Performance-Aware Energy-Efficient Processes Grouping for Embedded Platforms

Paulo Silas Severo de Souza†, Wagner dos Santos Marques†, Marcelo da Silva Conterato†,
Tiago Coelho Ferreto†, and Fábio Diniz Rossi*
*Federal Institute of Education, Science, and Technology Farroupilha (IFFar) - Alegrete - Brazil
†Pontifical Catholic University of Rio Grande do Sul (PUCRS) - Porto Alegre - Brazil
Email: {paulo.silas, wagner.marques.001, marcelo.conterato}@acad.pucrs.br
tiago.ferreto@pucrs.br, fabio.rossi@iffarroupilha.edu.br

***Abstract*—Embedded systems are becoming more popular in several sectors of society by performing a broad range of tasks. In this context, there is a concern about improving the trade-off between performance and energy savings since they are usually battery-dependent. However, it is not a trivial task since some embedded devices are developed with strict hardware constraints. In this sense, we present an operating system level tool that groups processes dynamically on resources. Our tool manages different types of processes, and through isolation characteristics, provides a better utilization of resources. The results show that our tool can improve the trade-off between performance and power saving of embedded systems in up to 15%.**

***Keywords*—Energy-Delay Product, Embedded Systems, Workload Management.**

I. Introduction

Embedded systems, which usually are computers designed for specific purposes with hardware constraints, are becoming more popular than ever by performing several tasks in the most diverse contexts of society such as education, telecommunications, security, and health [1]. One of the most significant challenges in such devices is how to improve the trade-off between application performance and power savings. Several works propose especially the use of dynamic voltage and frequency scaling on processors [2]. To increase the applications performance, the processor's frequency is maximized [3]. However, this approach presents some architectural limitations since keeping the processor always operating at high frequencies generates heat and also consumes unnecessary amounts of energy. On the contrary, reducing the processor frequency decreases the power consumption, but also leads to performance issues, since fewer instructions will be processed per second. Besides, processes may have different resources' demands depending on its behavior, so changing the processor frequency based on the applications' behavior result in energy savings without sacrificing the performance of the device [4]. However, this mechanism is usually linked to a particular application behavior, which leads most to static solutions.

In this context, **this paper presents a dynamic resource allocation tool that uses multiple Linux kernel features in order to make the following main contributions**: (i) The proposed tool uses process-grouping mechanisms to separate applications that involve the business logic processes (e.g. applications that are directly linked with the main purpose of the embedded system in the context it is inserted, for example, a data pre-processing algorithm) from the general purpose ones (e.g., system tasks), organizing them into different processor cores to improve the resources usage and avoid performance degrading events such as cache misses; (ii) It dynamically enables or disables processor cores according to the system needs in order to reduce the dynamic power consumption; (iii) It increases the performance of embedded systems during the execution of real-time and high demanding applications by dedicating some processor cores to these tasks, so they do not need to compete for resources with other processes. The remaining of this article is organized as follows: in Section II, we present theoretical concepts that pave the way to our proposal, which is described in Section III. Then, in Section IV we present and discuss the results of the experiments that were conducted to evaluate the efficiency of the proposal, and based on such findings, we compare our tool with related work in Section V. Finally, Section VI is reserved to the final remarks.

II. Motivation

Techniques that operate at the operating-system level emerge as viable solutions to improve the trade-off between performance and power saving even in embedded systems with architectural limitations that can hamper the use of some hardware-level features. One of the possibilities is manipulating the affinity of the processor with the running applications, which can be achieved through scheduling algorithms that can ensure that some processes or threads will only be handled by specific processor cores. For example, in a scenario when an embedded system with four processor cores is running a single high-performance task, the system processes could be grouped into specific cores to improve the CPU affinity and increase the overall system performance. Moreover, the remaining cores could be dedicated only to processing the high-performance application, avoiding unnecessary competition for resources with the system jobs. However, applications with several threads that work independently (e.g., clustering-based analysis algorithms, where each thread can be responsible for the analysis in a cluster) are not much benefited by running on a particular processor since other threads would have already using the Cache memory of such processor.

