Improving EDP in Multi-Core Embedded Systems through Multidimensional Frequency Scaling

Wagner dos Santos Marques*, Paulo Silas Severo de Souza*, Arthur Francisco Lorenzon†,
Antonio Carlos Schneider Beck†, Mateus Beck Rutzig‡, and Fábio Diniz Rossi*
*Federal Institute of Education, Science, and Technology Farroupilha - Alegrete - Brazil
†Institute of Informatics - Federal University of Rio Grande do Sul - Porto Alegre - Brazil
‡Center of Technology - Federal University of Santa Maria - Santa Maria - Brazil
Email: {wagner.marques, paulo.souza}@email.com
{aflorenzon, caco}@inf.ufrgs.br, fabio.rossi@iffarroupilha.edu.br, mateus@inf.ufsm.br

***Abstract*—Energy saving management in multi-core embedded environments has been a challenge for designers. To achieve energy efficiency, most studies consider dynamic frequency scaling on one hardware component only, such as processor or memory – which will most likely also affect performance. This work proposes the use of frequency scaling considering the three most important hardware components altogether: processors, L2 cache, and RAM; seeking for the best set of frequencies for each one of them to improve the Energy-Delay Product (EDP), depending on the application's behavior. Therefore, this work addresses multidimensional frequency scaling for multi-core embedded systems. By evaluating different frequency levels, we show that the EDP can be improved in up to 46.4% when compared to the standard way that the frequencies are configured.**

***Keywords*—EDP, Embedded Systems, Frequency Scaling.**

I. Introduction

The popularization of embedded devices, associated with the demand for more computational power and autonomy, increased the importance of energy and performance as quality metrics in the context of embedded systems projects [1]. Computational power is achieved mainly by the improvement of processing units and their replication, and fast memories. Energy saving is a result of the intelligent use of resources associated with techniques and mechanisms offered by the architecture, such as sleep states, throttling, green schedulers, and voltage and frequency scaling and so on [2].

Specifically concerning voltage and frequency scaling [3], several works have applied such technique on the processor or memory to study the trade-off between energy and performance[4][5][6]. For instance, the reduction in the processor frequency results in fewer instructions to be executed per second, which may increase the execution time. Besides, the decrease in the memory frequency increases the latency, which may also reduce the performance. If one considers that most of the embedded processors are multi-core and the execution of parallel applications increases the competition for shared resources (e.g., L2 cache and RAM), such trade-off might be even more pronounced.

Nevertheless, the processes are not uniformly distributed across the hardware components, such as processor and memory [7]. For instance, when CPU-bound processes are running, it is not necessary to keep the memory frequency at the maximum. On the other hand, when memory-bound processes are running, the processor frequency can be reduced. Therefore, if these operating frequencies can be adjusted independently, focusing on the behavior of the application at hand, energy savings can be achieved without sacrificing performance.

Considering this research area that involves a huge number of variables, a multidimensional frequency scaling could enable energy savings, while maintaining the performance requirements of applications. Therefore, the main contribution of this work is to identify the best settings and apply frequency scaling for the processor, L2 cache, and RAM based on each application behavior, relating this behavior to how CPU/Memory intensive the application is. By targeting the energy-delay product (EDP), we show the best point of balance between energy saving and performance.

For that, we use a set of nine well-known parallel applications from assorted benchmarks suites and domains. They are executed on an ARM Cortex-A7 processor that enables changing the operating frequency of the three hardware components handled in this work. We consider four different frequency levels based on the maximum allowed for each of them: 25%, 50%, 75%, and 100%; resulting in a total of 64 possible combinations per application. In the most significant case, by choosing the ideal frequency level for each component, 46.4% in EDP improvements are achieved, when compared to the standard way that the frequencies are configured on the target platform (i.e.: at the maximum allowed).

This paper is organized as follows: In Section II we show a theoretical reference about the technologies that support frequency and voltage changing in computing devices, as well as some related work; In Section III we present the methodology, scenario, evaluation tools, results, and discussions. We finish this work in Section IV presenting our conclusions and future work.

