_monitor()* is for monitoring thread information gathered from user space such as the number of child thread of an application, the process id of the parent thread and the required completion time of a thread. *perf_stat()* function is called to pull system metrics such as cache miss rate and branch miss rate are retrieved with perf stat, and used to compute the AppSet table values before the instructions of each thread are balanced into the respective run queue. The updated the AppSet table is used to

formulate power dissipation models for each individual thread and calculate the *Computation Budget* (CB) of each thread. Green blocks in Fig. 2 require call the function block *ann_compute()* for Artificial Neural Network computation. Details about the ANN computation block is discussed in the Section IV of this paper. A scheduling scheme will be picked in the *re_monitor()* and propagated to *re_balance()* and *re_schedule()*. In *re_balance()* cupinfo from each core in every cluster are assessed to compute *Core Capacity* (CC) of each core, optimized workload balancing decisions and output the optimized CPU core and application thread pair to carry out Load Balancing. In *re_schedule()* the Linux run queque content are analyzed to compute the optimized schedulingsequence and the time slice length based on the predicted TIP dissipation. The optimized scheduling sequence will guide the run queque to dispatch the thread accordingly with the optimized time slice length. For re_scheduler an accurate CPU core modeling is important to compute CC and CB to make operating point adjustments for each core. Initially the original Linux Load Balancing rule applies as the CB values are incomplete for each thread, a fixed set of operating points and CC to CB mappings are made based on fairness. As the AppSet table builds up CB values can be calculated accurately, it is then possible to map a thread with its CB to a core with its predicted CC. the process of modeling each core and thread can be redundant, hence the ANN block is employed to ensure the mapping accuracy and efficiency.

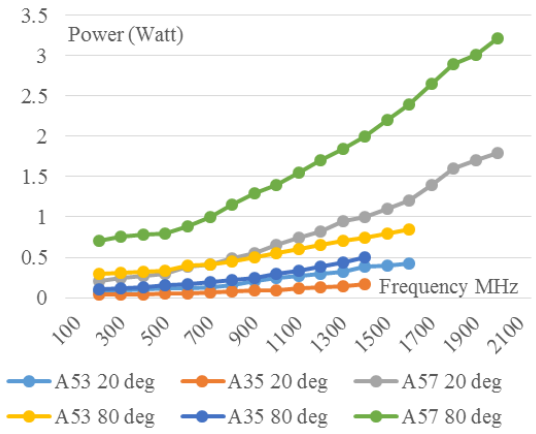

Fig. 3. Power dissipation and frequency of different clusters at different temperature.

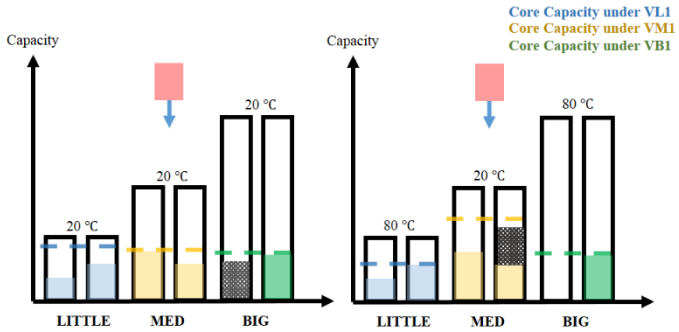

Fig. 4. Operating point balancing at different temperature.

As shown in Fig. 3 the power dissipation pattern of different ARM Cortex Core varies with the working frequency and the temperature of the CPU plays a vital role while modeling the CPU *Core Capacity*. According to the power performance curve, at different temperature level the smaller cores do not always consume less power comparing to the larger cores. For example, the A57 cores dissipates more power than A53 cores if the operating temperature is the same, but A57 cores can beat A53 cores power to performance rating at low temperature, this enables the possibility to explore the unique power

performance response of different cores at different temperature to maximize the computing capability and minimize the power dissipation at the same time. In Fig. 4. the diagram on the left shows that at the same temperature, the incoming thread is balanced to the Big core by considering the CC of the cores, CB of the threads and the cost of increasing the operating point of a core with lower power performance rating. The diagram on the right demonstrates the same scenario but each core operates at a different temperature, the thread thread is allocated to the Med core with insufficient CC instead of the Big core with enough CC and resulted in ramping up the operating point of the Med core. It is because of the cost of increasing the operating point is lower than running it on the Big core with high temperature.