In addition to CPU affinity, load balancing also emerges as an alternative to ensure the efficient use of the resources. Load balancing is a technique that distributes the workload across the available computing resources in order to minimize the response time and avoid unnecessary overload of a single computing resource. Load balancing can be implemented in the context of a single device or a cluster: it can be used to balance the workload between the multiple processor cores of a single device, or it can be implemented to ensure the

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

cooperation of many embedded systems distributed across an environment during the execution of a single task [5]. Such technique can also be useful when the running processes do not require the use of all the processor cores of the device or in cases when is not possible to use parallelism due to high data dependency. In such scenarios, the processor cores that are not being used could be temporarily disabled in order to decrease the dynamic power consumption of the device.

Load balancing mechanisms also allow the use of techniques such as Cache prefetching. Such technique relies on the memories hierarchy because it fetches data or instructions from their storage (usually main memory) to faster memories (e.g., Cache memories) before the application needs it. The prefetching process is based on the applications' behavior, and it can be accomplished through software mechanisms that send special instructions to the processor warning that some data will be required at some point soon. In this sense, when the application asks for that data, it loads faster since it already is in the Cache memories [6]. Cache prefetching increases the performance of applications by avoiding performance degrading events such as Cache misses, which refers to attempts to find some data which already was replaced from the Cache memory. In such cases, the processor needs to read the missed data from the main memory [7]. As this kind of Cache prefetching is controlled at the software-level, it can be used by computers with architectural constraints such as some embedded systems. Nonetheless, usually operating systems do not provide load balancing algorithms to ensure the Cache affinity for the running applications, so developers need to implement their own algorithms to this end. Moreover, implementing such kind of technique in the context of embedded systems is not a trivial task since if not controlled load balancing techniques can generate unnecessary power consumption by distributing tasks that could be executed on a single processor to multiple processors [8]. In this context, load unbalancing strategies emerge as an option to be combined with load balancing policies to provide a most efficient use of the resources.

III. Resource Allocation Tool

The increasing demand for computational power management in the context of embedded systems is promoting the use of techniques such as virtualization and frequency scaling to find optimal points between performance and power saving. In this context, techniques such as Cache prefetching [9] that can be manipulated at the operating-system level emerge as a viable solution. Nevertheless, there are few studies with the focus on using such type of technique in the context of embedded systems. In this sense, we introduce a process grouping tool that performs workload balancing in embedded systems to improve the trade-off between performance and power saving.

Usually, applications that involve the business logic of the environment in which the embedded device is included have to compete for resources with the general purpose tasks (e.g., system processes or other running applications). This competition can be harmful to both types of tasks since it can increase the number of Cache misses, which is a state where some data or instruction requested by a process is not found in the Cache memory. In such cases, a performance overhead is produced since the system needs to fetch that data from other Cache levels or even from the main memory. In this sense, we present a tool that takes advantage of Linux kernel features to balance the workload between the available computing resources in the embedded device in order to improve the resources usage efficiency and avoid problems such as Cache misses.

In Linux operating systems, the processes are organized into a tree in which all processes are treated as children of the init process, which is initialized by the kernel during the system's initialization. Such hierarchical scheme makes it possible to manipulate multiple processes simultaneously. One of the options to achieve multiple processes manipulation is through Control Groups (also known as cgroups, a feature present in the Linux kernel since version 2.6.24) that joins processes into groups. In addition to monitoring a particular set of processes, cgroups provides a way to limit the use of resources (CPU, memory and I/O) to each created group. Moreover, the features offered by the Control Groups are distributed into subsystems or controllers, making it possible to choose which controllers will be initialized with the system to improve the system's performance.