II. Background and Related Work

Due to current market needs, energy efficient processors became increasingly relevant. Dynamic Voltage and Frequency Scaling (DVFS) is a technique that saves power by changing the operating frequency and voltage within a given range at run-time. By consequence, the power consumption of the chip will reduce, considering that the most significant source of

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

power is dynamic, which is given by $P = C.f.V^2$, where C is the transistor capacitance, f is the operating frequency and V is the supplied voltage. The operating frequency determines the voltage required for a stable operation, and it can be reduced as the operating frequency decreases. It can produce a significant reduction in power consumption due to the relation V^2 shown above. Many works have applied such technique on processors in order to save energy, such as Mantovani et al. [8], Ribic and Liu [9], and Melot et al. [10].

Several studies have used the very same technique to address the memory only [11], [6] [4]. In [11], Chen et al. proposed to improve the trade-off between performance and power saving, reducing the frequency of the memory. However, it only considers CPU-intensive processes. Although the paper presents up to 23% energy savings, in a production environment may be overhead when there are IO-bound processes.

Similarly, the work of Puttaswamy et al. [6] assessed frequency scaling of the L2 cache. Although the work has achieved energy savings of up to 36%, performance was impacted by up to 29%. In [4], David et al. proposed DVFS to adapt the memory system's operating frequency dynamically. However, the study only considered IO-bound processes, in which DVFS will highly benefit from since most of the time the resources will be waiting for external data (I/O). In contrast, the proposed work considers different applications behavior, and does so by maintaining high frequencies for components that most affect the performance and reducing the frequency of the least necessary ones.

Deng et al. [5] developed CoScale, a system for coordinating both the CPU and memory system DVFS under performance constraints. The system explores a set of possible frequency settings in such a way it efficiently minimizes the full-system energy consumption within a performance bound. The work of Li et al. [12] is similar, and also focus on energy savings. Although the results are positive, both studies do not consider the L2 cache, and they were implemented in a simulated environment. Different from such studies, this paper considers the L2 cache and addresses multidimensional frequency scaling on real hardware using hardware counters, which are more accurate. Besides, Li et al. [12] have not considered specific benchmark for embedded platforms. Our work is focused on embedded platforms and uses specific benchmarks for such platform.

When compared to the previous work, our proposal orchestrates all the possible operating frequencies that can be tuned in an embedded platform, aiming to improve the trade-off between performance and energy saving, using specific applications for embedded systems, with different behaviors.

III. Evaluation and Discussion

Unlike some specific applications used to measure the performance of HPC systems, most applications do not use all available resources always at maximum capacity. Thus, there is a fluctuation in their load as they are executed, depending on their processing needs and access to I/O. This means that when an application is intensively using the CPU, the L2 or RAM memories may be underutilized. In the same way, when the application is Memory-Bound , it does not need 100% of the CPU.

TABLE I: Frequency scaling each component. In our tests, we evaluated intervals of the total for each frequency component.

Components	Frequency Usage			
	25%	50%	75%	100%
Processor	225MHz	450MHz	675MHz	900MHz
L2 Cache	62MHz	125MHz	187MHz	250MHz
RAM	112MHz	225MHz	337MHz	450MHz

As already discussed, we aim to improve EDP by finding the best-operating frequency level of each component. For that, nine parallel applications from assorted benchmark suites and domains were chosen: four applications (BasicMath, Dijkstra, Patricia, and Susan) from the Parmibench [13], an open-source benchmark for embedded multiprocessor systems; four applications (BT, IS, MG, and SP) from the NAS Parallel Benchmark [14], a well know suite used to measure high-performance systems; and one application from the literature: calculation of the PI number (PI). The applications were classified into three groups: CPU-bound, balanced use of the resources, and memory-bound. In order to classify these applications, Performance Application Programming Interface (PAPI [15]) was used to collect and analyze data related to the number of cycles, executed instructions, and the number of memory accesses (hits and misses in the L1, L2, and RAM).

