Increasing the energy efficiency of microcontroller platforms with low-design margin co-processors

Andres Gomez*, Andrea Bartolini*, Davide Rossi†, Baris Can Kara‡, Hamed Fatemi‡, Jose Pineda de Gyvez‡, Luca Be

*D-ITET, ETH Zurich, Switzerland
Email: {gomeza, andrea.bartolini, lbenini}@ethz.ch

†DEIS, University of Bologna, Italy
Email: {a.bartolini, luca.benini@unibo.it}@unibo.it

‡NXP Semiconductors, Netherlands
Email: {baris.can.kara, hamed.fatemi, jose.pineda.de.gyvez@nxp.com}@nxp.com

***Abstract*—Reducing the energy consumption in low cost, performance-constrained microcontroller units (MCU's) cannot be achieved with complex energy minimization techniques (i.e. fine-grained DVFS, Thermal Management, etc), due to their high overheads. To this end, we propose an energy-efficient, multi-core architecture combining two homogeneous cores with different design margins. One is a performance-guaranteed core, also called Heavy Core (HC), fabricated with a worst-case design margin. The other is a low-power core, called Light Core (LC), which has only a typical-corner design margin. Post-silicon measurements show that the *Light* core has a 30% lower power density compared to the *Heavy* core, with only a small loss in reliability. Furthermore, we derive the energy-optimal workload distribution and propose a runtime environment for Heavy/Light MCU platforms. The runtime decreases the overall energy by exploiting available parallelism to minimize the platform's active time. Results show that, depending on the core to peripherals power-ratio and the *Light* core's operating frequency, the expected energy savings range from 10 to 20%.**

I. Introduction

In the mid-to-high performance range, symmetric multi-core 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]. A big.LITTLE multi-core is composed of one (or a set of) high performance cores (e.g. ARM Cortex A15), and a set of smaller but more power-efficient cores (e.g. ARM Cortex A7). In this way, if the low-power cores can satisfy an application's performance requirements, it is possible to achieve significant energy savings. In these systems, the use of advanced hardware infrastructure such as fine-grained Dynamic Voltage and Frequency Scaling (DVFS) can further improve the energy efficiency. These systems, however, require memory virtualization, multi-threading and full-featured operating systems, such as Linux or Android. If these requirements are met, task allocation and voltage/frequency configuration processes can be easily integrated with the operating system.

On the other side of the spectrum, the low-end micro controller unit (MCU) market is dominated by products based on the ARM Cortex M processor family. These platforms have very simple cores, mainly optimized for low power and cost. They are cache-less and do not support memory virtualization, limiting them to either bare-metal applications or very basic operating systems. Full-blown DVFS is generally not supported for cost reasons: a DVFS-ready switching-mode power converter (SMPS) is a complex and expensive component usually optimized for efficiency at high currents, and featuring a complex SW interface for voltage control. In the low-complexity, low-power domains the SMPS's have limited tuning range for cost and efficiency reasons. More flexible and inexpensive LDO's have significant losses when downconverting, which negates most of the savings expected by voltage scaling. Hence, these low-end systems typically rely on frequency scaling and shutdown as the main power management knobs. Furthermore, logic synthesis is problematic for low-power IC's with ultra-wide voltage and frequency scaling, since the behavior of synthesis tools for timing optimization at a high voltage and an ultra-low-voltage are so conflicting that convergence in one scenario can create violations in other scenarios [2]. This translates to further inefficiencies in the final device.

At the same time, multi-core architectures are beginning to penetrate the microcontroller business segment: recently, a new class of heterogeneous dual-core MCU products appeared in the market. These devices have cores with different instruction set architectures [3] and rely on the architectural heterogeneity to achieve energy efficiency with a principle similar to the mid-to-high-end Big-Little multi-cores described previously, but with some limitations. The

restrictions imposed by the simple nature of the processor architectures, namely the lack of first level caches and support for virtualization, impose two big restrictions in the task allocation policies. First, tasks must be statically partitioned at design time, since processors execute different instruction sets. Second, the limited-performance nature of MCUs does not support the computational load required to implement complex feedback loop allocation policies, as well as voltage and frequency tuning. In recent years, the semiconductor industry has started to reduce model guardbands as a way to decrease core area and increase energy efficiency for harvesting-based applications such as [4], [5]. 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, [6] have shown up to 13% standard-cell area reduction from a 40% model guardband reduction. In [7], the authors pair a better-than-worst-case design core with recovery-driven techniques and report power savings in the range of 11.8% to 29.1%. We will focus on exploiting such heterogeneity in platforms consisting of two cores featuring the same architecture, but two different implementation methodologies: one designed reliably using worst-case (*Heavy*) design margins, and another designed using more energy-efficient, typical (*Light*) design margins. It should be noted that while a *Light* core was designed to be clocked at F_{max}, because of process variations, the is a small probability that it will not reach this operating frequency. There are different ways to handle this scenario, if it were to happen. One would be to "overclock" the *Light* core to run at F_{max}, at which point, error detection and correction schemes would have to be implemented in order to guarantee its functionality, introducing non-negligible overheads. A second, simpler alternative, which we use, is to reduce its operating frequency. A reduced frequency will prevent any errors from occurring, meaning that no additional mechanism will be necessary to guarantee its functionality, albeit with a clear performance trade-off. 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. In this work, we propose a simple, yet efficient allocation policy that can dynamically select which processor will execute each task, depending on the application workloads and the operating conditions, in order to achieve the most energy savings. The main contributions of this work are:

1) A lightweight, bare-metal task allocation framework that supports both static and dynamic policies without relying on an advanced operating system.
2) A simulation infrastructure for the system which can model the execution of applications, with events and shared memory typical of this domain, and profile their energy consumption.
3) Derivation of the task allocation policies that minimize the total energy consumption under both homogeneous and heterogeneous frequencies.
4) A design space exploration that evaluates the performance/energy trade-off, as well as the allocation policies with respect to parameters typical of this domain.
5) Analytical and experimental extensions to evaluate energy savings with a reduced frequency *Light* core.
6) Calculation of energy-saving F_{TYP} ranges as a function of the utilization.
7) Reliability model based on silicon measurements of a Heavy/Light platform fabricated using 40nm CMOS technology.
8) The analysis of the system and the evaluation of the energy savings from exploiting heterogeneity with different allocation policies in real-life applications.

With respect to our previous work [8], we have lifted the homogeneous frequency conditions, and we have calculated the boundary conditions for which a reduced frequency *Light* core would introduce energy savings compared to a single core. Our post-silicon measurements show that this is an unlikely case, most *Light* core are statistically able to reach their target frequency. The remainder of this paper is organized as follows: In the next section, we review various approaches used in other systems to increase energy efficiency. In Section III, we present the proposed architecture of our next-generation dual-core platform. In Section IV, we describe the power model for our proposed system in detail. In Section V, we derive the energy-optimal task allocation policies under the condition of a homogeneous frequency. In Section VI, we relax this condition to allow a reduced frequency *Light* core, and we calculate the frequency ranges for which the dual-core platform will have savings. We present our experimental results in Section VII, and conclude our findings in Section VIII.

II. Related work

In this section, we position our work in relation to similar approaches from existing literature. From the market viewpoint, the ARM big.LITTLE [1] family of products is the most striking example of heterogeneous multi-core 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. It includes Cortex-A15 cores and Cortex-A7 cores, designed to be ISA-compatible and fit into the big.LITTLE system. This multi-core system depends 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 [9]. They realized a single-ISA heterogeneous multi-core architecture, including a single-threaded version of the EV8 high performance processor (Alpha 21464), the MIPS R4700 (a processor targeted at very low-power applications), the EV4 (Alpha 21064), EV5 (Alpha 21164), and EV6 (Alpha 21264).

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 [10]. Near Threshold Computing, in which the supply voltage is only slightly higher than the transistor's threshold voltage, is a promising approach to reducing 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 [11]. Several works have proposed variability-aware techniques that improve the predictability and energy efficiency of parallel multiprocessor arrays. The authors of [12] propose a workload allocation policy to mitigate the effects of core-level performance and power variations. Using an LP+Bin-Packing formulation, their online policy minimizes the energy consumption and deadline violations in multi-core platforms with power and performance variation. In [13], the authors propose a variation-tolerant tasking technique for processor clusters. By characterizing dynamic, circuit-level variability in specific, tightly-coupled data structures, their proposed runtime implements a task-level errant instruction management technique to reduce the error-recovery cost and increase cluster throughput. The authors of [14] propose an architectural scheme to tolerate ambient temperature-induced variations capable of statically (off-line) and dynamically (on-line) adapting the processor-to-L1-memory latency without compromising execution correctness.