This line of reasoning is followed by our proposal. As shown in Figure The tool collects information from the running processes and applies such knowledge along with operating system features to make decisions based on the running applications' behavior. The focus is on raising the resources usage efficiency and by consequence improving the trade-off between performance and power saving. In this context, our proposal analyzes the demand for resources of the running applications and organizes them into the processor cores to improve the cache prefetching accuracy and consequently decrease the response time of the device. Figure 1 presents three scenarios in which a quad-core embedded system is running four HPC applications to exemplify the use of the proposed tool. The first scenario (Figure 1a), shows how operating systems usually allocate applications among the processor cores (e.g., each application is scheduled to a distinct processor core). Figure 1b exemplifies a situation when our proposal analyzes the resource demand of the running applications and organizes them into only two processor cores, deactivating the other two unused cores to achieve power saving. Figure 1c represents a situation when, after organizing the processes into a scheme identical to the previous scenario, the tool realizes that the "App 1" is requiring more processing power. In such scenario, the tool dedicates an entire processor core to this application in order to avoid performance loss.

The proposed tool uses one of the Control Groups subsystems which is called *cpuset* that is responsible for manipulating the processor units and memory nodes in a cgroup. More specifically, the proposed tool dynamically manipulates two cpuset properties: "cpuset.cpus" and "cpuset.cpu_exclusive". The first property is used to balance the workload among the processor units of the embedded device. The second is employed to isolate certain processor units to specific groups. Our proposal divides the processes into two groups. Group 1 gathers general purpose and system processes. Group 2 gathers processes related to the environment business logic in which the embedded device is included. In this way, our proposed tool avoids unnecessary competition for resources between the system processes and the business logic applications. Moreover, as we can see in Figure 2, such approach also allows disabling processor cores when they are not necessary, in order to decrease power consumption.

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

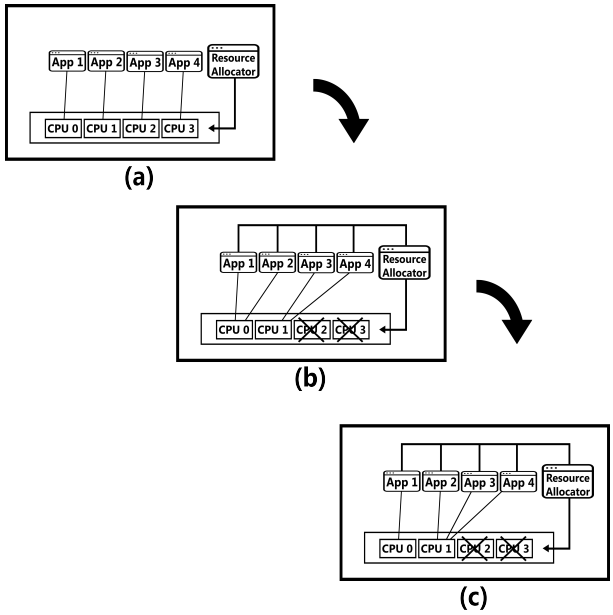

Fig. 1: Workload management scheme provided by the proposed tool exemplified in three scenarios in a quad-core embedded system running four HPC applications.

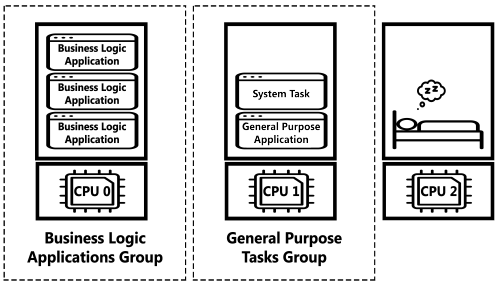

Fig. 2: Workload management provided by the proposed tool.