4. Machine Learning and Prediction

4.1. ANN Based Prediction

It is very important to ensure the accuracy of the result obtained from the predicted performance metrics and it is also crucial to make sure that the prediction will take little effort to derive the result. Our previous scheduler predicts based on computing each related elements in the AppSet table statistically and requires to walk through the entire AppSet table to compute each optimization. Each entry of AppSet table contains information (TIP, thread size, time slice size, estimated completion etc.) about an AppSet which can be a single app or a combination of threads from different Apps. In the current design the AppSet table logs App combinations up to three Apps, hence it could be very inefficient if the entire table is required to be accessed every scheduling cycle. The current re_scheduler improved the computation with machine learning method to increase the efficiency by reducing the feature space. Upon initialization the computation overhead of the ANN is relatively expensive compared to the statistical generated power dissipation pattern and management scheme matching mechanism. The ANN computation block named *ann_compute()* is shown in Fig. 5. The functions in re_scheduler control blocks will call *ann_compute()* to compute for dedicatedly. It establishes a three layered MLP (Multilayer Perceptron) upon taking receiving parameters from the caller function such as the step size and neuron number. The input signals are processed through each layer with feed-forward operation and sigmoid activation function. And sum squared errors are computed and back-propagated to neural network to update the weights in each layer. Functions in the control block that calls *ann_compute()* are having their computation matrices isolated from each other, and the step size can be adjusted at runtime to control the learning rate for optimized neural net convergence rate.

4.2. ANN Block Application

Fig. 6. shows two of numerous re_scheduler functions that utilizes the *ann_compute()* block, the prediction of next incoming thread at run queue and the computation of AppSet table. To compute the next incoming thread (T_{next}), a matrix contents the historical run queue thread sequences is set as the input of the *ann_compute()*, the number of the input layer are set to be equivalent to the number of thread entries in the AppSet table. The matrix logs the thread composition with a three thread window, all threads listed in the table are logged as the neurons of the first layer, and the second thread coming into the run queue constructs the neurons in the second or the hidden layer. The hidden layer is a reduced set of neurons where only previous recorded thread combinations contribute to it. And the output layer carries as many neurons as the input layer. The initial weights are set as the normalized occurrence of each thread combination in run queue. The number of neurons of the MLP are updated and recomputed each time there is a new entry longed in the AppSet table. The weightage of each neuron are then used to predict $T_{next}()$. The *update_appset_table()* calls *ann_compute()* to establish the AppSet table, where each entry of the table has an independent neural network. The input layer has a number of neurons equivalent to the system metrics

monitored by the *thread_monitor()* and the same amount of neurons in the output layer, the number of neurons in the hidden layer equals to the number of threads in the entry to simulate the interference effects between each thread of the entry. The input layer feeds the accumulated performance metrics of all threads in the entry and compare the output with *perf_stat()* reading upon completed execution of the threads in the entry. The weights are updated according to the sum squared error computed. The weight of the neuron network is outputted as the TIP model for the AppSet table. The AppSet table entries and the T_{next} are used to compute trigger parameters for each scheduling schemes proposed in re_scheduler. In Hold scheme the AppSet table TIP values are used to determine the optimized scheduling sequence to avoid excessive TIP in the run queue and the T_{next} is used to compute the minimum scheduling period of the scheme to avoid excessive overhead to impact the performance. In the Bind scheme the AppSet table TIP values are used to interleave thread with high TIP into different CPU cores to avoid excessive power dissipation, and to calculate the possible CB of each thread. The thread and core pair will be then determined based on the CB and CC calculated based on the cpuinfo gathered from the CPU cores. Similarly the T_{next} is used to compute the minimum and maximum scheduling period of the scheme. The Super scheme reads the average thread execution time and T_{next} to determine how much time slice length can the other threads in run queue spare for the target thread to upsize its time slice.

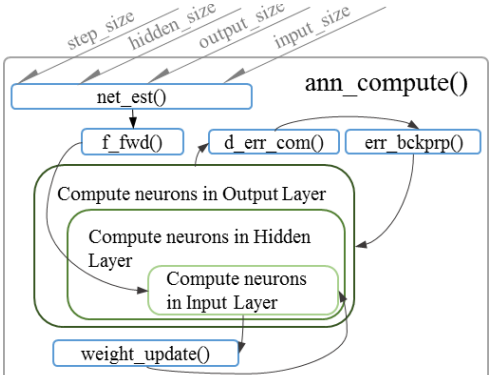

Fig. 5. Artificial neural network computation block.

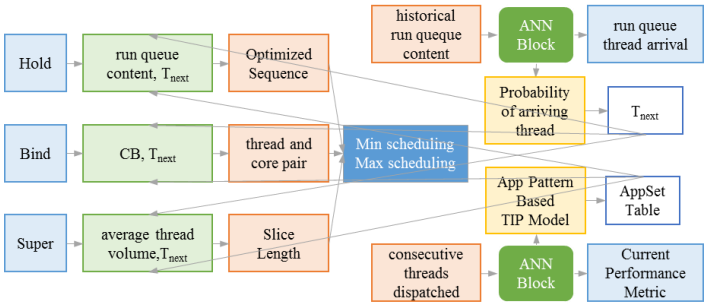

Fig. 6. ANN computation utilization in re_scheduler.