The experiments were performed on a Raspberry PI[1] 2, a single-board computer that enables frequency scaling for its quad-core ARM Cortex-A7 processor, L2 cache, and RAM. Table I depicts the frequency values tested for each one of the three studied components. On this platform, the benchmarks were executed with the maximum number of hardware threads supported (which is the regular way parallel applications are executed [16]). The energy consumption was obtained through the use of an external multimeter, plugged between the platform and the power outlet. Thus, the energy consumption obtained considers the whole platform (processor, memories, bus, etc.). The applications were compiled with the GCC 4.9.2 using the optimization flag -O3 and executed over the Linux system (kernel 4.4.13). The results presented next are an average of ten executions for each configuration (CPU, L2 cache, and RAM frequencies) with a standard deviation lower than 0.5%. When executing such applications on multicore embedded systems, usually all the components are configured to the maximum frequency allowed (100%, 100%, 100%). Therefore, we consider this scenario as **Baseline** for the comparison.

Fig. 1 shows the results for the EDP associated with all tested frequency combinations (CPU, L2 cache, and RAM). They are classified in a way that the first ones (e.g.: (a) PI, (b) Dijkstra, (c) Patricia, (d) BasicMath etc) are the most CPU-intensive ones, in this order; while the last ones are the most memory-intensive ones (e.g. (i) SP). Table II depicts the classification of each benchmark regarding its behavior, the best configuration (i.e.: best frequency for each component), and the EDP improvements (EDP Gain) over the baseline.

Considering all the results presented in Figure 1, one can note that the worst EDP results are achieved when the CPU is