A. Workload Management with the Proposed Tool

The proposed tool works as follows. First, it automatically adds the system processes into a group (Group 1) and attributes a processor core to these tasks. Then, a second group (Group 2) designated for the business logic applications is created. After this phase, users need to inform the proposed tool which are the business logic applications (if not specified as business logic applications, all processes started after the groups' creation are automatically considered general purpose tasks). After the configuration phase, which involves the processes organization, the proposed tool starts a function that calculates the number of processor cores that are required by the working applications. The algorithm gets the overall CPU usage and analyzes how many processor cores are needed to ensure the performance of the running applications even if that usage rises in up to 5% (such extra percentage was added since the applications resource usage can fluctuate). After analyzing the optimal number of processor cores to run the working tasks, the proposed tool sets this limit to the processes through the cpuset properties presented above. The proposed tool calculates the standard deviation of the resources usage history of the running applications to set a time interval between the executions of the cores limits management procedure. Both the decision on the number of cores and the interval between runs always occurs when a new application is started. In this sense, the proposed tool will readjust the processor cores limits more frequently as the resources usage oscillation increases. The operations performed by the proposed tool are graphically presented in the diagram presented in Figure 3.

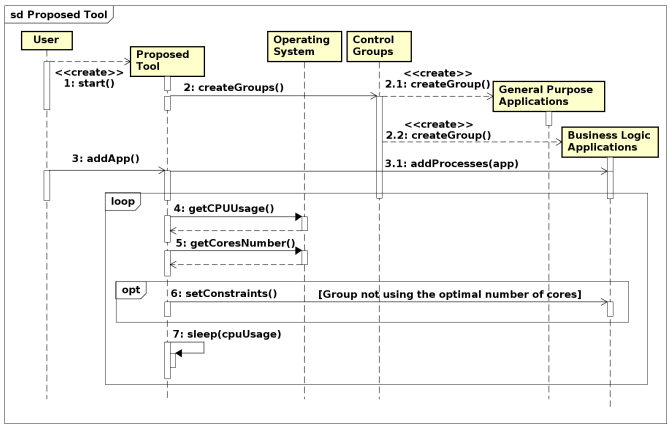

Fig. 3: Sequence diagram with the activities' flow generated by the proposed tool.

IV. Evaluation and Discussion

Experiments were conducted to evaluate the impact caused by the tool in embedded systems during the execution of several algorithms with different behaviors. We analyzed performance, power consumption, and Energy-Delay Product (which is a metric that takes into account both performance and power consumption). The energy consumption was obtained through the use of an ACS712 hall-effect current sensor. We conducted the experiments using nine algorithms from the NAS Parallel Benchmarks [10] (as business logic applications) version 3.3-OMP, a popular suite that contains high-performance computing (HPC) algorithms derived from real-life applications. We ran two instances of each algorithm in parallel to simulate scenarios when embedded systems need to execute more than one task at the same time. The experiments were performed on a Raspberry PI 2 model B, a single-board computer with a 900MHz quad-core ARM Cortex-7 processor and 1GB of RAM running the operating system Ubuntu ARM 17.04. The algorithms were compiled with GCC and GFortran both in version 6.3.0. The results presented next are an average of ten executions of each algorithm with a standard deviation less than 1%.

A. Performance

Figure 4 shows the performance results during the execution of the chosen algorithms. As we can see, in all cases our tool was capable of improving the performance in up to 7.54%. Such results show that dynamic processor grouping can increase the performance by improving the CPU affinity and avoiding unnecessary competition for resources with the system tasks even in cases when more than one application is being executed at the same time.

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

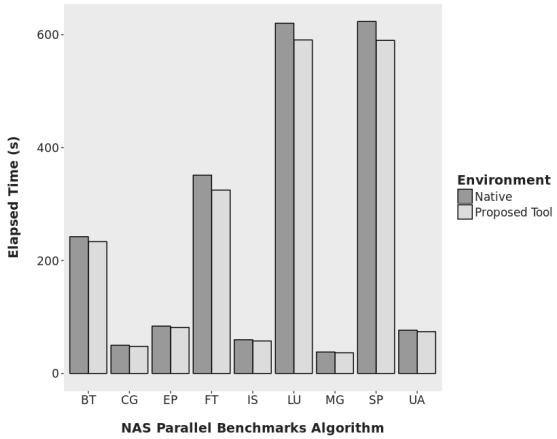

Fig. 4: Performance comparison between the proposed tool and the approach adopted by the default by the operating system.

