On-Chip Intelligent Dynamic Frequency Scaling for Real-Time Systems

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

24-10-2020 / 27-10-2020

CITATION

Sharma, Prachi; Bera, Arkid; Gupta, Anu (2020): On-Chip Intelligent Dynamic Frequency Scaling for Real-Time Systems. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.13139021.v1>

DOI

[10.36227/techrxiv.13139021.v1](https://doi.org/10.36227/techrxiv.13139021.v1)

On-Chip Intelligent Dynamic Frequency Scaling for Real-Time Systems

An ANN-based DVFS Methodology for both CPU and Memory-Intensive Processes

Arkid Kalyan Bera, Prachi Sharma
· Dr. Anu Gupta

Received: date / Accepted: date

Abstract To curb redundant power consumption in portable embedded and real-time applications, processors are equipped with various Dynamic Voltage and Frequency Scaling (DVFS) techniques. The accuracy of the prediction of the operating frequency of any such technique determines how power-efficient it makes a processor for a variety of programs and users. But, in the recent techniques, the focus has been too much on saving power, thus, ignoring the user-satisfaction metric, i.e. performance. The DVFS technique used to save power, in turn, introduces unwanted latency due to the high complexity of the algorithm. Also, many of the modern DVFS techniques provide feedback manually triggered by the user to change the frequency to conserve energy efficiently, thus, further increasing the reaction time. In this paper, we implement a novel Artificial Neural Networks-driven frequency scaling methodology, which makes it possible to save power and boost performance at the same time, implicitly i.e. without any feedback from the user. Also, to make the system more inclusive concerning the kinds of processes run on it, we trained the ANN not only for CPU-intensive programs but also on the ones that are more memory-bound, i.e. which have frequent memory accesses during its average CPU cycle. The proposed technique has been evaluated on Intel i7-4720HQ Haswell processor and has shown performance boost by up to 20%, SoC power savings up to 16%, and Performance per Watt improvement by up to 30%, as compared to the existing DVFS technique. An open-source memory-intensive

Arkid Bera
BITS Pilani, Pilani, Rajasthan India
E-mail: f2015271p@alumni.bits-pilani.ac.in

Prachi Sharma
BITS Pilani, Pilani, Rajasthan, India
E-mail: f2015278p@alumni.bits-pilani.ac.in

Dr. Anu Gupta
BITS Pilani, Pilani, Rajasthan, India
E-mail: anug@pilani.bits-pilani.ac.in

benchmark kit called Mibench was used to verify the utility of the suggested technique.

Keywords Dynamic Frequency Scaling · Artificial Neural Network · User-CPU profile · Memory Access Rate

1 Introduction

In this era where more and more handy and mobile devices are replacing conventional electronics with wires and heavy batteries, energy conservation becomes significant to reduce unnecessary power loss and heat dissipation, thus, in a nutshell, improving battery life. But, in the efforts to improve the same, the focus has shifted away from user satisfaction, i.e. not compromising the performance required by a user for comfortable programming. As the demand for high-performance CPUs increases, the chip-making industry is competing for leadership by a mere 1-2 percent margins in frequencies of operation concerning their counterparts. Thus, conscious efforts have to be put in so that we do not face a big trade-off between power and performance while using the Dynamic Voltage and Frequency Scaling(DVFS) schemes. In this paper, we propose an on-chip machine-learning-based frequency selection method that intelligently predicts the performance demand based on the CPU utilization and Memory Access Rate[1] (MAR is the ratio of cache misses to the total number of instructions executed), while achieving significant power savings at the same time. Also, most of the earlier work in this sphere focuses on either CPU-intensive or memory-intensive operations, but not both. But in reality, when the user operates a device, both kinds of applications are accessed. Thus, we propose a technique which addresses a more practical need, wherein, while the CPU utilization tells the ANN the engagement of the user with the processor, the MAR controls the frequency prediction for programs in which the memory is accessed more, like in image and video processing or ones with high IO activity[12].

In the previous mathematical models of DVFS, a crucial issue has been the time-complexity. In [2], Rangan et al. first posited that the sampling intervals of Operating System (OS) scheduler are in the range of milliseconds, while the computational requirements may be in nanoseconds due to CPU events like cache misses. Thus, the existing DVFS techniques often fail to respond to such granular program behavior. The increased time-complexity delays the frequency selection decision, thus, reducing its effectiveness. Also, in some previous implementations, the frequency scaling is done based on explicit user input to gauge user satisfaction. This delays the decision further. Thus, to address the issues, a lightweight artificial neural network named Kohonen Self-Organizing Map is used, for a quicker selection of frequency. In the proposed implementation, the pre-trained network is implemented on-chip, thus, reducing latency and power overhead considerably. Since no communication of data to the cloud is required, it enables much faster inferencing on-site. Our approach is based on breaking down the frequency scaling into a classification

problem, i.e. classify the user-behavior into appropriate frequency levels, which give the highest performance per watt.