At the software level, determining how to appropriately allocate tasks to the different available cores is of upmost importance in heterogeneous systems [15]. Otherwise, results will lead to reduced performance and possibly an increase in energy. As a result, much attention devoted to developing allocation techniques for optimizing different objectives [16]. Generally speaking, allocation techniques can be classified as offline or online. Offline algorithms take all decisions before execution, while online algorithms take decisions after task activation. Offline algorithms typically require more knowledge about the workload and can often leverage more computational power to determine optimal solutions. The work presented in [17] is one such example, where the authors propose projecting the core configurations and the program's resource demands into a unified multi-dimensional space, where the program-core matching is obtained with weighted euclidean distances. Online algorithms are more flexible because they do not require any previous knowledge and can dynamically adapt to changing conditions. The authors of [18] propose a simulated annealing-based heuristic to maximize energy efficiency using dynamic power management and dynamic voltage and frequency scaling in multi-core platforms. Similarly, the energy-aware scheduling algorithm proposed in [19] reduces consumption by re-utilizing slack time in DVS-enabled platforms.

The authors of [20] show a non-clairvoyant, preemptive online scheduling algorithm and show that it is scalable for the objective of unweighted flow plus energy on speed-scalable processors. The work presented in [21] studies a hierarchical power management methodology for asymmetric multi-core architectures. Their control-theory centric approach uses DVFS and thread migration to achieve optimal power-performance efficiency while respecting the thermal design power budget. All of these works use advanced software infrastructure such as multi-threading, thread migration and other operating system facilities making them unsuitable for the ultra low power applications [22]. MCU-based platforms need lightweight implementations to reduce their performance overhead and maximize an application's energy efficiency.

In the MCU domain, the most relevant example of heterogeneous multi-core architecture is the LPC54000 platform [3]. The chip includes one (more powerful) ARM Cortex-M4F processor core assisted by one (more energy efficient) ARM Cortex-M0+ coprocessor. It should be noted that while this is not a single-ISA platform, the Cortex-M4 is able to execute M0 binaries, though not vice-versa. The platform is designed to allow developers to statically partition tasks to each core at design time. In this way, tasks can be strategically positioned to match the application's performance requirements in an energy efficient way. However, this platform does not allow for dynamic scheduling of tasks among the two cores, mainly because of the different ISAs and the asymmetrical nature of the system. The approach proposed in this work overcomes this issue by using an SMP architecture with heterogeneous power consumption. The co-processor could be used for more energy-efficient task execution, or to exploit any available parallelism to speed up execution and allow the system to enter sleep mode earlier. 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 [23], which focuses on optimal resource management for control tasks in MCUs using a minimal real-time kernel. However, it does not address energy efficiency, and targets single-core MCUs.

With respect to the presented systems, this work proposes and implements different task allocation policies targeting low-end MCU-based platforms. These policies can introduce important energy savings, without a sophisticated software infrastructure, in the low-power, low-cost and deeply embedded applications context.

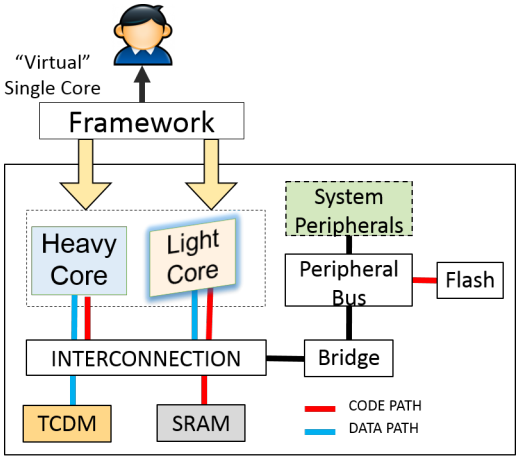


Figure 1: Hardware architecture of the heavy-light platform.

III. Platform architecture

This section describes the reference heterogeneous system architecture. The architecture, shown in Figure 1, is inspired by heterogeneous dual-core MCUs, such as LPC54XX series [3].

As shown in Figure 1, the platform is composed of two simple cores (i.e. ARM Cortex M4), equivalent from the architectural viewpoint. The platform heterogeneity is therefore given by the physical implementation methodology adopted for each core, rather than the ISA like in current products. 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 that the worst-case (*Heavy*) processor is always able to reach the target implementation frequency, even in the presence of process, voltage and temperature variations. However, depending on the actual process quality and operating conditions (i.e. voltage, temperature) of each die, the typical (*Light*) core is not guaranteed to reach the target frequency. Nonetheless, the *Light* core will be more energy efficient than the *Heavy* (Worst-Case) core because of the larger sizing of cells and buffering required to achieve the timing closure in the worst-case corner with respect to the typical corner. This heterogeneous implementation methodology guarantees that the proposed Heavy/Light platform will always be able to deliver the same performance as a single-core MCU designed in worst-case operating conditions. In addition, it allows for energy savings whenever tasks can be offloaded to the more energy-efficient *Light* core.

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 high-bandwidth logarithmic interconnect (INTERCONNECT), 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 message 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 far as the instruction path is concerned, mid- and high-end multi-core platforms typically feature a private, per-core instruction cache. When moving to the MCU domain, this assumption is no longer realistic, as 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 the TCDM for both data and memory. At boot time, the *Heavy* core copies the contents of the Flash memory (32k) to the TCDM. Since the Flash memory is only used once, we do not consider its power consumption.

As was reported in our previous work [8], the area breakdown was estimated through synthesis on a 40nm technology low-power standard cell library. The bank of Flash memory (32K) occupies only 6% of the overall area. The core implemented using a typical corner for timing closure occupies 12% of the overall system area. Considering that an additional overhead of 2% is required at system level for the introduction of one additional core, the total area overhead for the introduction of the typical core is only 13.9%.

IV. Preliminaries

In this section, we begin by describing our power model, as well as our estimated power consumption for all components. In addition, we provide a reliability model for the *Light* core, characterizing the probability of being able to reach the target operating frequency for different environmental conditions (i.e., temperature, voltage supply). For power simulations, it is first assumed that both the *Heavy* and *Light* cores can be clocked at the maximum frequency. Later on, in Section VI-A, the differences for heterogeneous frequencies will be described in detail.

A. Reliability model

Industrial flows for implementation of digital devices leverage design margins to deal with process, voltage and temperature variation. For the 40nm CMOS technology considered in this work the nominal operating point is defined as 1.1V of supply voltage, 25°C of ambient temperature and typical process condition. Contrarily, the corners used for signoff consider a combination of parameters that slow-down the chip for setup timing checks, and a combination of the parameters that speed-up the chip for hold timing checks. Table I summarizes the signoff corners of the 40nm technology used in this work (*Heavy* core).

If we now only consider process variations, the signoff margins of the worst-case corner are defined by degrading

Table I: Signoff corners used for timing closure of Worst-Case (*Heavy*) core and Typical (*Light*) core.