[1]https://www.raspberrypi.org

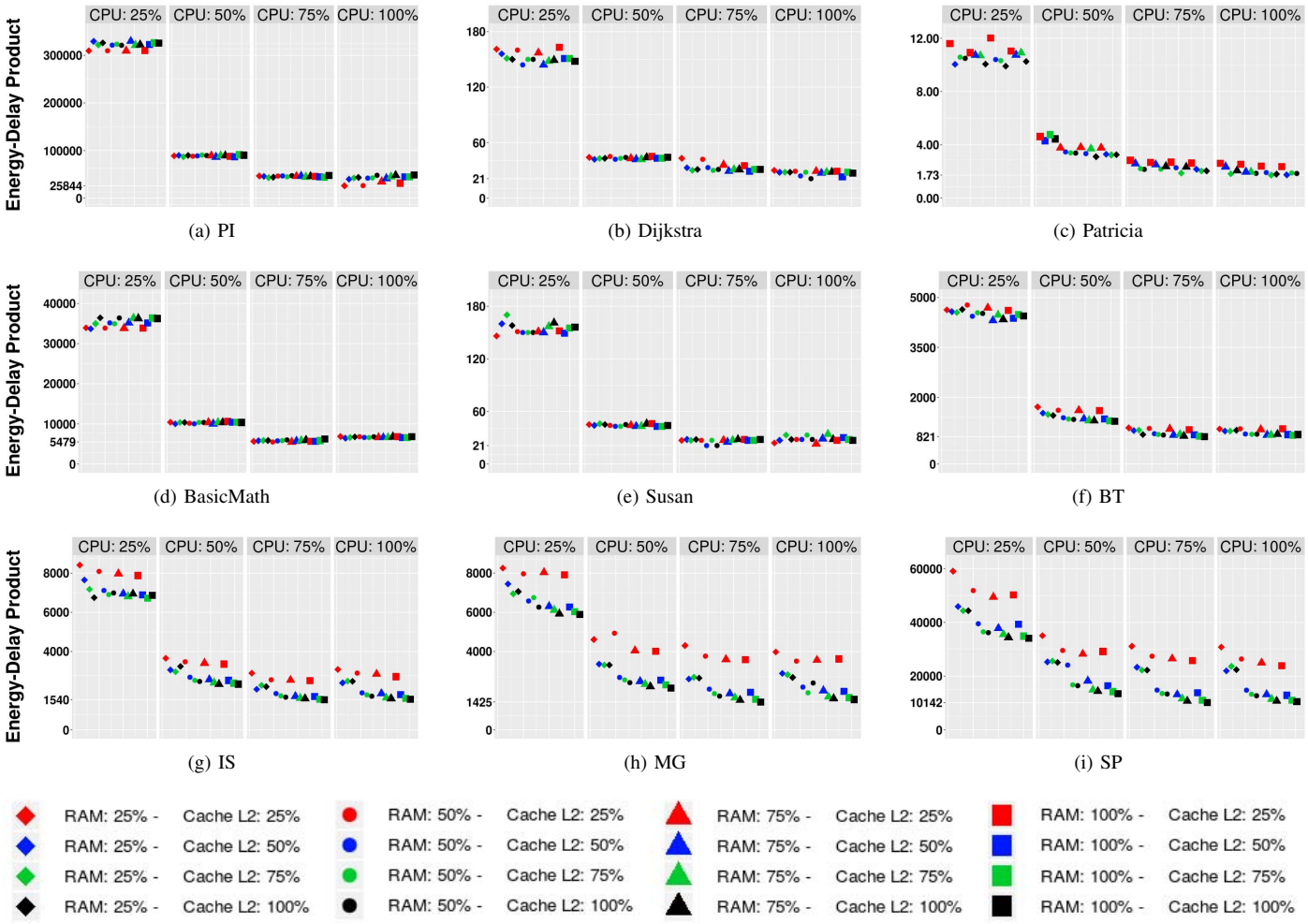

Fig. 1: Optimal specs obtained on the execution of the algorithms.

using only 25% of the maximum frequency allowed, regardless the frequency level of the memories. This scenario happens because the increase in the execution time is greater than the reduction in energy consumption caused by decreasing the processor frequency. On the other hand, the best results are achieved when the CPU is configured to use 75% or 100% of the maximum possible frequency, depending on the benchmark. In such cases, the memory starts to play an important role. For instance, in the applications IS, MG, and SP, keeping the frequency of the L2 cache at 25% negatively affects the EDP, while for others (CPU-B application), decreasing its operating frequency may bring good results (e.g.: PI).

Let us first discuss the CPU-Bound applications. The calculation of the PI number (PI) is part of a group of scientific applications that perform calculations and recursions almost exclusively using the CPU, with few accesses to data in memory [17]. As Table II shows, for such applications set the CPU frequency at the highest level while keeping the memory frequency at the minimum value possible improved the EDP in up to 46.4% when compared to the the baseline.

Dijkstra searches shortest paths between graph nodes. It is also a CPU-intensive application, but with more memory accesses than PI. In this application, each graph node will be accessed several times during execution and there will be data exchange between the threads through L2 cache (first level of memory that is shared between the cores). Therefore, keeping the L2 cache operating at the maximum frequency while reducing the RAM frequency to 50% improved the EDP by 21.4% when compared to the baseline. Similarly, Patricia

TABLE II: The best set of frequencies among CPU, L2, and RAM, aiming at the best EDP

Behavior	Application	Best conf. (CPU, L2, RAM)	EDP Gain
CPU-B	PI	(100%, 25%, 25%)	46.4%
	Dijkstra	(100%, 100%, 50%)	21.4%
	Patricia	(100%, 75%, 75%)	6.2%
Balanced resources usage	BasicMath	(75%, 25%, 50%)	19.0%
	Susan	(75%, 50%, 50%)	22.6%
MEM-B	BT	(75%, 100%, 100%)	7.1%
	IS	(75%, 100%, 100%)	1.8%
	MG	(75%, 100%, 100%)	7.9%
	SP	(75%, 100%, 100%)	3.5%
Geometric Mean			9.94%

also stores data in a tree structure and performs searches on the leaf nodes. However, different from Dijkstra, it needs more accesses to the main memory, requiring a higher frequency for such component.

BasicMath performs simple mathematical calculations, such as integer square root, while Susan performs pattern recognition on images [13]. Both benchmarks present a balanced load of their resources, and this behavior is reflected in the set of best EDP configurations. A multidimensional frequency scaling can take advantage of this type of application because it does not need to keep the CPU at 100% during the whole program execution, due to the amount of non-uniform memory accesses. In the same way, but for the exact opposite reason, neither the memories need to run at their maximum frequency. In both applications, by setting the memory frequency to 50% improved the EDP in approximately 20% when compared to the baseline.