The proposed technique has been implemented and evaluated on the Intel i7-4720HQ Haswell processor, but can be extended to any computer or edge processor. It has been compared with the existing DVFS technique of the processor (on-demand governor in a Linux-based OS) on the CPU benchmark named MiBench. The experimental results suggest that the proposed frequency selection algorithm can achieve up to 20% performance boost and save up to up to 16% SoC power. As compared to the existing DVFS implementation, the proposed technique can give up to 30% Performance per Watt improvement. The paper is structured as follows. Section 2 describes the related work in this field, while section 3 explains the proposed algorithm. Section 4 focuses on the on-chip implementation of the neural network. Section 5 contains the experiment and verification of the methodology. Finally, Section 6 concludes our analysis and hints at the future work on this project.

2 Related Works

ANN-based DVFS techniques proposed in the past have focused mainly on just power savings, without taking performance impact into account. Also, these implementations primarily cater to CPU-intensive operations. In [3], Lahiri et al. proposed a back-propagation neural network model that predicts the future computational load to select frequency intelligently. This method delivered 20% power savings, but no performance boost. To optimize expected energy per user-instruction, Moeng, and Melhem [4] employed a decision tree algorithm. Qingchen [5] proposed a deep Q-learning model to form a hybrid of various DVFS technologies for saving power. Jung and Pedram [6] have implemented a supervised-learning-based DVFS technique to predict the system performance state based on some predefined input features, which is then used to judge the optimal power policy. Tesauro et al. [7] have proposed a reinforcement learning approach that saves power at the cost of minor performance loss.

Further, there have been certain implementations that focus only on MAR [1], catering to saving only power for memory-intensive activities. In [12], a feedback-based DVFS scheme was proposed for memory-bound applications that monitor the cache misses running a software feedback loop. Our proposed methodology aims at reducing the latency caused due to using their algorithmic approach by using Artificial Intelligence.

Thus, related works in this sphere primarily focus on saving power (at the cost of performance) of either CPU or memory-intensive applications at a time, but not both. Also in this paper, we propose a very lightweight unsupervised-learning-based on-chip frequency-selection technique, which has negligible power overhead and latency. Thus, it boosts performance boost and saves power simultaneously. Also, the selected neural network is capable of taking multiple inputs without increasing much complexity, thus, able to take both CPU uti-

lization and Memory Access Rate (MAR) into account.

Also, some more previous work in this sphere has focused on taking explicit input from the user for expression of user-satisfaction, for example, HAPPE [8] (Human and Application- Driven Frequency Scaling for Processor Power Efficiency). In the proposed implementation, we make the system more implicit by training the ANN and then making it independent of any user inputs in the future, thereby reducing the human error and bias in the prediction, while also reducing the delay in decision-making.

3 CPU and Memory-aware Frequency Scaling Approach

In the proposed implementation, we have considered the CPU Utilization and MAR to be the impacting factors. CPU utilization by a process or program can not only vary due to its complexity but also upon how the user uses it. A simple Microsoft Word document, for example, is used for varying lengths of time and with varying speeds dependent on the user's traits. Similarly, monitoring cache misses is important to determine the frequency of operation in practical applications where memory and IO accesses are frequent. Both these parameters form the basis of prediction in our methodology.

Now, the raw data required to train this network is unlabeled, making a linear relationship between input and output impossible. Thus, to find patterns in the raw, unlabeled data, and cluster them, we needed to choose an unsupervised classification algorithm. Kohonen Self Organizing Map[9] fits the criteria, wherein, the inputs proximal to each other are mapped to nearby or the same clusters. The network and its applications are further explained in [15].

There are two stages in the proposed algorithm – offline training and online inference.

3.1 Offline training

The training of the network takes place offline, i.e. not on-chip. Figure 1 describes the training algorithm in detail. In the figure, 'lr' denotes learning rate, lr_limit denotes the minimum value up to which the learning rate can reduce before the training stops, and alpha is the damping factor for the learning rate.

The entire range of user-behavior, i.e. CPU utilization and MAR values (lowest possible to highest possible), is fed into the network as 2-dimensional input for offline training (on a software). In our experiment, 500 data points covering all kinds of utilization/user-behavior are used as training data. Figure 2 shows the same.

