Reducing Energy Consumption in Microcontroller-based Platforms with Low Design Margin Co-Processors

Andres Gomez[1,2], Christian Pinto[3], Andrea Bartolini[2,3], Davide Rossi[3], Luca Benini[2,3], Hamed Fatemi[4], and Jose Pineda de Gyvez[4]

[1]Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
[2]Integrated Systems Laboratory, ETH Zurich, Switzerland, firstname.lastname@iis.ee.ethz.ch
[3]DEI, University of Bologna, Italy, firstname.lastname@unibo.it
[4]NXP Semiconductors, Central R&D, The Netherlands, firstname.lastname@nxp.com

***Abstract*—Advanced energy minimization techniques (i.e. DVFS, Thermal Management, etc) and their high-level HW/SW requirements are well established in high-throughput multi-core systems. These techniques would have an intolerable over-head in low-cost, performance-constrained microcontroller units (MCU's). These devices can further reduce power by operating at a lower voltage, at the cost of increased sensitivity to PVT variation and increased design margins. In this paper, we propose a runtime environment for next-generation dual-core MCU platforms. These platforms complement a single-core with a low area overhead, reduced design margin shadow-processor. The runtime decreases the overall energy consumption by exploiting design corner heterogeneity between the two cores, rather than increasing the throughput. This allows the platform's power envelope to be dynamically adjusted to application-specific requirements. Our simulations show that, depending on the ratio of core to platform energy, total energy savings can be up to 20%.**

I. Introduction

In the mid-to-high performance range, symmetric multicore architectures have typically been used to achieve a higher throughput with a lower power consumption than single-core systems. Such systems exploit several architectural techniques to improve their energy efficiency proportionally to performance requirements. More recently, heterogeneous computing has been proposed for the purpose of increasing the energy efficiency for a wide range of performance targets, such as the Big-Little architectures [1]. In these systems, fine-grained DVFS can greatly improve the energy efficiency. However, this requires an advanced hardware/software infrastructure such as memory virtualization, multi-threading and full-featured operating systems, such as Linux or Android.

On the other end of the spectrum, the low-end micro controller unit (MCU) market is dominated by products based on the ARM Cortex M processor family, such as [2]. These platforms have very simple, cache-less cores and mainly optimized for low power and cost. Full-blown DVFS is generally not supported in these systems for cost reasons: a DVFS-ready switching-mode power converter (SMPS) is a complex and expensive component generally optimized for efficiency at high currents, and featuring a complex SW interface for voltage control. In the low-complexity, low-power domains the SMPSes have limited tuning range for cost and efficiency reasons. Hence these low-end systems typically rely on frequency scaling and shutdown as the main power management knobs. As multi-core architectures begin to appear in the microcontroller business segment, new task allocation schemes must be designed in accordance to its limitations, namely, the lack of first level caches and support for virtualization.

In recent years, the semiconductor industry began reducing model guardbands to reduce core area and increase energy efficiency. Though the economic viability of relaxed process variations is still an open question, researchers have been studying its impact on efficiency and performance, and have proposed both architectural and techniques to recover some of the penalties. For example, [3] shows up to 13% standard-cell area reduction from a 40% model guardband reduction.

We propose a platform which consists of two cores featuring the same architecture, but two different implementation methodologies: one designed reliably using worst-case design margins, and another designed using more energy-efficient, typical design margins. The major advantage of this approach, is that both processors can execute the same instruction set, as in a typical symmetric multiprocessing (SMP) system, and, at the same time, have the increased energy efficiency from the power heterogeneity. The main contributions of this work include a lightweight, bare-metal task allocation framework and a design space exploration that evaluates the performance/energy trade-off of our proposed solutions. Lastly, we apply the proposed techniques to several real-life applications and evaluate the energy savings.