Signoff corners	Process (PMOS-NMOS)	Supply voltage	Temperature
Setup (*Heavy* core)	Slow-slow (-3 σ)	0.99	125°C
Setup (*Light* core)	Typical-typical	0.99	125°C
Hold	Fast-fast (+3 σ)	1.21	-40°C

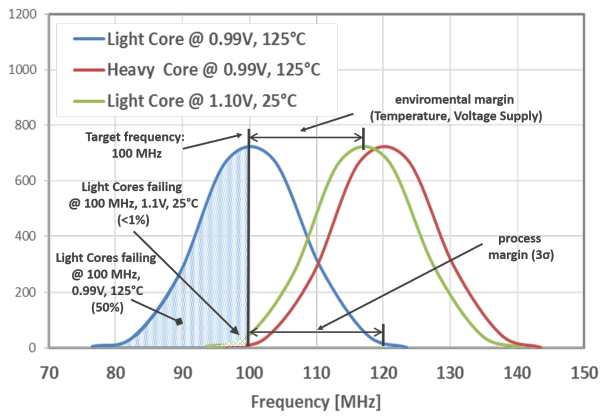


Figure 2: Frequency distribution of *Heavy* core operating at worst case environmental conditions (0.9V, 125°C), *Light* core operating at worst case environmental conditions (0.9V, 125°C), and *Light* core operating in typical operating conditions.

all the spice parameters of the transistors by 3-sigma of their Gaussian distribution. This guarantees that in the extremely unlikely case where the chips are fabricated with slow process conditions, and they operate in worst-case environmental conditions (i.e. VDD = 0.99V, temperature = 125°C), 99.7% of those are still able to achieve the target frequency. On the other hand, when chips are fabricated with a proper process centering and they operate in a typical environment the failing probability is orders of magnitude smaller. This scenario is depicted in Figure 2, where we consider a target frequency of 100 MHz for the *Heavy* core. Considering the implementation methodology we propose for the *Light* core, where the worst process corner is replaced by the typical one, the Gaussian distribution of the *Heavy* core frequencies is shifted to the left by 3-sigma, being centered on 100 MHz. In other words, implementing a core with a target frequency of 100 MHz using typical process parameters, leads to a signoff frequency of 120 MHz, if we consider worst-case process parameters for signoff. These results are obtained by fabricating a silicon prototype implementing a Cortex-M4 core using a typical corner implementation methodology, and measuring the operating frequency of the samples produced in wafers doped to emulate typical and slow process conditions. As a consequence, the failing probability of a *Light* core (i.e. not be able to reach the target frequency) operating in worst-case environmental conditions (i.e. VDD = 0.99V, temperature = 125°C) is 50%, while it is less than 1% when it operates with typical environmental conditions (i.e. VDD = 1.1V, temperature = 25°C). This demonstrates that in most common cases the *Light* core is able to operate at the same frequency as the *Heavy* core, while providing significant energy boost thanks to less buffering required to fix setup time.

B. Power estimates

The power consumption and operating frequency of the cores that implement the proposed system architecture, as described in Fig. 1, are based on measurements performed on silicon prototypes that integrate a Cortex-M4 processor architecture. The *Heavy* core prototype was designed with a state-of-the-art methodology (worst-case process corner for setup checks), while the *Light* core prototype was designed with the methodology described in section IV.A (typical process corner for setup checks). Our measurements show that the dynamic power density (μW/MHz) of the *Light* core is 30% smaller than the power density of the *Heavy* core, thanks to the smaller sizing of buffers and logic cells required to fix setup timing during implementation. In the CMOS 40nm low-power technology chosen for implementation of the processor, which is a typical technology for the MCU domain, the impact of leakage power is negligible, hence it is neglected in the rest of our exploration. The power consumption of the other system components is estimated by performing simulations on a post place and route netlist of the system implemented with the same 40nm technology for the digital blocks (e.g., timers, SPI, UART), and by measuring the power consumption on real silicon prototypes for the analog blocks (e.g. PLL, IO PADs). In these simulations and measurements, the system components accounted for up to 50% of the energy consumed in a single-core platform implemented with a worst-case methodology. Since the activity of the system components heavily depends on the number of active peripherals and analog blocks, which is heavily application dependent, in the rest of the paper we will conduct a parametric exploration of the system power for values ranging from 0% (ideal case) to 100% of the power consumption of the *Heavy* core. The power numbers

Table II: Estimated power values with F_{active} =100MHz and $F_{sleep} = F_{active}/17$ =5.8MHz.

Device type	Active power (mW)	Sleep power (mW)	Time for synthesis closure
Heavy core	5.841	0.343	31 min.
Light core	4.088	0.240	26 min.
Sys. peripherals	5.841	0.343	–

used in the rest of the paper are shown in Table II. For experiments with a reduced F_{LC} , the 17:1 active to sleep power ratio will be kept constant.

C. Power model

We divide our platform's total energy consumption into three different categories: core, memory, and system energies; the last of which 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.

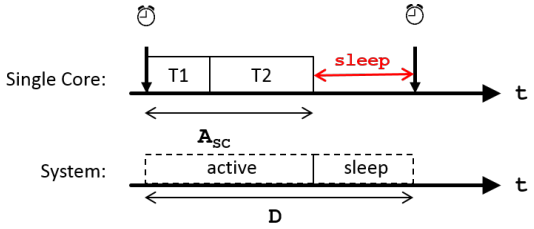
Figure 3 shows an example of two periodic, independent tasks (T1 and T2), executing on a single-core $_{(SC)}$ and dual-core $_{(DC)}$ platform. Let these tasks be executed in parallel on the dual-core platform. The task running on the *Heavy* core will have an active (execution) time A_{HC}, and the other, A_{LC}. The system component must remain active while there are active cores, meaning its *active* time is $A_{SYS} = max(A_{HC}, A_{LC})$. By multiplying the *active* time with the *active* power for each component, we can obtain their *active* energies. Similarly, we can calculate the *sleep* energy from the *sleep* time (obtained by subtracting the *active* time from period D) and *sleep* power. The energies per component for the dual-core platform, which are the sum of the *active* and *sleep* energies, are defined in equation 1.

$$E_{HC} = A_{HC} * P_{A,HC} + (D - A_{HC}) * P_{S,HC} \quad (1a)$$
$$E_{LC} = A_{LC} * P_{A,LC} + (D - A_{LC}) * P_{S,LC} \quad (1b)$$
$$E_{SYS,DC} = A_{SYS} * P_{A,SYS} + (D - A_{SYS}) * P_{S,SYS} \quad (1c)$$

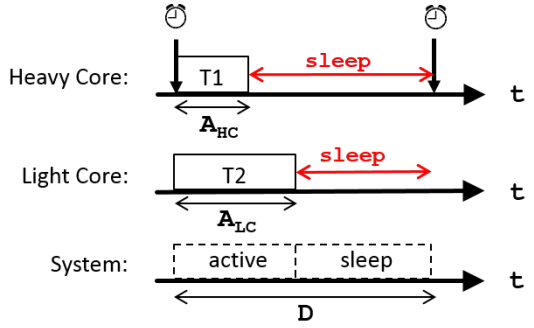
Where $P_{A,X}$ is the *active* power for the component X. Similarly, $P_{S,X}$ is the *sleep* power for component X. The total platform energy in the dual-core case, $E_{TOT,DC}$ can be approximated as the sum of the three previous energies, which are the power-dominant components in a typical MCU-based platform. Correspondingly, in the single-core case, $E_{TOT,SC}$ is the sum of the single *Heavy* core and its respective system energy. Note that E_{SC} is similar to (1a), the only difference being that A_{SC} accounts for the single-core execution of T1 and T2; equivalently, $E_{SYS,SC}$ has $A_{SYS} = A_{SC}$.

$$E_{TOT,DC} = E_{HC} + E_{LC} + E_{SYS,DC} \quad (2a)$$
$$E_{TOT,SC} = E_{SC} + E_{SYS,SC} \quad (2b)$$

There are two main factors that determine the total energy consumption at a given voltage/frequency operating point: 1) the inherent power consumption of the different components, and 2) the distribution of instructions to be executed by the processing elements. Since we do not take into account any thermal variations, the first factor is a constant, whose estimates will be addressed in the next subsection. The second factor is our input variable, which will be controlled online by our software infrastructure.



(a) Single-core Platform



(b) Dual-core Platform

Figure 3: Sample execution of two tasks on a single- and dual-core platforms.

D. Task model

Our basic unit of work is a set of instructions. We first use a fluid model to derive the optimal distribution of cycles between the Heavy/Light cores. This model will then be extended to atomic tasks, which will restrict the way instructions can be partitioned among cores. We directly calculate the execution time by factoring the number of instructions with the core's *cpi* and frequency. While the actual *cpi* value depends on the application, we are comparing a single-core to a dual-core execution. If, for a given application, the *cpi* value is the same in both, then the comparison is valid for any value. Since the *Light* and *Heavy* cores have exactly the same microarchitecture, we assume this to be true. When both cores run at the same frequency, a task's execution time will be the same in either core, otherwise the execution scales linearly. Task-sets group more than one task together and can establish a precedence constraint between tasks. If tasks have precedence constraints, they are called dependent, otherwise they are independent. 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.