These input data points can be classified into n number of clusters, based on their values. Here, we have taken 48 clusters and each cluster assigned a 2-dimensional weight value (one dimension each for CPU utilization and MAR) randomly before the commencement of training.

Fig. 1: Kohonen Network Training algorithm

Fig. 2: Kohonen Network Training Data points

At the commencement of the training phase, each data point is taken one at a time, its Euclidean distance compared with all the 2-dimensional weights, and the cluster whose weight is closest to the input is chosen as the winning cluster/node, and a fraction of the input data point added to the weight. This

process is repeated for all the data-points. On completion of training, it is observed that the input user-behaviors (training data points) have been classified into 48 different groups based on the amount of CPU and memory usage, with each group having a final weight value. Finally, after clustering, each kind of behavior (cluster) is assigned a frequency level.

Note, the assignment of frequency level does not involve any intelligent algorithm. 18 equally-spaced frequency levels are chosen from the 900MHz to 2.6GHz, and assigned to the clusters in such a way that the cluster with the weight value with the highest CPU utilization and lowest MAR value parameter is assigned the highest CPU frequency, and so on. Figure 3 shows the allotment of the clusters. This is because we are aware that CPU utilization is directly proportional to CPU operating frequency, and from [1], we gather the knowledge that MAR is inversely proportional to CPU speed, i.e. if MAR increases, we need to reduce CPU frequency to gain maximum performance per watt.

Fig. 3: Cluster distribution and frequency allotment

Thus, during the training phase, we generate 48 different clusters, each representing a specific level of CPU and memory usage (i.e. each cluster or node having a final weight representing the cluster's CPU utilization and MAR) and an appropriate frequency level which the CPU should run at, for the particular behavior. Once the training is over and the software training algorithm determines the trained weight values for each cluster, the corresponding weight registers on the chip are set with the correct values, thus, setting up the circuit for on-chip inference. Figure 3 shows the assignment of clusters.

3.2 Online Inference

Once training is done, the inference circuit is deployed on the SoC, thus, requiring no other communication of the neural network with the cloud, etc. Now, the aim is to select one of the 18 frequency levels based on user behavior. The network takes the CPU-utilization and MAR of the processor as 2-D input in real-time. It then compares the input with the weight of each cluster, and the input is grouped into the cluster whose weight value is closest to it (i.e. winning node/cluster), and finally selecting the frequency assigned to the winning cluster as the new CPU frequency. Thus, intelligent frequency-selection is achieved based on the CPU and memory-intensive applications accessed by the user. Figure 4 further explains the inference.

3.3 On-Chip Implementation of Inference

To measure the hardware cost, the inference circuit has been implemented on Verilog and netlist generated using Synopsys Design Compiler. Figure 5 shows the block-level circuit implementation, wherein the Euclidean distance between input and each weight is calculated. The comparator is used to compare the distances and deliver a 48-bit output. If n^{th} cluster is the winning node, the n^{th} bit from MSB is set to 1, while others to 0. Also, in the figure, fp_tc is used to convert 2's complement of the input, fp_adder is a 32-bit 2-input floating-point adder, and fp_mult is a 32-bit 2-input floating-point multiplier. Overhead cost data is mentioned in the following section.

4 Experiments and Results

In the Linux Kernel, there is a modularized interface that aids in scaling CPU frequency, known as the CPUfreq subsystem. In Linux, we have a policy manager called the governor who controls the CPU frequency through the CPUfreq interface.

The various kernel-level governors supported by Linux for CPU frequency management are as follows – ‘Performance’, ‘Ondemand’, and ‘Userspace’. The performance governor always selects the highest frequency throughput, whereas the Ondemand governor selects frequency based on CPU usage. By default, the Ondemand governor is used for DVFS in Linux. In this paper, we compare the proposed implementation with the Ondemand governor. The proposed governor can access the CPU utilization and MAR and set the frequency directly through the CPUfreq subsystem. The experiment has been carried out on an Intel i7-4720HQ Haswell processor with a 2.6GHz base frequency.

The latency overheads of the existing and proposed frequency selection schemes are 50 μ s[10] and 0.13 μ s, respectively. Also, the measured power overhead of the proposed implementation is 0.22 W, which is meager as compared to 47

Fig. 4: Kohonen Network Inference algorithm

W, the package Thermal Design Power(TDP) of Intel i7-4720HQ Haswell processor, thus, further strengthening the argument that our implementation is very lightweight.