II. Related Work

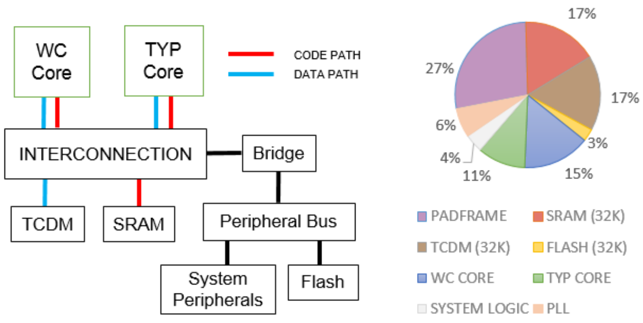
From the market viewpoint, the ARM big.LITTLE [1] family of products is the most striking example of heterogeneous multicore platforms. ARM's big.LITTLE is a heterogeneous computing architecture coupling (relatively) slower, low-power processor cores with (relatively) more powerful and power-hungry ones. The intention is to create a multi-core processor that can adjust to dynamic computing needs and use less power than frequency scaling alone. Such multicore systems depend on high-level software infrastructure to achieve energy-efficient allocation/mapping of different applications. Ideally, the operating system is able to detect how computationally demanding tasks are, in order allocate them to the correct core type. A similar approach has been proposed by the authors of [4]. They realized a single-ISA heterogeneous multi-core architecture, including four Alpha cores and a MIPS R4700. The allocation of tasks among cores is integrated as part of the operating system. Other methods besides heterogeneous architectures have been proposed for increasing energy efficiency. Near Threshold Computing, in which the supply voltage is only slightly higher than the transistor's threshold voltage, is a promising approach to reduce the energy per operation. However, this methodology is highly sensitive to parameter variations and requires a combination of architectural adaptations or specific software techniques to mitigate the effects variability-induced heterogeneity [5].

Compared to high-end systems, there has been very little attention paid to task allocation/scheduling on low-cost, limited performance systems. The closest work in this domain would be [6], which focuses on optimal resource management for control tasks in MCUs using a minimal real-time kernel. However, energy efficiency is not addressed, and it targets single-core MCUs. In contrast, our proposed solution is able introduce significant energy savings in the low-power, low-cost and deeply embedded applications context without requiring a sophisticated hardware/software infrastructure.

III. Platform Architecture

As shown in Fig. 1a, our proposed system architecture is similar to heterogeneous dual-core MCUs such as the LPC43XX series [7], except that both cores are identical from the architectural point of view. The heterogeneity is given by the physical implementation methodology adopted for each core. One core is implemented using the worst-case corners for standard cell library characterization, while the other one is implemented using typical operating conditions. In this scenario, we can assume the worst-case processor will always reach the target implementation frequency, even with process, voltage and temperature variations. On the other hand, depending on the actual process quality and operating conditions (i.e. voltage, temperature) of each die, the typical core might not reach the target frequency. It should be noted that at the reduced frequency, the functionality and reliability of the typical core is equal to that of the worst-case cores. The only difference will be that the typical core will be more energy efficient than the worst-case core, because of the larger sizing of cells and buffering required to achieve the timing closure in the worst-case corner. Rather than increase the system's throughput, we use the co-processor to offload tasks and deliver the same performance as a single-core MCU, but with substantial energy savings.

The cores share a L1 multi-banked tightly coupled data memory (TCDM) that can be accessed in one clock cycle acting as a shared data scratchpad memory. The communication is based on a cross-bar interconnect (e.g. AMBA multilayer found in typical MCU's [2]), implementing a word-level interleaving scheme aimed at reducing the access contention to TCDM banks. Moreover, the interconnect provides test and set capabilities used to implement synchronization and a mes-



(a) Hardware architecture. (b) Area breakdown.

Fig. 1. Proposed architecture and area breakdown.

sage passing mechanism between the cores. The interconnect communicates to a peripheral bus through a bridge, that allows the two cores to access the system peripherals. As opposed to mid- and high-end multi-core platforms, most processors for deeply embedded applications (e.g. ARM Cortex M series) do not feature an instruction cache. For this reason, current dual-core heterogeneous architectures feature a private, per core instruction memory. Our target architecture uses SRAM as instruction memory, which is initialized by the *WC* core with the contents of the Flash memory at boot. Since the Flash is only used once, its power consumption is not considered.

Figure 1b shows the area breakdown of the proposed dual-core MCU system. The results are estimated through synthesis on an industrial 40nm CMOS process optimized for low-power, low-leakage products. It is possible to note that a bank of Flash memory occupies only 3% of the overall area, while the TCDM and SRAM together occupy 34%. The *TYP* core by itself occupies 11% of the overall system area. Considering that an additional 2% overhead is required at the system level for introducing the additional core, the total area overhead for the introduction of the typical core is only 13%.