V. Energy optimal allocation under homogeneous frequencies

In this section, we derive the ideal, energy-optimal policies for our proposed dual-core platform. Starting from the power model introduced in the Section IV, we calculate

the conditions for each policy. Afterwards, we discuss more realistic, generic task-level allocation techniques which reduce energy through the use of our proposed shadow co-processor. Moreover, we discuss how these task allocation policies can be refined for further energy savings by using knowledge regarding the task-set.

To derive the optimal allocation policy, we begin with the total energy for the dual-core platform as defined in (2a). Then, by substituting the terms in (1) and re-arranging the *active* and *sleep* terms, the total energy can be expressed as follows:

$$\begin{aligned} E_{TOT,DC} =& A_{HC} * P_{\Delta,HC} \\ &+ A_{LC} * P_{\Delta,LC} \\ &+ A_{SYS} * P_{\Delta,SYS} \\ &+ D * (P_{S,LC} + P_{S,HC} + P_{S,SYS}) \end{aligned} \quad (3)$$

Where $P_{\Delta,X}$ represents the difference between the *active* and *sleep* powers of component X: $P_{A,X} - P_{S,X}$. We now introduce two auxiliary variables: $A_S = A_{LC} + A_{HC}$ and $A_D = A_{LC} - A_{HC}$, which are the sum and difference of the *active* times in the *Heavy* and *Light* cores. By substituting these values in (3), the total energy can be re-written in the following manner:

$$E_{TOT,DC} = D * (P_{S,LC} + P_{S,HC} + P_{S,SYS}) \quad (4a)$$

$$+ \frac{A_S}{2}(P_{\Delta,LC} + P_{\Delta,HC} + P_{\Delta,SYS}) \quad (4b)$$

$$+ \frac{A_D}{2}(P_{\Delta,LC} - P_{\Delta,HC}) + \frac{|A_D|}{2}P_{\Delta,SYS} \quad (4c)$$

Where the term $|A_D|$ in (4c) can simply be rewritten as A_D since $A_D \in [0, A_S]$. This is intuitive, since the extreme cases of energy minimization are: 1) off-loading all work to the more energy-efficient *Light* 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 its power is dominant, where $A_D = 0$. Since the right-hand terms in (4a) are constants, and terms in (4b) are independent of task allocation, the minimization of the total energy can be expressed as follows:

$$min\ E_{TOT,DC} \Rightarrow min\ \underbrace{A_D}_{var}\underbrace{(P_{\Delta,LC} - P_{\Delta,HC} + P_{\Delta,SYS})}_{const} \quad (5)$$

Thus, the energy minimization problem reduces to choosing the appropriate A_D^* for a given $P_{\Delta,HC}$,$P_{\Delta,LC}$ and $P_{\Delta,SYS}$. Minimizing the product of these terms 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 turns to selecting the smallest value, or $A_D = 0$.

$$A_D^* = \begin{cases} A_S & \text{if } (P_{\Delta,LC} - P_{\Delta,HC} + P_{\Delta,SYS}) < 0 \\ 0 & \text{if } (P_{\Delta,LC} - P_{\Delta,HC} + P_{\Delta,SYS}) \geq 0 \end{cases} \quad (6)$$

In the end, 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 in which of these cases it finds itself, and the task allocation framework will enforce the optimal policy, depending on the operating point.

Task serialization: In deriving (6), we have shown that in cases were the system power is low enough, all instructions should be allocated to the *Light* core. For reliability purposes, the software infrastructure remains on the *Heavy* core and performs very light-weight tasks such as setting timers and increasing some counters. As a result, the *Heavy* 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, or 24%. Table III shows the energy savings from off-loading all tasks to the *Light* core. In this example, we have a fully utilized single-core ($D = A_{SC} = A_{HC} = 100ms$) and no system power is considered. In the dual-core, the *Heavy* accumulates sleep power, but because of the energy efficiency of the *Light* core, the total energy is 24% less than in the single-core.

Table III: Dual-core energy savings (without system peripherals)

Platform type	A_{HC} (ms)	A_{LC} (ms)	**Energy**(μJ)		
			Heavy	*Light*	**Total**
Single-core	100	-	584.1	–	584.1
Dual-core	0	100	34.3	408.8	443.1

Task parallelization: As was shown in (6), in cases where the system power is considerable, instructions should be equally distributed among the two cores. Ideally, the HW/SW infrastructure would allow fine-grained control mechanisms for any load to be precisely distributed among both cores. Since low-cost, performance-constrained MCU's do not have such infrastructure available, they can only exploit the task-level parallelism provided by the application.

A. Measuring Energy Savings

In order to have a global view of the total energy savings derived from introducing a low design margin co-processor, we define the following function:

$$E_{savings}(A_{SC}, P_{SYS}) = 100 * \frac{E_{TOT,SC} - E_{TOT,DC}}{E_{TOT,SC}}[\%] \quad (7)$$

This function compares the total energy spent executing a given number of instructions on a single-core and our next-generation dual-core platform. For a given single-core utilization, defined as $U_{SC} = A_{SC}/D$, there are infinite ways to split instructions among two cores. We thus introduce an auxiliary variable, *Util Ratio*, defined as

A_{HC}/A_{LC}. This variable illustrates how the instructions are distributed among the *Heavy* and *Light* cores. For example, *Util Ratio* =0 means all instructions are executed on the *Light* core. On the other hand, *Util Ratio* = 1, means a given number of instructions was evenly divided among both cores. As was mentioned earlier, our proposed low design margin co-processor will not be used to increase the platform's throughput, but rather to increase the energy efficiency for a given single-core utilization. This is achieved by reducing the system's active time, while still maintaining the same number of instructions being executed. Figure 4 shows the energy savings as a function of the system power, and the *Util Ratio*, after setting $U_{SC} = 1$ and $A_{SC} = A_{HC} + A_{LC}$. The power numbers used for the different components can be seen in Table II. It has been

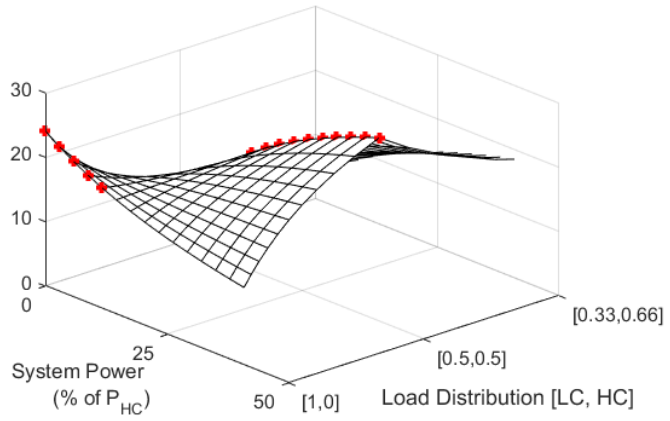


Figure 4: Energy savings for homogeneous frequencies as a $U_{SC} = 100\%$. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

shown already that for low system power values, the highest energy savings are achieved by executing all instructions on the *Light* core (*Util Ratio* =0). This can be seen by the red points on the left hand side, which represent the maximum savings when *Util Ratio* is 0. Similarly, is has been shown that, when $P_{\Delta,SYS} \geq P_{\Delta,HC} - P_{\Delta,LC}$, dividing the load equally among both cores yields the greatest savings. This can be seen by the red points on the right hand side of the figure, which represent the highest energy savings with *Util Ratio* equal to 1.

B. Effects of task variance

Task-level parallelism varies heavily with the type of application. Some applications can be very easily divided into more or less equal sub-tasks, in terms of instructions. In others, this is not the case. Table IV shows the changes in total energy consumption when two tasks have high variance, and when they are equal. For this example, we are splitting a task with full utilization and a single-core execution time ($D = A_{SC} = 100ms$). Depending on the type of task it might be split-up differently, but the equation

Table IV: Effect of task variance on total active energy for dual-core platforms.

Task variance	A_{HC} (ms)	A_{LC} (ms)	**Total energy**(μJ)			
			Heavy	*Light*	System	**Total**
High	90	10	525.6	40.88	525.6	1092
High	10	90	58.41	367.92	525.6	952.0
None	50	50	292.0	204.4	292.0	788.5

$A_{SC} = A_{HC} + A_{LC}$ will always hold, since Heavy-Light platforms increase the energy efficiency while maintaining the same throughput.