The results show that applications that rely on shared-memory access can take advantage of processes grouping to improve the chances that the data used during its execution will remain in the Cache memory. For example, the highest performance gain was achieved by FT (7.52%), which uses Fast-Fourier Transform to solve three-dimensional partial differential equations. During its execution, several discrete multi-dimensional FFTs need to access a vast array multiple times. SP (Scalar Penta-diagonal solver) and LU (Lower-Upper Gauss-Seidel solver) achieved the second and third best performance results that presented gains of 5.36% and 4.78% respectively. These tests are applications containing structured-mesh codes that execute a set of operations on multidimensional arrays. As SP and LU require regular memory access by using structured-mesh codes, they have taken advantage of the processor cores isolation provided by the proposed tool. It increases the CPU affinity and enables the use of cache prefetching to boost performance by fetching data or instructions to faster memories before they are needed, based on mechanisms that recognize the next elements that will be used by the algorithm.

The Multi-Grid (MG) algorithm, which is an iterative problem that involves both short and long-distance communication of structured data, and therefore is also highly dependent on shared-memory access speed also showed positive results, presenting a performance increase of 3.86%. Moreover, the proposed tool also produced a performance gain of 3.76% in the Integer Sort kernel (IS), which also highly depends on the memory-access speed since it executes integer sorting operations that used particle method simulations. This result shows that not only applications that rely on floating-point arithmetic can take advantage of the features provided by the proposed tool. The algorithm which was less benefited by the use of the proposed tool was the Embarrassingly Parallel (EP) with a gain of 2.68%. This kernel is responsible for estimating the performance during floating-point operations. What makes EP different from the other executed tests is the fact that it performs very few memory accesses. In this sense, improving the CPU affinity by processes grouping does not make a huge difference in its performance. Nonetheless, the proposed tool not only ensures the CPU affinity for the running applications but also isolates them into specific processor cores to avoid unnecessary competition for resources with the system tasks. Hence, the EP kernel also presented a performance gain even not being highly dependent on the memory access speed as the other executed algorithms.

B. Power Consumption

The results regarding power consumption are presented in Figure 5. The best case was a gain of 4.04% achieved during the execution of LU. Such algorithm presents a very similar behavior to BT and SP, which also presented power savings of 0.38% and 0.22% respectively. The positive results achieved during the execution of these algorithms can be explained by its symmetric behavior which increases the potential of CPU affinity since they can continually reuse the data stored in the cache memory.

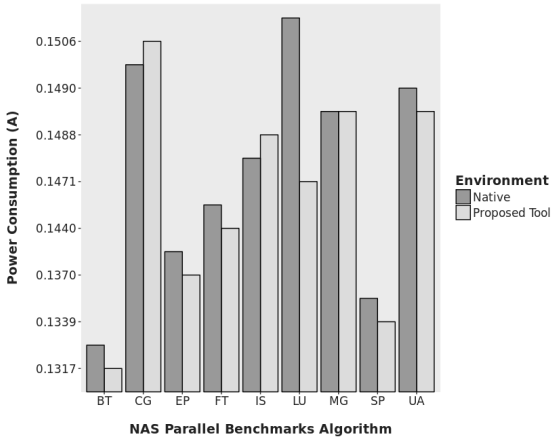

Fig. 5: Power consumption comparison between the proposed tool and the approach adopted by the default by the operating system.

The Fast Fourier Transform (FT) algorithm presents benefits from the use of the proposed tool, showing power saving of 1.23%. The positive result achieved during the execution of this algorithm can be explained by its behavior in which a partial differential equation is transformed by multidimensional FFTs that access a vast array multiple times. As the proposed tool dedicates processor cores to the applications, parts or even the whole single array that must be accessed by the FT algorithm can be kept in the cache memory. In addition to increasing the performance, such approach can also reduce the dynamic power of the device since it reduces the data transfer among the memories (the data required by the application do not need to be accessed from the main memory and then consequently moved to the cache memory several times).