IV. Power Model

We divide our platform's total energy consumption into three different categories: core, memory, and system energy; the latter corresponds to all peripherals besides cores and memories. In our model, the core and system components have two different states: *active*, and *sleep*. The system component is *active* only if there is an active core, otherwise we assume that peripherals can enter a low-power *sleep* mode.

Power Estimates: We have estimated the consumption of our next-generation platform with power measurements of currently available commercial platforms fabricated in the *WC* corner, such as [7]. With information coming from product groups, we have determined that a low design margin process can produce cores in the *TYP* corner with an power efficiency improvement of up to 30% at a maximum frequency of $204MHz$. These results are similar to those obtained in [8]. The summary of the core and system power values, calculated with $F_{active} = 204MHz$ and $F_{sleep} = 12MHz$, can be seen in Table I and will be used in all our simulations, unless otherwise specified. We have scaled the access power consumption of SRAM memory to one fourth of the *WC* core, and the idle power equal to one tenth of the access power. These ratios are based on post-synthesis power simulations.

Device Type	**Power Den.** [mW/MHz]	**Active Power** [mW]	**Sleep Power** [mW]
WC Core	0.80	163.20	9.60
TYP Core	0.56	114.50	6.74
System	0.80	163.20	9.60

TABLE I
ESTIMATED POWER CONSUMPTION VALUES.

V. TASK ALLOCATION POLICIES

In this section, we derive the ideal, energy-optimal policies using our power model and we discuss generic task-level allocation techniques which reduce the total energy consumption through the use of our proposed shadow co-processor.

Task Model: Our basic unit of work is a set of instructions, which we define as a task. We assume that the number of instructions can be directly translated to an execution time by factoring the core's CPI and frequency. Since both cores run at the same frequency, there is never a performance penalty in choosing one core over the other. Task-sets group more than one task together, and can establish a precedence constraint between tasks. As is typically the case in embedded system applications, we assume a periodic task-set activation. At the beginning of each iteration, one or more tasks can be activated, depending on the task-set.

Energy-efficient Allocation: To derive the optimal allocation policy, we begin with the total energy for the dual-core platform, which can be expressed as:

$$\begin{aligned} E_{TOT,DC} = & A_{WC} * P_{\Delta,WC} + A_{TYP} * P_{\Delta,TYP} \\ & + A_{SYS} * P_{\Delta,SYS} \\ & + D * (P_{S,TYP} + P_{S,WC} + P_{S,SYS}) \end{aligned} \quad (1)$$

Where $P_{\Delta,X}$ represents the difference between the *active* and *sleep* powers of component X: $P_{A,X} - P_{S,X}$; A_X represents the amount of time component X was active, and D is the task set's period. We now introduce two auxiliary variables: $A_S = A_{TYP} + A_{WC}$ and $A_D = A_{TYP} - A_{WC}$, which are the sum and difference between the *active* times of the *WC* and *TYP* cores. By substituting these values in Eq. (1), we are able to express the total energy a function with only one variable: A_D. All other terms will be constant for a given workload and architecture. As a result, minimizing to total energy can be expressed as follows:

$$min\ E_{TOT,DC} \Rightarrow min\ \underbrace{A_D}_{var} \underbrace{(P_{\Delta,TYP} - P_{\Delta,WC} + P_{\Delta,SYS})}_{const} \quad (2)$$

Thus, the energy minimization problem is reduced to choosing the appropriate A_D^* for a given $P_{\Delta,WC}$,$P_{\Delta,TYP}$ and $P_{\Delta,SYS}$. The extreme cases are: 1) off-loading all work to the more energy-efficient *TYP* core when system power is negligible, where $A_D = A_S$, or 2) fully parallelizing the load in order to minimize the system energy when it is dominant, where $A_D = 0$. This optimization problem can be split into two cases, since $A_D \geq 0$. If the sign of the constant term is negative, then minimizing translates to selecting the largest value, or $A_D = A_S$, as was mentioned earlier. On the other hand, if the constant term is positive, minimizing requires selecting the smallest value, or $A_D = 0$. In a nutshell, the optimal distribution of tasks among cores depends only on the differences between the cores' powers and the system power. If the shadow co-processor's power savings are greater than the system energy, then all tasks should be offloaded to the co-processor. Conversely, if the system power is greater, then the optimal policy is to exploit all available parallelism, in order to maximize the sleep time, and thus minimize system energy. Our next-generation MCU-based platform will be able to determine on which of these cases it finds itself, and the task allocation framework will enforce the optimal policy, depending on the operating point.