The two rows in Table IV with high variance show the impact smart allocation can have on the total energy. In the top row, the larger task was allocated to the *Heavy* core, while in the bottom row, the larger task was allocated to the *Light* core. It should be noted how the total energy drops around 11% when the large task is allocated to the more energy-efficient *Light* core, even though the system energy is exactly the same. Lastly, when there is no task variance, the task is evenly split and it minimizes the system energy component; consequently, the total energy is minimized.

Without execution time knowledge: As was seen in Table IV, having tasks with very different execution times can prevent practical savings from reaching their ideal limit. Our framework, built for general purposes, is able to dynamically allocate tasks without any previous knowledge of task execution times. This facilitates the developer's work when porting an application to our framework. For these cases, a *least recently used LRU* policy has been developed. After tasks are activated and placed in a queue, they will be allocated in a first-in-first-out (FIFO) fashion to an available core. If both cores are idle, the policy chooses the core that has been idle the longest. This policy alternates the allocated core for each task in the queue, and leads to a more balanced load distribution, on average. However, if, for example, the are only two tasks: one long, one short, then allocating the longer task to the *Light* core will lead to more savings. These additional energy savings can only be achieved with knowledge of the task-set's bounds.

With execution time knowledge: As previously discussed, exploiting information about the task set can bring practical savings closer to their ideal maximum. For cases where the execution time distribution is known, or tight bounds can be calculated offline, this information can be used to sort tasks by their length, and distribute their load in such a way that both the system sleep time is maximized and the *Light* core's utilization is simultaneously maximized. We approximate this bin-packing problem with a *First Fit* strategy. It should be noted that the programmer does not necessarily need to manually specify this information. Our proposed framework is able to gather profiling information and build statistics from the tasks. In cases were there execution time does not fluctuate significantly, this would be enough to obtain additional savings.

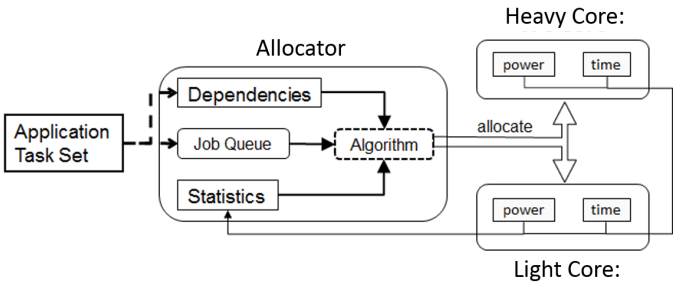


Figure 5: The software allocator

C. Task dependencies

So far, we have discussed a fluid task model to derive the energy-optimal distribution of instructions among Heavy-Light core. We have also shown how an atomic task model with discrete sizes can prevent an application from reaching the theoretical energy minimum. While we have treaded these atomic tasks to be independent, our proposed algorithms can be used to run generic task sets with dependencies. Though we cannot prove our heuristics can minimize energy in the general case, since it is a well-known NP-Hard problem, we will show with real-world case studies that they can still be used to achieve results close to the ideal scenario.

D. Software infrastructure

We describe here the implementation of the static and dynamic software allocation policies implemented in our framework, whose overview can be seen in Figure 5. For dynamic allocation, we include both the case with task-set knowledge (*First Fit*) and without (*LRU*).

Legacy code: When designing our framework, special care was taken to facilitate the developer's effort in porting legacy code to our proposed platform. Listing 1 shows an example application with three independent tasks: taskA, taskB, and taskC. The infinite loop in lines 6–11 performs the periodic activation of these tasks. To port this code to our platform, only the function calls have to be modified, as shown in Listing 2. The master core creates a reference table for each task, and with the *schedule()* function, automatically activates the tasks at the given period. This same function automatically determines the optimal allocation policy, according to the platform's characteristics, and offloads tasks to the *Light* core.

Listing 1: Legacy Code

```
01 #include "tasks.h"
02
03 int main() {
04   ...
05
06   while (1) {
07     taskA();
08     taskB();
09     taskC();
10     sleep();
11   }
12 }
```

Listing 2: Framework's API

```
13   #include "tasks.h"
14
15 int main(void) {
16   #ifdef MASTER
17   addTask(&taskA, period);
18   addTask(&taskB, period);
19   addTask(&taskC, period);
20   #endif
21   while (1) {
22     schedule()
23     sleep();
24   }
25 }
```

Dual-core task activation: The allocation of tasks to cores is handled using a mechanism based on counters. We associate an arrival and completion counter to each of the tasks, which are used respectively to notify cores for the arrival of a task and, after the execution, to signal its completion. At the beginning of each period, the software allocator assigns tasks according to the predefined schedule and notifies the respective cores of the task arrivals. When the execution of the task-set is finished, the *Light* core goes into sleep mode, waiting for the next iteration of task arrivals. The *Heavy* core, in charge of task registration, will go to sleep until it is automatically awakened at the next period.

In the case of dynamic allocation without task-set knowledge, tasks are allocated using the *LRU* policy. Since both cores can potentially access the arrays at the same time, we associate a lock (mutex) to each element pair (to each task). Locks are implemented using the T&S functionality provided by the interconnect. This hardware mechanism can be used to enable both single and two-phase synchronization barriers accessible though software primitives [24].

Finally, our framework supports task-sets with task dependencies. When a certain precedence between tasks is to be ensured, a core able to execute a specific task has to wait until the lock of the preceding task is free. In this way, we can enforce that the previous task must be completed before the next task begins execution.

VI. Energy optimal allocation under heterogeneous frequencies

In this section, we derive the energy-optimal policies for the proposed dual-core platform. As opposed to the previous section, the frequencies for the *Heavy* and *Light* will no longer be required to be equal. This would allow the use of a *Light* core even at a reduced frequency, without limiting the maximum frequency of operation for the *Heavy* core. To this end, the power model presented in Section IV for homogeneous frequencies will be extended. Afterwards, we discuss the ideal load distribution to minimize the total energy. Finally, we will derive the *Light* core's frequency boundaries as a function of the system's utilization, for which the dual-core saves energy.

A. Updated power model

In the homogeneous frequency case, there was an equivalence in the total activity of the single-core and dual-core: $A_{SC} = A_{LC} + A_{HC}$. This meant that, regardless of how the load was distributed among the *Light* and *Heavy* cores, the sum of their execution times was equal to the execution time of the single-core - due to the equal frequencies. In the heterogeneous frequency case, however, this time equivalence is no longer valid due to the reduced *Light* frequency, yielding the following inequality: $A_{SC} \leq A_{LC} + A_{HC}$. Nonetheless, the total number of instructions executed does remain constant. From this, the minimal execution time can be calculated by multiplying the cpi, which, for a given application, is constant for both cores, and the *Heavy*'s maximum frequency F_{MAX}, another constant. This expression is similar to the one derived in the homogeneous case:

$$\frac{I_{SC} * cpi}{F_{MAX}} = \frac{I_{HC} * cpi}{F_{MAX}} + \frac{I_{LC} * cpi}{F_{MAX}} \tag{8}$$

By applying the definition of activity, and taking into account that the single-core and *Heavy* core run at F_{MAX}, the previous equation is equivalent to:

$$A_{SC} = A_{HC} + A_{LC,MIN} \tag{9}$$

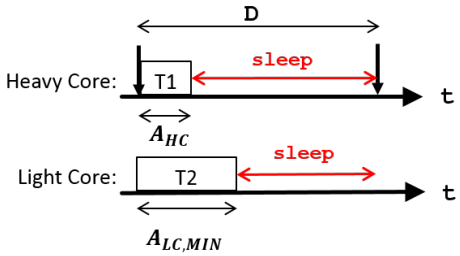
Where $A_{LC,MIN} = \frac{I_{LC} * cpi}{F_{MAX}}$, represents the minimum activity the *Light* core as if it were running at F_{MAX}. Similarly, the activity of the *Light* core running at a reduced frequency is $A_{LC} = \frac{I_{LC} * cpi}{F_{LC}}$, which can be expressed as:

$$A_{LC} = \frac{F_{MAX}}{F_{LC}} * A_{LC,MIN} \tag{10}$$

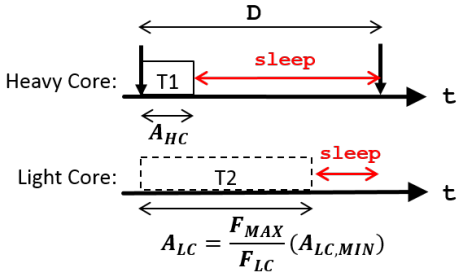
Figure 6 shows the expanded activity in the *Light* core as a result of the frequency scaling. Figure 6a shows the homogeneous case, where both the *Heavy* and *Light* cores can be clocked at the maximum frequency, resulting in the minimum activity. Figure 6b shows the heterogeneous case, where the *Light* core executes at a reduced frequency resulting in an expansion of its active time. While the energy spent executing a task remains constant, an increased execution time can lead to a higher accumulation of system energy. As a result, the criteria for the energy optimal allocation will depend not only on the power ratios, as in the homogeneous case, but on the frequency ratios as well.