5. Experiment and Result

The experiment is conducted in a simulated environment with GEM5 simulator and McPat [10] to provide accurate power dissipation of CPU cores under different thermal configuration. The environment consists of three 28 nm clusters with two cores each: Cortex A57 (out-of-order) at 2 GHz, A53 (in-order) at 1.6 GHz and A35 (in-order) at 1.4 GHz. 32 KB of L1 cache for each core and 32 KB of L2 cache for each cluster. We created simulation test scenarios with Moby Bench [11] to assess the re_scheduler against the GTS under Linux Kernel 3.10 and the IKS under Linux Kernel 3.14.

Each scenario consists of three set of test cases: 1. Individual App; 2. Apps executing in pairs; 3. three Apps executing simultaneously. There are 10 Apps in the benchmark, hence there are 10 setups in test case 1; 5 setups in test case 2; and 4 setups in test case 3. Each test setup consists of ten runs and the workload of each Apps are assigned randomly. Table 1. shows the average result of the re_scheduler over the GTS and IKS scheduler. The proposed scheduler is able to gain better power reduction compared to the native scheduler in all three scenarios.

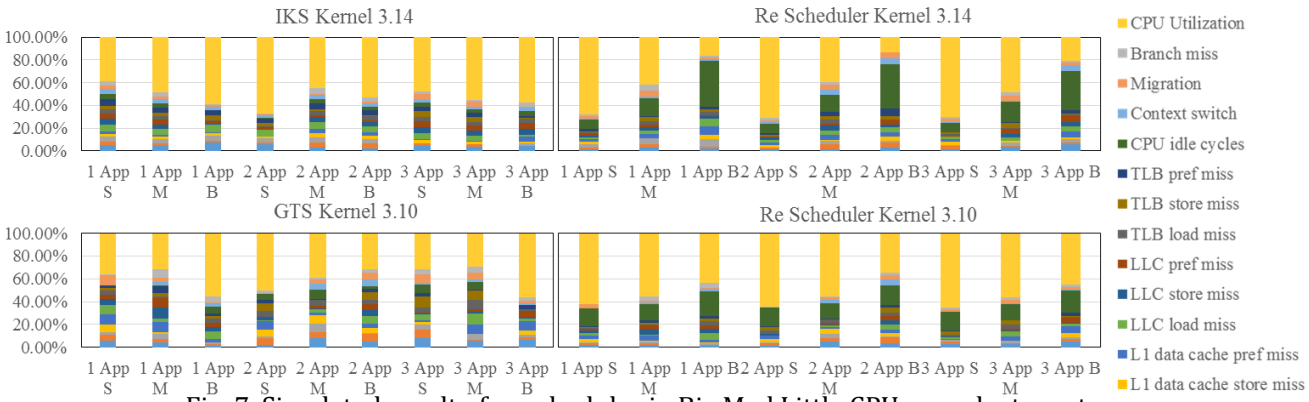

Fig. 7. Simulated result of re scheduler in Big.Med.Little CPU core cluster setup.

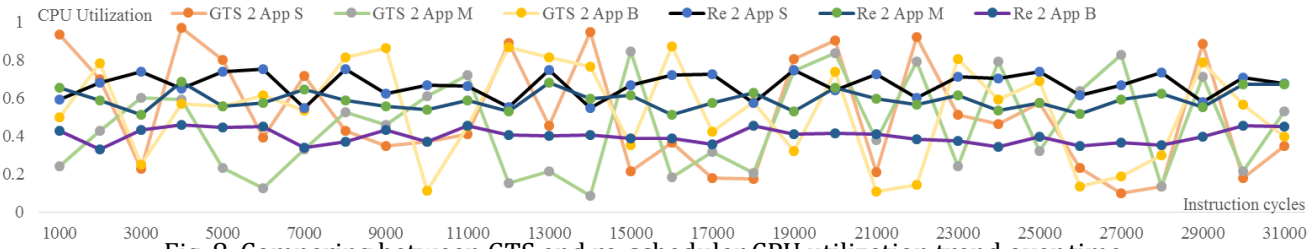

Fig. 8. Comparing between GTS and re_scheduler CPU utilization trend over time.