Task Serialization: We have shown that in cases were the system power is low enough, all tasks should be allocated to the *TYP* core. The software infrastructure is kept on the *WC* core and performs very light-weight tasks such as setting timers and increasing some counters. As a result, the *WC* core will have very low utilization and contribute very little to the total energy consumption. In this case, the maximum energy savings would occur at high utilizations, near $A_{SC} = 1$, where the energy savings will be directly proportional to the ratio of *active* powers.

Task Parallelization: In the other case, where the system power is considerable, tasks should be equally distributed among the two cores. Ideally, the HW/SW infrastructure would allow fine-grained control mechanisms for any load to be equally distributed among both cores. Due to the limitations of low-cost MCU's, they can only exploit the task-level parallelism available in the application. Our proposed task allocation framework, built for the general-purpose domain, does not require any knowledge of task execution times. Naturally, if such knowledge is available, one could improve energy savings. As a result, we have implemented two different solutions: 1) Least Recently Used (*LRU*) and 2) FirstFit. In *LRU*, if both cores are available, the core that was least recently used is chosen. This policy automatically alternates between cores and leads to a relatively balanced load distribution, on average. In FirstFit, the tasks are sorted according to size, and using a first fit heuristic, distributed among both cores. This bin-packing heurisitic, by exploiting task-set information can bring practical savings closer to their ideal limit.

VI. EVALUATION METHODOLOGY

In this section, we evaluate the energy savings our allocation policies using two different simulators. We validate our initial results from the previous section, by using synthetic benchmarks to sweep the entire utilization range. Furthermore, we have ported typical MCU-based applications to our platform, and show the energy savings from real-world applications.

For our simulations, we use a modified version of [9], which leverages an instruction accurate ARMv6 instruction set simulator (ISS) to model each of the two cores, and a SystemC interface model to interconnect multiple cores and memory banks. This instruction-accurate simulator allows us to test our framework, written in C, with variable parameters on our specified hardware architecture, shown in Fig. 1a. This simulator's power model includes the cores, system peripherals and data and instruction memories.

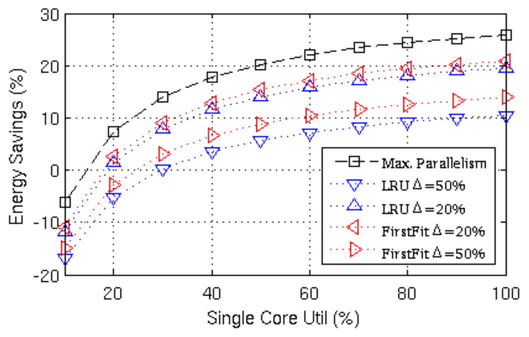


Fig. 2. Parallelization Results ($P_{SYS} = P_{WC}$)

For our initial experiments, we used synthetic tasks in order to linearly sweep the entire utilization range for a given period. For each utilization value, we instantiated a queue of $I = 100$ iterations, each iteration activating $N = 2$ independent tasks of specific lengths such that the desired utilization is kept. Each of these queues was then executed in both a single-core and a dual-core platform. We present the results as the relative energy savings with respect to the single-core.

Task Parallelization: As was previously seen, whenever the system power is *over* the threshold value $P_{\Delta,SYS} \geq (P_{\Delta,WC} - P_{\Delta,TYP})$, the most energy-efficient allocation policy is to parallelize as much as possible. We tested different levels of parallelism by explicitly introducing task length variation (Δ). When $\Delta = 0\%$, it means both tasks are of the same length. When $\Delta = 50\%$, it means tasks are $\pm 50\%$ from the mean. For example, if $T1 = T2 = 10ms$ with $\Delta = 0$, then with $\Delta = 50\%$ $T1 = 5ms$ and $T2 = 15ms$.

Fig. 2 shows the energy savings as a function of the single-core utilization. We have implemented two different task allocation policies: *LRU* and *FirstFit*, as discussed in Eq. (2). The blue lines show the energy savings for the *LRU* policy. As expected, when $\Delta = 50\%$ one task is substantially longer than the other, leading to low savings. As Δ decreases, the load can be distributed better, the system power is minimized, and the energy savings are maximized. The red lines show the energy savings for the *FirstFit* policy. This policy uses task-set knowledge to both minimize the active system energy and maximize the *TYP* core's utilization. With high variance, it has additional savings of ~5 % with respect to *LRU*.