B. Minimizing dual-core energy

We now set the foundation for the optimal allocation of tasks for dual-core platforms with heterogeneous frequencies by minimizing the dual-core's energy. As with the homogeneous case, we begin with the expression for the total energy, (3). Since A_{SYS} is defined as $max(A_{LC}, A_{HC})$, the two possible cases, $A_{SYS} = A_{LC}$ and $A_{SYS} = A_{HC}$, will be analyzed separately to determine the load distribution that will minimize the total energy.



(a) Homogeneous Frequencies



(b) Hetegeneous Frequencies

Figure 6: Sample execution of two tasks on a dual-core platform with homogeneous and heterogeneous frequencies.

Case 1. $A_{SYS} = A_{LC}$*:* If the activity of the *Light* core is dominant, this implies $A_{LC} >= A_{HC}$. The expression for the total energy can be reduced by substituting A_{HC} from (9), as well as A_{LC} from Eq. 10 to obtain:

$$\begin{aligned} E_{TOT,DC} =& A_{LC,MIN} * \left(\frac{F_{MAX}}{F_{LC}} * (P_{\Delta,LC+SYS}) - P_{\Delta,HC} \right) \\ &+ A_{SC} * P_{\Delta,HC} \\ &+ D * (P_{S,LC} + P_{S,HC} + P_{S,SYS}) \end{aligned} \tag{11}$$

Where $P_{\Delta,LC+SYS}$ is shorthand for $P_{\Delta,LC} + P_{\Delta,SYS}$. In order to minimize this energy, only the first term needs to be considered, since the only variable is $A_{LC,MIN}$, all other are constants. The decision variable indicates how much load will be allocated to the *Light* core, which could be in the range $[0, A_{SC}]$. The total energy minimization problem can then be stated as:

$$min\left\{ A_{LC,MIN} * \underbrace{\left(\frac{F_{MAX}}{F_{LC}} (P_{\Delta,LC} + P_{\Delta,SYS}) - P_{\Delta,HC} \right)}_{const.} \right\} \tag{12}$$

In this expression, the constant term, which depends on the platform's characteristics and the *Light* core's frequency, can be either positive or negative. These cases should be analyzed separately, in order to determine when to minimize or maximize the decision variable: $A'_{LC,MIN}$.

$$A'_{LC,MIN} = \begin{cases} A_{SC} & \text{if } \frac{F_{MAX}}{F_{LC}} (P_{\Delta,LC+SYS}) < P_{\Delta,HC} \\ \frac{A_{SC}}{1 + \frac{F_{MAX}}{F_{LC}}} & \text{if } \frac{F_{MAX}}{F_{LC}} (P_{\Delta,LC+SYS}) > P_{\Delta,HC} \end{cases} \tag{13}$$

The first case is calculated with the constant less than zero. In this case, the decision variable $A'_{LC,MIN}$ has to

be maximized. In other words, the entire single-core load should be sent to the *Light* core, meaning $A'_{LC,MIN} = A_{SC}$. The second case occurs when the constant is greater than zero. Here, the decision variable has to be minimized. Since this is Case 1, $A'_{LC,MIN}$ cannot be set to zero, otherwise $max(A_{LC}, A_{HC}) = A_{HC}$, which contradicts our case. So the minimum value occurs when both cores have the same activity, or $A_{LC} = A_{HC}$. Using Eqs. 9 and 10, one obtains $A'_{LC,MIN} * (1 + \frac{F_{MAX}}{F_{LC}}) = A_{SC}$. Rearranging the terms yields the second solution in (13).

The solutions of (13) already mark a clear distinction between two policies: serialization to the *Light* and parallelization. Depending on the F_{LC} and $P_{\Delta,SYS}$, one policy can lead to greater savings than the other. If, for example, $P_{\Delta,SYS} = 0$, the allocation test would reduce to $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC}) < P_{\Delta,HC}$. While the left hand side seems to depend on F_{LC}, it is in fact a constant, because the $P_{\Delta,LC}$ is proportional F_{LC} as well. As a result, for non-zero frequencies, the inequality always holds. This means that for low system power, the energy is always minimized by serializing the load to the *Light* core.

If, on the other hand, $P_{\Delta,SYS} = P_{\Delta,HC}$, the allocation test becomes $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC}) = P_{\Delta,HC} * (1 - \frac{F_{MAX}}{F_{LC}})$. Using the usual P_{HC} values, we obtain a critical frequency of $F_{LC} = 456.5$ MHz. This means that for frequencies below 456.5MHz, the energy will be minimized by parallelizing. Since our maximum operating frequency is 100MHz, parallelization is, in effect, the only possible solution to minimize the energy.

Case 2. $A_{SYS} = A_{HC}$: If the activity of the *Heavy* core is dominant, this implies $A_{HC} >= A_{LC}$. The expression for the total energy can be reduced by substituting A_{HC} from Eq. 9, as well as A_{LC} from Eq. 10 to obtain:

$$\begin{aligned} E_{TOT,DC} =& A_{HC} * [P_{\Delta,HC} + P_{\Delta,SYS} - \frac{F_{MAX}}{F_{LC}} * P_{\Delta,LC}] \\ &+ A_{SC} * \frac{F_{MAX}}{F_{LC}} * (P_{\Delta,LC}) \\ &+ D * (P_{S,LC} + P_{S,HC} + P_{S,SYS}) \end{aligned} \quad (14)$$

In order to minimize this energy, only the first term needs to be considered, since A_{SC} is a constant (the single-core load), and the power terms are also constants. The variable A_{HC} will indicate how much load will be allocated to the *Heavy* core, which could be in the $[0, A_{SC}]$ range. As a result, the total energy minimization problem can then be stated as:

$$min\left\{ A_{HC} * \underbrace{\left(P_{\Delta,HC} + P_{\Delta,SYS} - \frac{F_{MAX}}{F_{LC}} * P_{\Delta,LC}\right)}_{const.} \right\} \quad (15)$$

Where the constant term, depending on the platform's characteristics, can be either positive or negative. These two cases will, once again, be analyzed separately to determine when to minimize or maximize our decision variable (A'_{HC}).

$$A'_{HC} = \begin{cases} A_{SC} & \text{if } P_{\Delta,HC+SYS} < \frac{F_{MAX}}{F_{LC}} * P_{\Delta,LC} \\ \frac{A_{SC}}{1+\frac{F_{LC}}{F_{MAX}}} & \text{if } P_{\Delta,HC+SYS} > \frac{F_{MAX}}{F_{LC}} * P_{\Delta,LC} \end{cases} \quad (16)$$

The first solution is calculated with the constant less than zero. Here, the value of the A_{HC} variable should be as large as possible to minimize the energy. In other words, the entire single-core load should be sent to the *Heavy* core, meaning $A'_{HC} = A_{SC}$. The second case occurs when the constant is greater than zero. Here, the decision variable should be as small as possible to minimize the energy. Since we are in Case 2 ($A_{SYS} = A_{HC}$), A'_{HC} cannot be set to zero, otherwise $max(A_{LC}, A_{HC}) = A_{LC}$, contradicting our case. So the minimum value occurs when both cores have the same activity (execution time), or, $A_{LC} = A_{HC}$. Using Eqs. 9 and 10, one obtains $A_{HC}(1 + \frac{F_{LC}}{F_{MAX}}) = A_{SC}$. Rearranging the terms yields the second solution in (16).