The worst power consumption results were achieved in the Integer Sort (IS) and Conjugate Gradient (CG) kernels, in which using the proposed tool generated overheads of 0.13% and 0.20% respectively. As IS performs integer sorting operation, the CG kernel computes the smallest eigenvalue of a large sparse matrix. Despite the different purposes of these algorithms, they present some similarities: the instructions executed by both require very irregular data access. In this sense, as opposed to the algorithms that showed the higher power savings, the irregular data access of IS and CG may require access to data that is not in the cache memory despite the high CPU affinity provided by the proposed tool.

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

C. Energy-Delay Product

EDP (Energy Delay Product) metric [11] considers the latency together with the energy consumption through EDP = Energy (Joules) x Delay (Seconds). Such metric was used, and it can clarify, summarize, and corroborate our findings correlating performance with energy consumption. The results are presented in Table I.

TABLE I: EPD gain of proposed tool versus power-agnostic environment EDP.

NPB Algorithm	EDP Gain
3D Fast Fourier Transform (FT)	15.49%
Lower-Upper Gauss-Seidel Solver (LU)	12.97%
Scalar Pentadiagonal Solver (SP)	10.62%
Multi-Grid (MG)	7.55%
Block Tri-diagonal solver (BT)	7.51%
Integer Sort (IS)	7.27%
Conjugate Gradient (CG)	7.12%
Unstructured Adaptive (UA)	6.65%
Embarrassingly Parallel (EP)	5.61%

The results showed that all executed algorithms take advantage of the isolation mechanism of the proposed tool that avoids unnecessary competition for resources with the system tasks. Looking more specifically at the results, we notice that our proposal can highly benefit applications that rely on shared-memory access speed. That is the reason why the Embarrassingly Parallel (EP) kernel, which performs very few memory accesses, presented the smallest EDP gain (only 5.61%): it just took advantage of not competing for resources with the system processes at most. The best results were achieved in the execution of the FT and LU algorithms that presented gains of 15.49% and 12.97% respectively. Such results are the product of a combination of two characteristics of these algorithms. Firstly, they highly depend on the shared-memory access speed. Secondly, they perform regular memory accesses. During the execution of applications with irregular memory access, the number of useless prefetches tends to increase since its behavior cannot be easily predicted. In this sense, applications with irregular memory access cannot explore the maximum potential of cache prefetching techniques. It explains the results presented by IS, CG, and UA algorithms. They were dynamically changing memory accesses and presented small EDP gains, only showing better results when compared to EP which does not rely on shared-memory access.

The executed algorithms are derived from Computational Fluid Dynamics (CFD) applications. Nowadays, there are several scenarios in which embedded systems are used to run such kind of application. For example, embedded systems are used to examine corrosion levels in oil pipelines and make decisions through analysis performed by CFD applications. In the automotive sector, embedded devices can be used to perform real-time simulations involving aerodynamics and thermodynamics in airplanes to avoid accidents and help in decision-making. In such scenarios, performance is an important metric since the embedded devices must be able to execute CFD applications instantly to provide reliable feedbacks. On the other hand, power saving is also important as in some cases they need to operate for extended periods of time without recharging their batteries. In these contexts, our tool emerges with the proposal of improving the trade-off between performance and power saving (measured by the EDP metric) of embedded systems in up to 15.49% during the execution of several types of CFD applications.

V. Related Work

Some prior investigations focused on using load balancing and load unbalancing techniques in order to increase the resources usage efficiency during the execution of applications with different behaviors and, consequently, with different demands for resources. Some of these studies are presented below.

Liu and Gao [12] presented a dynamic load balancing algorithm that focuses on managing the workload between the computing resources of multicore embedded systems organized in distributed environments. Firstly, their algorithm analyzes the load state of each node of the cluster by getting information such as average runtime of all running processes and overall memory usage. After calculating the load state of a node, the algorithm proposed by them select the load balancing destination by implementing a sender-initiated policy. If the algorithm finds a lightly loaded node that can handle the data to be migrated then the load migration scheme is started.