For measuring the performance improvement, MiBench, a popular CPU and memory-intensive benchmark for Linux systems have been used. In [18], the experiment is based on another Linux benchmark called SysBench, which

Fig. 5: On-chip inference circuit

was giving a 47% performance boost with that implementation. But, it does not cater to processes having high memory access times, providing unnecessary fluctuations in frequency and inaccurate predictions for those programs. To verify our system for all flavours of programs, we use Mibench in this paper which gives us correct estimates of frequency for both memory-intensive and/or processor-intensive programs. The following sub-tests of Mibench are chosen: bitcount, fft, susan, patricia, qsort, typeset, crc32, sha, and stringsearch. Sub-test 'bitcount' is the most CPU-intensive workload among these, with negligible memory activity, while stringsearch is the most memory-intensive [11]. For our experiment, we have simulated five real-life user-behaviors, ranging from extensive use of memory-bound applications to that of CPU-intensive ones. E.g. User1 executes highly memory-intensive applications, while User5 is focused on running CPU-intensive programs. Each of these behaviors is mimicked by a subset of the above-mentioned benchmarks. Table 1 shows the same.

Figure 6 shows a comparison of CPU frequency of the proposed implementation concerning the existing implementation, for each user. Since CPU frequency transitions incur certain latency, and the proposed implementation uses unsupervised learning to group a certain range of utilization into one par-

Table 1: Benchmarks per User

<i>User1</i>	<i>User2</i>	<i>User3</i>	<i>User4</i>	<i>User5</i>
stringsearch	sha	bitcount	susan	bitcount
sha	crc32	qsort	patricia	fft
crc32	typeset	stringsearch	qsort	susan

Fig. 6: CPU Frequency per user

ticular frequency, it reduces the number of p-state (CPU frequency) transitions (as compared to the ondemand governor, which changes frequency every time the CPU utilization value changes even by 1 percent), thus, reducing the latency of CPU operations and hence, better performance.

Moreover, the existing DVFS technique does not take the MAR into account. From [1], we know that if MAR increases, to get higher performance per watt, we need to lower the CPU frequency (i.e. to hit the critical speed). Thus, in our proposed implementation, this is achieved intelligently through the two-input neural network, wherein we see that for the memory-intensive applications,

the CPU frequency is reduced as compared to the existing technology to get better results.

Finally, since the implementation is lightweight, the inference is faster, leading to the quicker selection of CPU frequency, thus, more boost in performance. Figures 7 to 9 show the comparison of the two techniques in terms of power and performance for each sub-test, here RAPL stands for Running Average Power Limit which is a commercial standard to provide estimated energy measurements for CPUs. Figure 10 shows the percentage impact of the proposed algorithm and implementation.

Fig. 7: Performance comparison

5 Conclusion

Replacing complicated mathematical models, this paper has proposed and implemented a very simplified intelligent frequency-selection algorithm using unsupervised clustering (Kohonen Self-Organizing Maps), which focuses on optimizing the CPU frequency for both CPU and memory-intensive applications. Moreover, due to the learning-based scheme being implemented on-chip, it has enabled faster and more lightweight neural inferencing. Thus, it incurs a much lower overhead cost than the existing DVFS scheme, which employs a relatively complex algorithm. The proposed approach focuses on finding the perfect CPU frequency such that, unlike related works in this field where either CPU or memory-bound (but, not both) applications are considered for DVFS, our implementation focuses on both simultaneously, thus, catering to the more realistic needs of the user. Moreover, unlike other works that focus only on saving power at the cost of performance, our implementation boosts or maintains

Fig. 8: RAPL domain SoC Power comparison

Fig. 9: Performance per Watt comparison

performance while saving power at the same time. The algorithm was implemented in the Linux operating system (Ubuntu) on an Intel i7-4720HQ Haswell Processor. In the aforementioned experiments, the proposed implementation was compared to the Ondemand DVFS governor. Results show that over the existing DVFS scheme in the system, our ANN-based frequency selection algorithm obtained up to 20% performance boost, up to 16% power savings, and up to 30% boost in Performance per Watt. Thus, based on user-behavior, the proposed technique can adjust the CPU frequency such that better performance is achieved, thus enhancing user experience, with the additional benefit of saving power, hence, increasing battery life.

Fig. 10: Percentage impact

Although this paper introduced a simple, yet effective frequency selection technique based only on CPU utilization and MAR, we have not touched any graphics aspect yet. Thus, a more advanced frequency selection can be proposed to include Graphics utilization, and thus, set the CPU, DDR, and Graphics frequencies intelligently to give the best performance on all kinds of workloads, be it CPU-intensive, memory-intensive or graphics-intensive. Further, voltage scaling can be incorporated as well. But, implementing such schemes shall increase the complexity, and hence, the overhead cost significantly. Thus, the ML algorithm should also be chosen wisely to reduce the cost as much as possible, thus, carrying out a proper trade-off between the benefits of the algorithm and power and performance overhead.