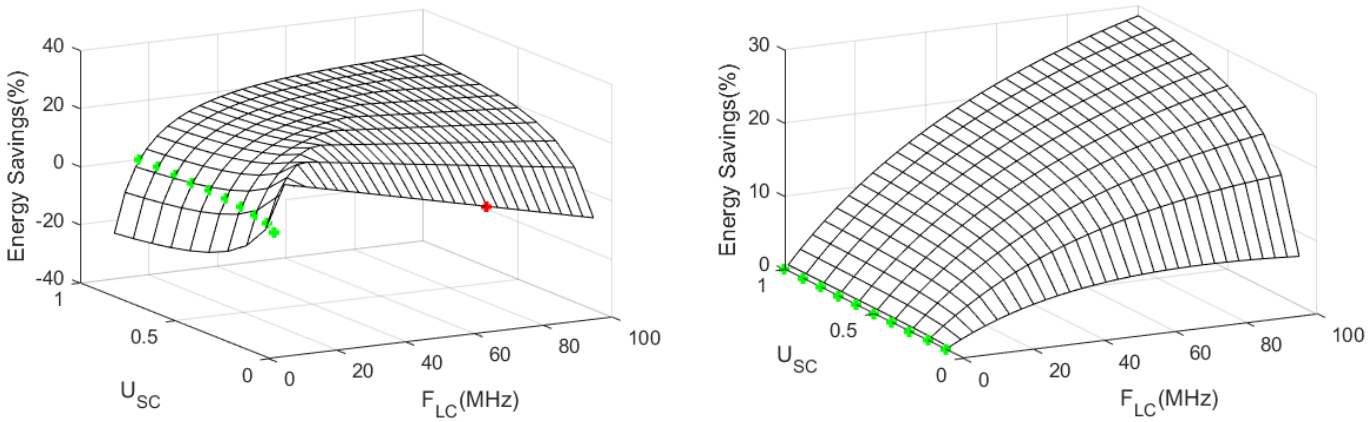
Just as in Case 1, the solutions of (16) mark a clear distinction between two policies: serialization to the *Heavy* and parallelization. Depending on the F_{LC} and $P_{\Delta,SYS}$, one policy can lead to greater savings than the other. If, for example, $P_{\Delta,SYS} = 0$, the allocation test would reduce to $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC}) > P_{\Delta,HC}$. Both sides of the inequality are constants, and with the usual power values from Table II, the inequality does not hold. This means that with low system power, serializing to the *Heavy* core will never minimize the energy. Notice that for $P_{\Delta,SYS} = 0$, the solution was $A_{LC,MIN} = A_{SC}$. Using (8), it follows that $A_{HC} = 0$. If, on the other hand, $P_{\Delta,SYS} = P_{\Delta,HC}$, the allocation test becomes $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC}) > 2 * P_{\Delta,HC}$. Once again, both sides of the inequality are constant, and using usual power values, the inequality always holds. This means that for high system power, parallelization is the only possible solution to minimize the energy. It should be noted that this solution is equal to the second one in Case 1; it can be seen that $A'_{LC,MIN}$ and A'_{HC} add up to A_{SC}, just as in (9).

C. Optimal allocation

Once a platform's power values and operating frequencies have been either calculated or estimated [1], it is fairly easy to apply these numerical tests to determine the optimal allocation policy: either serialization or parallelization. Afterwards, the software infrastructure is in charge of allocating the task in accordance to the optimal policy.

In the previous subsection, we have derived the policies that minimize the total energy consumed by the proposed dual-core platform. Depending on the core frequencies and platform's power values (see first solution to (13)), the energy optimal policy can be either serialization (to offload the entire load to the *Light* core) or parallelization (to

[1]Note that these conditions might be modified during runtime, e.g., in response to temperature variations.



(a) Serialization ($P_{SYS} = 0$)

(b) Parallelization ($P_{SYS} = P_{HC}$)

Figure 7: Energy savings for both serialization and parallelization policies as a function of the utilization and F_{LC}. Green and red points indicate the minimum and maximum values of F_{LC} for which the proposed dual-core is more energy-efficient than the single-core. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

split the load between to *Heavy* and *Light* cores). Whether the minimal energy consumption of the Heavy-Light is less than the single-core is another matter. Just as in the homogeneous frequency case, the total energy savings of each policy will depend on the utilization as well as the *Light* core's operating frequency. We will now derive the frequency range for which the dual-core platform will have energy savings, as a function of the utilization.

1) Task serialization: It is easy to see why the utilization has a big impact on the savings. At very low utilizations, the devices are mostly in sleep mode. Since the dual-core consumes more power during sleep than the single core, there will be energy losses. At the opposite end, with very high utilizations, the dual-core is able to exploit the lower power density in the *Light* core to lower the total energy spent. We are interested the following problem: given a utilization, what frequency ranges will result in energy savings for the dual-core platform?

To find these frequency values, we begin once again with the total energy equation (3). Since there are no restrictions on the frequency scaling, there will come a point for low frequencies where the execution time will exceed the period. To simplify the analysis, we will split this into two cases, since the energy reduces to different terms. The first case, when the period is exceeded, or $A_{LC} > D$, occurs when $F_{LC} < \frac{F_{MAX}}{D} * A_{LC,min}$. In this case, the total energy equation can be expressed as follows:

$$E_{DC} = A_{LC} * (P_{A,LC} + P_{S,HC}) \tag{17}$$

Note that since task serialization is being discussed, the system power is assumed to be zero, so its terms can be ignored. Since the device will always be computing, it means the *Light* core is always on and the *Heavy* core is always in sleep mode. Nonetheless, its minimum activity is equal to the single core activity, or $A_{LC,min} = A_{SC}$. The dual-core energy spent in one period is simply the computation time multiplied by the *Light* core's active power and the *Heavy* core's sleep power. When this is equated to the single-core energy, and (10) is used to reduce terms, the following relation is obtained:

$$\frac{F_{MAX}}{F_{LC}} * (P_{A,LC} + P_{S,HC}) = P_{\Delta,HC} + \frac{P_{S,HC}}{U_{SC}} \tag{18}$$

Where U_{SC} is the single core utilization, defined as $\frac{A_{SC}}{D}$. By calculating the F_{LC} values which, for a given utilization, satisfy this relation, the lower frequency range can be determined. To determine the upper frequency range, we now analyze the second case where the period is not exceeded, or $A_{LC} \leq D$. Here, the dual-core energy reduces to the following expression:

$$E_{DC} = A_{LC} * P_{\Delta,LC} + D * (P_{S,HC} + P_{S,LC}) \tag{19}$$

In this case, the computation does not overrun the period, and the device will be able to enter sleep mode. Once again, by equating these values to the single core energy, the following relation is found:

$$\frac{F_{MAX}}{F_{LC}} * P_{\Delta,LC} + \frac{P_{S,LC}}{U_{SC}} = P_{\Delta,HC} \tag{20}$$

Given a utilization, the F_{LC} values that satisfy this relation can be calculated. Figure 7a shows the energy savings of serialization as a function of the F_{LC} and U_{SC}. The red and green points show the maximum and minimum F_{LC} values, respectively, for which the single and dual-core energies are equal. Note the $U_{SC} = 0.1$ plane, which includes the only visible red point. Starting from the maximum value (100MHz), as F_{LC} starts decreasing, the computation will

take longer. An extended execution time means a smaller amount of sleep time, which is more expensive in the dual-core case, hence the increasing savings. At the red dot, $F_{LC} = 68.97$MHz, the single and dual-core energy are equal, below that, the dual-core saves energy. There is, however, an inflection point and then a minimum frequency, the green point at $F_{LC} = 6.94$MHz, after which, there will only be energy losses. This happens because very low frequencies will extend the computation time beyond the period. In a single-core platform, this would normally be irrelevant from the energy-only perspective. The problem with the dual-core is that this extended execution time leads to an even greater accumulation of sleep energy from the *Heavy* core. So even when ignoring performance constraints, there is a minimum frequency of operation for energy savings. These limits are fairly low, in the $[6 - -17]$MHz range for $U_{SC} \in [0.1, 1]$, when compared to the frequencies needed to avoid a deadline miss: [10–100]MHz for $U_{SC} \in [0.1, 1]$. It should be noted that the other red dots are not visible for higher utilizations because they go to the GHz range, so the limiting factor is the maximum operating frequency.