Jahnich and Rettberg [13] presented a load balancing tool with the focus on distributed embedded automotive systems. Their proposal acts a middleware architecture that allows the migration of tasks between the devices connected to the vehicle system. For example, multiple mobile devices attached to a vehicle system can share resources with other to perform complex tasks such as data processing to provide faster and more reliable feedbacks.

Jeon, Lee, and Chung [8] presented a workload management mechanism that uses both load balancing and unbalancing techniques to increase the device's resources usage efficiency during the execution of both periodic and aperiodic tasks. To do that, the tool proposed by them focuses on managing the workload between the computing resources by classifying applications by its behavior. For example, their scheduling algorithm uses a concentration strategy during the execution of periodic tasks to reduce the power consumption by deactivating unused processor cores. On the other hand, their workload management mechanism employs a load balancing technique to allow that aperiodic tasks take advantage of the device's multiple processors since such kind of task usually focuses on achieving smaller response times as contrary to periodic tasks (in which the primary performance metric is a stipulated deadline, so a long waiting time is not a significant problem as long as the deadline is met).

Lim, Min, and Eom [14] proposed a scheduler algorithm called operation zone based load-balancer which migrate tasks aiming to achieve low latency in multicore embedded systems by maintaining overall CPU utilization balanced among the processor units. The goal of such load balancer is to perform load balancing in cases when the utilization of each processor of the device is not balanced. Moreover, the operation zone based load-balancer have an operation zone mechanism that analyzes the load-balancing frequency of each processor to avoid unnecessary task migrations and reduce the device's power consumption.

Bellasi, Massari, and Fornaciari [15] proposed the BarbequeRTRM, a runtime resource management framework with

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

support for multiple platforms (including embedded systems). The BarbequeRTRM framework employs an event-based scheduling algorithm where events can be variations on the amount of available resources or changes on the applications' resources usage so when an event happens the scheduling algorithm is triggered to identify an optimal resource partitioning for the running applications based on policies established by the framework.

Cho et al. [16] proposed a deadline-aware scheduling algorithm with the focus on decreasing the power consumption and the deadline misses of real-time multicore systems. The authors created a DVFS-based algorithm called ED^3VFS that determines the optimal instant to scale the device's operating frequency based on the tasks deadlines. Moreover, the tool proposed by them uses a deadline-aware load dispatch algorithm that employs multiple load-balance strategies to deal with real-time and normal applications simultaneously.

As techniques such as load balancing can be manipulated at the operating-system level, several investigations have been undertaken to take advantage of such features to improve the trade-off between performance and power saving. However, there are still underused operating-system level functionalities that could be employed to boost the resources usage efficiency and consequently improve such trade-off. For the best of our knowledge, our proposed tool is the first to focus on using process grouping, load balancing and load unbalancing strategies to manage the distribution of the running processes among processor cores to improve the system's performance during the execution of HPC applications and reduce the dynamic power consumption by deactivating unused processor cores. To better the understanding of the differences between our proposal and previous work, a checklist with characteristics is presented in Table II.

TABLE II: Comparison among the previous work and the proposed tool that takes into consideration five characteristics: i) *LB:* it takes advantage of load balancing techniques; ii) *SD:* a single embedded device can employ it; iii) *HI:* it is a hardware-independent solution; iv) *HPC-OP:* it is optimized for HPC applications; and v) *UCD:* it allows unused processor units deactivation.