Acknowledgements We, the authors, would like to thank the Department of Science and Technology, Govt. of India for their continuous support towards research activities at BITS-PILANI institute. We would like to extend a special thanks to Prof. S. Gurunaryanan for helping us simplifying our inchoate ideas to a more comprehensive format.

References

1. W. Liang, S. Chen, Y. Chang and J. Fang, "Memory-Aware Dynamic Voltage and Frequency Prediction for Portable Devices," 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Kaohsiung, 2008, pp. 229-236, doi: 10.1109/RTCSA.2008.19.
2. K.K. Rangan, G.-Y. Wei, D. Brooks, Thread motion: "Fine-grained power management for multi-core systems," ACM SIGARCH Comput. Arch. News 2009, 37, 302-313.
3. A. Lahiri, N. Bussa, P. Saraswat, "A Neural Network Approach to Dynamic Frequency Scaling," in Proceedings of the 2007 International Conference on Advanced Computing and Communications, Guwahati, India, 18-21 December 2007; pp.738-743.
4. M. Moeng, R. Melhem, "Applying statistical machine learning to multicore voltage & frequency scaling," in Proceedings of the 7th ACM international conference on Computing frontiers, Bertinoro, Italy, 17-19 May 2010; pp. 277-286.

5. Q. Zhang, M. Lin, L.T. Lin, L.T. Yang, Z. Chen, P. Li, "Energy-Efficient Scheduling for Real-Time Systems Based on Deep Q-Learning Model," *IEEE Trans. Sustain. Comput.* 2017.
6. H. Jung, M. Pedram, "Supervised Learning Based Power Management for Multicore Processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2010, 29, 1395–1408.
7. G. Tesauro, R. Das, H. Chan, J. Kephart, D. Levine, F. Rawson, C. Lefurgy, "Managing power consumption and performance of computing systems using reinforcement learning," in *Proceedings of the NIPS 2007*, Vancouver, BC, Canada, 3–6 December 2007.
8. L. Yang, R. P. Dick, G. Memik and P. Dinda, "HAPPE: Human and Application-Driven Frequency Scaling for Processor Power Efficiency," in *IEEE Transactions on Mobile Computing*, vol. 12, no. 8, pp. 1546-1557, Aug. 2013.
9. T. Kohonen, "The self-organizing map," in *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480, Sept. 1990.
10. D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart and R. Geyer, "An Energy Efficiency Feature Survey of the Intel Haswell Processor," 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, Hyderabad, 2015, pp. 896-904.
11. M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4* (Cat. No.01EX538), Austin, TX, USA, 2001, pp. 3-14, doi: 10.1109/WWC.2001.990739.
12. Christian Poellabauer, Leo Singleton and Karsten Schwan, "Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications" *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05)*, 2005.
13. C. M. Kamga, "CPU Frequency Emulation Based on DVFS," 2012 IEEE Fifth International Conference on Utility and Cloud Computing, Chicago, IL, 2012, pp. 367-374, doi: 10.1109/UCC.2012.34.
14. K. B. Chun, C. Lee and W. W. Ro, "A frequency scaling model for energy efficient DVFS designs based on circuit delay optimization," 2015 International Symposium on Consumer Electronics (ISCE), Madrid, 2015, pp. 1-2, doi: 10.1109/ISCE.2015.7177841.
15. T. Kohonen, "Essentials of the self-organizing map," in *Neural Networks*, Volume 37, 2013, Pages 52-65, ISSN 0893-6080, <https://doi.org/10.1016/j.neunet.2012.09.018>.
16. D. Rajan, R. Zuck and C. Poellabauer, "Workload-Aware Dual-Speed Dynamic Voltage Scaling," 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06), Sydney, Qld., 2006, pp. 251-256, doi: 10.1109/RTCSA.2006.64.
17. A. Shye et al., "Learning and Leveraging the Relationship between Architecture-Level Measurements and Individual User Satisfaction," 2008 International Symposium on Computer Architecture, Beijing, 2008, pp. 427-438, doi: 10.1109/ISCA.2008.29.
18. A. K. Bera, P. Sharma and A. Gupta, "On-Chip Intelligent Frequency Scaling using Artificial Neural Networks" , 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 2020 (under review)