Table 1. Average Power Reduction of Proposed Scheduler

Cases	GTS (W)	Re (W)	Power Reduced	IKS (W)	Re (W)	Power Reduced
1 App	1.72	1.489	13.43%	1.57	1.43	8.92%
2 Apps	2.2	1.816	17.45%	1.87	1.78	4.81%
3 Apps	3.92	3.045	22.32%	3.37	3.1	8.01%

As shown in Fig. 7, the re_scheduler assigns the thread to little core more often than the GTS and KIS where a fair load balancing scheme is in charge of assigning tasks to run queues. Our scheduler has achieved in less thread migration, more CPU idle cycles and reduced power dissipation. Fig. 8. shows that CPU utilization of re_scheduler is more smoothly distributed over time compared to the GTS scheme which is a result of the accurate prediction of workload at runtime. The re_scheduler minimized thread migration counts in Kernel 3.14 by allocating most of the threads into Small cores and avoiding the Big cores, this approach has also generated TIP in maintaining cache coherency. Hence the power reduction result from 3.14 is not as good as what has been achieved in 3.10 despite of the better overall CPU utilization.

6. Conclusion

The re_scheduler has demonstrated the ability in better managing a multi clustered asymmetric SoC compared to the two native Android schedulers. It has also demonstrated the capability of extending the user app usage driven scheduler with machine learning function blocks and the passive DVFS incorporation. In the future we will focus on better the learning process with object optimization measures.

References

[1] Liang, X., Zhu, Y., Yang, J., Ye, J., & Gu, Z. (2010). Implementing a thermal-aware scheduler in linux kernel on a multi-core processor. *The Computer Journal, 53(7)*.

[2] David, T., Azimi, R., & Michael, S. (2007). Thread clustering: Sharing-aware scheduling on SMP-CMP-SMT multiprocessors. *ACM SIGOPS Operating Systems Review*, *41(3),* 47-58.

[3] Hou, Z. Q. R., Jong, C. C., & Andreas, H. (2014). Apps-usage driven energy management for multicore mobile computing systems. *Proceedings of 2014 14th International Symposium on Integrated Circuits (ISIC)* (pp. 472-475). Singapore: IEEE.

[4] Atul, N., & Kishore, K. P. (2005). Applying machine learning techniques to improve linux process scheduling. *Proceedings of TENCON 2005 2005 IEEE Region 10 (*pp. 1-6). Melbourne: IEEE.

[5] George, A., Tshilidzi, M., & Fulufhelo, V. N. (2013). Multicore scheduling based on learning from optimization models. *Int. J. Innovative Comput. Inform. Control, 9(4),* 1511-1522.

[6] Andrew, K., Seokhyeong, K., Ravindra, K., & John, S. (2013). Enhancing the efficiency of energy-constrained DVFS designs. *IEEE Transactions on VLSI Systems*, *21(10),* 1769-1782.

[7] Stijn, E., & Lieven, E. (2011). Fine-grained dvfs using on-chip regulators. *ACM Transactions on Architecture and Code Optimization (TACO), 8(1),* 1.

[8] Andrew, L., Shruti, P., Reetuparna, D., *et al*. (2014). Heterogeneous microarchitectures trump voltage scaling for low-power cores. *Proceedings of ACM the 23rd PACT* (pp. 237-250).

[9] Nathan, B., Bradford, B., Gabriel, B., Steven, K. R., Ali, S., Arkaprava, B., *et al*. (2011). The gem5 simulator. In *ACM SIGARCH Computer Architecture News, 39(2),* 1-7.

[10] Sheng, L., Jung, H. A., Richard, D. S., *et al*. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. *Proceedings of the 42nd Annual IEEE/ACM MICRO*, (pp. 469-480).

[11] Yongbing, H., Zhongbin, Z., Mingyu, C., & Lixin, Z. (March 2014). Moby: A mobile benchmark suite for architectural simulators. *IEEE ISPASS.*

Hou Zhao Qi Rex obtained his B.Eng in electronic and electrical engineering (EEE) from Nanayang Technological University (NTU). He is now doing his PhD in EEE in NTU under supervision of Professor Jong and Professor Herkersdorf. Mr Hou's research area is focused on the power management of the mobile computing platform.

Jong Ching Chuen obtained his BSc(Eng) and PhD in electronic engineering from Queen Mary College, University of London, U.K. He is now an associate professor in the School of Electrical & Electronic Engineering. Assoc Prof. Dr. Jong is a chartered engineer (CEng), a member of the Institution of Electrical Engineers (IEE) and a member of the British Computer Society (BCS).

Andreas Herkersdorf did his doctorate in 1991 at the Swiss Federal Institute of Technology Zurich. He then joined the research team at IBM Research – Zurich in the Communication Systems Department. Since 2003, Prof. Herkersdorf has held the chair of Integrated Systems at TUM.