Finally, let us consider the applications that make intensive use of memory. BT and SP solve differential equations, IS sorts data in parallel, and MG tests multigrid communication [14]. Since such applications spend most of their time performing memory operations, it makes sense for them to remain at their maximum frequencies. Note, however, that the most data is kept in the cache memory and, during these periods, CPU will have to handle a significant load. For this reason, the CPU frequency could be reduced to save energy, but not quite to the point of compromising performance.

Considering the results showed in Table II, one can note that the memory system is the component which influences EDP the most: in applications where the RAM frequency is reduced to 25% or 50%, huge EDP improvements over the baseline are achieved. On the other hand, when only the processor frequency is adjusted, the EDP improvements over the baseline are up to 7.9%. When all the three components are adjusted at the same time, it is possible to achieve considerable EDP improvements (19% and 22.6%) when compared to the baseline. Therefore, these results reinforce the importance of a multidimensional frequency exploration. By using that, an improvement in EDP of up to 46.4% and an average of 9.94% among were achieved in all applications.

IV. Conclusions and Future Work

Proposals to balance energy savings with the smallest impact on performance have been the focus of several studies. However, the vast majority of work consider only one dimension (e.g.: CPU frequency scaling) when it comes to frequency scaling, not fully exploiting the capabilities that architecture offers.

This paper has shown that multidimensional frequency scaling of CPU, L2, and RAM could improve EDP in embedded multi-core systems by up to 46.4%; and that it is possible to reduce energy without impacting performance. As future work, we intend to increase the set of tests using more operating frequency intervals.

References

[1] J. T. Russell and M. F. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," in *Proceedings of the International Conference on Computer Design*, ser. ICCD '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 328–333.

[2] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 1, pp. 7:1–7:34, Jan. 2016.

[3] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "Cpu db: Recording microprocessor history," *Commun. ACM*, vol. 55, no. 4, pp. 55–63, Apr. 2012.

[4] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*. New York, NY, USA: ACM, 2011, pp. 31–40.

[5] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 143–154.

[6] K. Puttaswamy, K.-W. Choi, J. C. Park, V. J. Mooney, III, A. Chatterjee, and P. Ellervee, "System level power-performance trade-offs in embedded systems using voltage and frequency scaling of off-chip buses and memory," in *Proceedings of the 15th International Symposium on System Synthesis*, ser. ISSS '02. New York, NY, USA: ACM, 2002, pp. 225–230.

[7] A. S. Tanenbaum, *Operating Systems: Design and Implementation*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[8] P. Mantovani, E. G. Cota, K. Tien, C. Pilato, G. Di Guglielmo, K. Shepard, and L. P. Carloni, "An fpga-based infrastructure for fine-grained dvfs analysis in high-performance embedded systems," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 157:1–157:6.

[9] H. Ribic and Y. D. Liu, "Aequitas: Coordinated energy management across parallel applications," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016, pp. 4:1–4:12.

[10] N. Melot, C. Kessler, and J. Keller, "Energy-optimized static scheduling for many-cores with task parallelization, dvfs and core consolidation," in *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '16. New York, NY, USA: ACM, 2016, pp. 126–126.

[11] M. Chen, X. Wang, and X. Li, "Coordinating processor and main memory for efficientserver power control," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 130–140.

[12] X. Li, R. Gupta, S. V. Adve, and Y. Zhou, "Cross-component energy management: Joint adaptation of processor and memory," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 3, Sep. 2007.

[13] S. M. Z. Iqbal, Y. Liang, and H. Grahn, "Parmibench - an open-source benchmark for embedded multiprocessor systems," *IEEE Computer Architecture Letters*, vol. 9, no. 2, pp. 45–48, Feb 2010.

[14] 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. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks;summary and preliminary results," in *Proceedings of the ACM/IEEE Conference on Supercomputing*. New York, NY, USA: ACM, 1991, pp. 158–165.

[15] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *Int. J. High Perform. Comput. Appl.*, vol. 14, no. 3, pp. 189–204, Aug. 2000.

[16] J. Lee, H. Wu, M. Ravichandran, and N. Clark, "Thread tailor: Dynamically weaving threads together for efficient, adaptive parallel applications," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 270–279, Jun. 2010.

[17] G. E. Andrews, R. Askey, and R. Roy, *Special functions*. Cambridge: Cambridge University Press, 1999.