Real-World Case Studies: We have selected three different applications that can be found in the microcontroller domain. The first application, from [10], implements zero-velocity-update INS which determines the position of an object in a space from the input of inertial sensors (i.e. accelerometers). The second application is an MP3 decoder which, as detailed in [11], can have a substantial degree of parallelism in certain subtasks. The last case study comes from the PixHawk project [12], where they execute a lightweight optical flow algorithm, used to approximate the speed of a quadcopter from changes in subsequent pictures. This application serves as a reference for maximum parallelism. These applications were ported to our instruction-accurate simulation framework, and were then executed under single-core and dual-core platforms. Using the same power numbers as before, the proposed solution had energy savings ranging 3% in the INS application to 20% in the Pix4Flow application. This was expected, as the energy savings depend on the degree of available parallelism.

VII. Conclusions

Next generation microcontroller platforms will feature an energy-efficient, low design margin shadow co-processor. In this work, we have presented a lightweight task allocation framework that exploits the available power heterogeneity to minimize the platform's total energy consumption. In cases where core power is dominant, energy is minimized by statically allocating all tasks to the more energy-efficient shadow co-processor. When the system power consumption is dominant, energy is minimized by exploiting the available task-level parallelism. We have validated our task allocation policies with instruction-accurate simulation platform. Experiments show that compared to a current single-core platform, our policies can save up to 20% of total system energy.

VIII. Acknowledgements

This work was supported by FP7 project PHIDIAS (g.a.318013) and funded by NXP Semiconductors and ETHZ Grant financing.

References

[1] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," ARM Ltd., Tech. Rep., 2011.

[2] Lpc1800 platform website. [Online]. Available: http://www.nxp.com/products/microcontrollers/cortex_m3/series/LPC1800.html

[3] K. Jeong, A. B. Kahng, and K. Samadi, "Quantified Impacts of Guard-band Reduction on Design Process Outcomes," in *IEEE ISQED*, 2008.

[4] R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Processor power reduction via single-ISA heterogeneous multi-core architectures," *IEEE Computer Architecture Letters*, vol. 2, no. 1, 2003.

[5] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "EnergySmart: Toward Energy-efficient Manycores for Near-Threshold Computing," in *ACM HPCA*, 2013.

[6] R. Marau, P. Leite, and M. Velasco, "Performing flexible control on low-cost microcontrollers using a minimal real-time kernel," *IEEE Trans. Ind. Informat*, vol. 4, no. 2, 2008.

[7] Lpc4300 platform website. [Online]. Available: http://www.nxp.com/products/microcontrollers/cortex_m4/series/LPC4300.html

[8] A. B. Kahng, R. Kumar, and J. Sartori, "Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors," *IEEE Trans. Computer-Aided Design of Integr. Circuit Syst.*, vol. 31, no. 3, 2012.

[9] D. Bortolotti, C. Pinto, A. Marongiu, M. Ruggiero, and L. Benini, "VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip," in *IEEE IPDPSW*, 2013.

[10] J.-O. Nilsson, I. Skog, P. Handel, and K. Hari, "Foot-mounted INS for everybody - an open-source embedded implementation," in *IEEE/ION Position, Location and Navigation Symposium*, 2012.

[11] W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A Practical Approach to Exploiting Coarse-Grained Pipeline Parallelism in C Programs," in *MICRO*. IEEE, 2007.

[12] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *IEEE ICRA*, May 2011.

[13] A. a. Eltawil, M. Engel, B. Geuskens, A. K. Djahromi, F. J. Kurdahi, P. Marwedel, S. Niar, and M. a.R. Saghir, "A survey of cross-layer power-reliability tradeoffs in multi and many core systems-on-chip," *Microprocessors and Microsystems*, no. 8, 2013.

[14] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite Cores: Pushing Heterogeneity Into a Core," in *MICRO*. IEEE, 2012.

[15] S. Navada, N. N. K. Choudhary, S. V. Wadhavkar, and E. Rotenberg, "A Unified View of Non-monotonic Core Selection and Application Steering in Heterogeneous Chip Multiprocessors," in *IEEE PACT*, 2013.

[16] R. Nishtala, D. Mossé, and V. Petrucci, "Energy-aware thread co-location in heterogeneous multicore processors," in *IEEE EMSOFT*, 2013, pp. 1 – 9.