Proposals/Features	**LB**	**SD**	**HI**	**HPC-OP**	**UCD**
Liu and Gao [12]	Yes	No	Yes	No	No
Jahnich and Rettberg [13]	Yes	No	Yes	No	No
Jeon, Lee, and Chung [8]	Yes	Yes	Yes	No	Yes
Lim, Min, and Eom [14]	Yes	Yes	Yes	No	No
Bellasi, Massari, and Fornaciari [15]	Yes	Yes	No	No	No
Cho et al. [16]	Yes	Yes	No	No	No
Proposed Tool	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

VI. Conclusion and Future Work

Several studies have been undertaken with the focus on improving the trade-off between performance and power saving in the context of embedded systems. However, most of them present some limitation. This paper presents a tool that employs Linux kernel features to group the running processes, combining load balancing and load unbalancing strategies to manage the workload between the processor cores with the focus on both increasing the performance and reducing the power consumption of embedded systems.

Experiments were conducted to evaluate the impact caused by our proposal to an embedded platform during the execution of several algorithms with different behaviors of a very known tests suite. We analyzed performance, power consumption, and Energy-Delay Product. Results showed that the proposed tool could increase the trade-off between performance and power saving of embedded systems in up to 15.49%. As future work, we intend to extend our tool to include the manipulation of other resources such as memory, disk reading and writing, and network.

References

[1] B. Guo, D. Zhang, and Z. Wang, "Living with internet of things: The emergence of embedded intelligence," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. IEEE, 2011, pp. 297–304.

[2] T. Kolpe, A. Zhai, and S. S. Sapatnekar, "Enabling improved power management in multicore processors through clustered DVFS," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[3] F. D. Rossi, M. Storch, I. de Oliveira, and C. A. F. D. Rose, "Modeling power consumption for dvfs policies," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 1879–1882.

[4] W. dos Santos Marques, P. S. S. de Souza, A. F. Lorenzon, A. C. S. Beck, M. B. Rutzig, and F. D. Rossi, "Improving edp in multi-core embedded systems through multidimensional frequency scaling," *IEEE International Symposium on Circuits and Systems*, 2017.

[5] A. Y. Zomaya and Y.-H. Teh, "Observations on using genetic algorithms for dynamic load-balancing," *IEEE transactions on parallel and distributed systems*, vol. 12, no. 9, pp. 899–911, 2001.

[6] D. M. Tullsen and S. J. Eggers, "Limitations of cache prefetching on a bus-based multiprocessor," in *ACM SIGARCH Computer Architecture News*, vol. 21, no. 2. ACM, 1993, pp. 278–288.

[7] C. B. Zilles and G. S. Sohi, *Understanding the backward slices of performance degrading instructions*. ACM, 2000, vol. 28, no. 2.

[8] H. Jeon, W. H. Lee, and S. W. Chung, "Load unbalancing strategy for multicore embedded processors," *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1434–1440, 2010.

[9] D. F. Zucker, R. B. Lee, and M. J. Flynn, "Hardware and software cache prefetching techniques for mpeg benchmarks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp. 782–796, 2000.

[10] 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 *et al.*, "The nas parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[11] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary arm and x86 architectures," in *Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, Feb 2013, pp. 1–12.

[12] B. Liu and Y. Gao, "Dynamic load balancing in embedded systems based on triplet-based hierarchical interconnection architecture," in *Mechatronic and Embedded Systems and Applications, Proceedings of the 2nd IEEE/ASME International Conference on*. IEEE, 2006, pp. 1–6.

[13] I. Jahnich and A. Rettberg, "Towards dynamic load balancing for distributed embedded automotive systems," in *Embedded System Design: Topics, Techniques and Trends*. Springer, 2007, pp. 97–106.

[14] G. Lim, C. Min, and Y. Eom, "Load-balancing for improving user responsiveness on multicore embedded systems," in *Proceedings of the Linux Symposium*, 2012, pp. 25–33.

[15] P. Bellasi, G. Massari, and W. Fornaciari, "A rtrm proposal for multi/many-core platforms and reconfigurable applications," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*. IEEE, 2012, pp. 1–8.

[16] K.-M. Cho, C.-W. Tsai, Y.-S. Chiu, and C.-S. Yang, "A high performance load balance strategy for real-time multicore systems," *The Scientific World Journal*, vol. 2014, 2014.

978-1-5386-6950-1/18/$31.00 ©2018 IEEE