2) Task parallelization: From the previous subsection it can be seen that in cases of sufficiently high F_{LC} and $P_{\Delta,SYS}$, the optimal solution is to parallelize the load such that the activities of the *Light* and *Heavy* cores are equal. In the homogeneous frequency scenario, this was achieved by splitting the single-core load equally among both cores. In the heterogeneous case, however, the load given to the *Light* core must be scaled down such that its execution time will be equal to that of the *Heavy* core. In order to find this proportionality, one begins with the activity equality: $A_{LC} = A_{HC}$. By substituting (9), and the definition of activity, one obtains:

$$\left(\frac{F_{MAX}}{F_{LC}}\right) * \left(\frac{I_{LC} * cpi}{F_{MAX}}\right) = \left(\frac{I_{HC} * cpi}{F_{MAX}}\right) \tag{21}$$

On the left-hand side, it can be seen that the activity of the *Light* core is scaled by the frequency ratio to equal the activity in the *Heavy* core. This, however, does not indicate how a load should be distributed in order to achieve this. To this end, we use the load equivalence, $I_{SC} = I_{HC} + I_{LC}$, and combine it with (21) to obtain the following two expressions:

$$I_{HC} = \frac{I_{SC}}{1 + \frac{F_{LC}}{F_{MAX}}} \tag{22}$$

$$I_{LC} = \left(\frac{F_{LC}}{F_{MAX}}\right) * \frac{I_{SC}}{1 + \frac{F_{LC}}{F_{MAX}}} \tag{23}$$

These expressions indicate how the load should be distributed among both cores to have the maximum, or ideal, parallelism under heterogeneous frequencies. It can be seen that, as F_{LC} decreases, its load I_{LC} decreases, and I_{HC} increases, thus maintaining the activity between cores equal. This result is similar to the homogeneous case, except that the load will be more skewed towards the *Heavy* core, depending on the frequency ratio.

In general, the previously proposed task allocation policies, *LRU* and *First Fit*, are valid for the heterogeneous case. The *LRU* will be used when there is no task-set knowledge, and for large numbers of tasks, the average utilization in both cores will be even. If, however, there are few and uneven tasks, there is a risk of allocating a larger task to the *Light* core. As the *Light* frequency is reduced, this results in an extended execution time, which leads to more accumulation of system energy and lastly, energy losses. The *First Fit* policy can exploit task-set knowledge and attempt to allocate the tasks closer to the ideal partitions of instructions. The only difference between the homogeneous and the heterogeneous case is that in the latter, *First Fit* maximizes the system sleep time and the *Heavy* core's utilization. Skewing the load to the *Light* core would have meant longer execution times, so the minimum execution time of *Heavy* core consumes less energy.

Just as with task serialization, we now turn our attention to determining the ranges of frequencies for which the dual-core platform will have energy savings, as a function of the single-core utility. Once again, we begin from the dual-core energy equation, (2). Since this is the task parallelization case, we assume high system power. Similarly, it has been shown that the optimal policy is to parallelize such that the *Light* and *Heavy* cores' activities are equal. As a result, the dual-core energy can be expressed as:

$$E_{TOT,DC} = \frac{A_{SC}}{1 + \frac{F_{LC}}{F_{MAX}}} * \sum_i P_{\Delta,i} + D * \sum_i P_{S,i} \tag{24}$$

Where the summations account for the power of all three components: the *Heavy*, *Light* cores and the system peripherals. To find the critical F_{LC} values, this energy is equated to the single-core energy, resulting in the following relation:

$$\frac{\sum_i P_{\Delta,i}}{1 + \frac{F_{LC}}{F_{MAX}}} + \frac{P_{S,LC}}{U_{SC}} = P_{\Delta,HC} + P_{\Delta,SYS} \tag{25}$$

For any given utilization, this relation only holds with $F_{LC} = 0$. This means that for any non-zero frequency and utilization, using the *Light* core will result in energy savings. These results can be seen in Figure 7b, where the green points form a line at $F_{LC} = 0$. For all other values, the ideal parallelization saves energy. For a given F_{LC}, the savings increase with the utilization because the ratio of more energy-efficient computation increases.

D. Summary

In this section, we have derived the optimal allocation policies that minimize the energy consumption of our proposed dual-core platform. The criteria for selecting the optimal policy includes the ratios of the cores' active and sleep powers, and the system's peripheral power. As explained in Sec. V, for platforms with low system power, the optimal policy is to serialize to the *Light* core. Offloading

a single-core's sequential load to the dual-core's *Light* core requires no special software considerations. On the other hand, if the system power goes beyond a certain threshold ($P_{SYS,THR}$), the energy-optimal policy becomes parallelization. This requires special considerations, particularly because our low-cost, performance constrained target platforms are usually limited to task level parallelism. Depending on the load, whether it can be profiled or not, we have proposed different task allocation algorithms. While the ideal load balancing derived in this section maximizes the savings, the total energy savings in a real-world application will vary, depending on the application type, and the available degree of parallelism, as will be shown in Section VII.

The flowchart in Figure 8 summarizes the process to determine the most appropriate policy for our dual-core platform. After the chip has been manufactured, the values for P_{SYS} and F_{LC} can be either measured or estimated. If the application can be analyzed and its utilization bounded, it can be used later to find the optimal frequency of operation. Using these values, our runtime can then determine if serialization or parallelization will offer the greatest savings. When $P_{SYS} > P_{SYY,THR}$ the optimal policy will be parallelization. But if the degree of parallelism it too low, or the reduced F_{LC} is below the frequency threshold (F_{THR}) necessary to guarantee minimum application requirements, then the only solution is to serialize to the *Heavy* core, at the expense of energy savings. Note that our focus is only energy-efficiency, and not performance guarantees. If there are no performance requirements, parallelization will introduce savings, although with a higher latency. When P_{SYS} is low, the minimal energy is obtained through serialization to the *Light* core. F_{LC} must first be checked if it is within the energy saving range, and whether it meets any performance requirements. If F_{LC} meets the criteria, our runtime performs serialization.

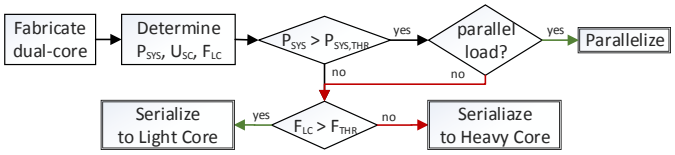


Figure 8: Flowchart summarizing methodology for determining optimal task allocation scheme.

VII. 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 common MCU-based applications to our platform, and show the energy savings from real-world applications.

A. Simulation infrastructure

Our simulator, a modified version of [24], 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 Figure 1. This simulator's power model includes not only the cores, but the TCDM and Flash memories as well. As was described in Section V-D, precedence constraints are supported through semaphores.

B. Homogeneous frequency simulation results

For our initial experiments, we will focus on the homogeneous frequency case, in which both the *Heavy* and the *Light* core can be clocked at the maximum frequency. We begin with 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 under two different scenarios: task serialization and task parallelization. We present the results as the energy savings with respect to the single-core, as defined in (7).

Due to the effects of task variance discussed in Section V, we tested different levels of parallelism by introducing task length variation $\Delta = \{0, 50\}\%$, while maintaining the utilization constant. When $\Delta = 0\%$, it means both tasks are of the same length. This will be used a reference of maximum theoretical parallelism, since the load can be perfectly split among the two cores. When $\Delta = 50\%$, it means tasks are $\pm 50\%$ from the mean. For example, if $T1 = T2 = 10ms$ with $\Delta = 0$, then with $\Delta = 50\%$ the tasks become $T1 = 5ms$, $T2 = 15ms$.

Task parallelization: As was seen in Section V, whenever the system power is *over* the threshold value $P_{\Delta,SYS} \geq (P_{\Delta,HC} - P_{\Delta,LC})$, the most energy-efficient allocation policy is to parallelize such that both cores have the same activity, or computation time. In the homogeneous case, this is achieved by distributing the load as evenly as possible among both cores. For this experiment, we used the core and system power numbers as described in Table II. Figure 9 shows the energy savings as a function of the single-core utilization. This utilization is calculated from a task-set running in the single-core. The energy consumed by the single-core is then compared to the energy consumed by the dual-core executing the same task-set. The single-core utilization was chosen as an independent variable to reference the effect the load has on total energy consumption, and compare the efficiency of the proposed dual-core platform. We have implemented two different task allocation policies: *LRU* and *First Fit*, as discussed in Section V. The blue lines show the energy savings for

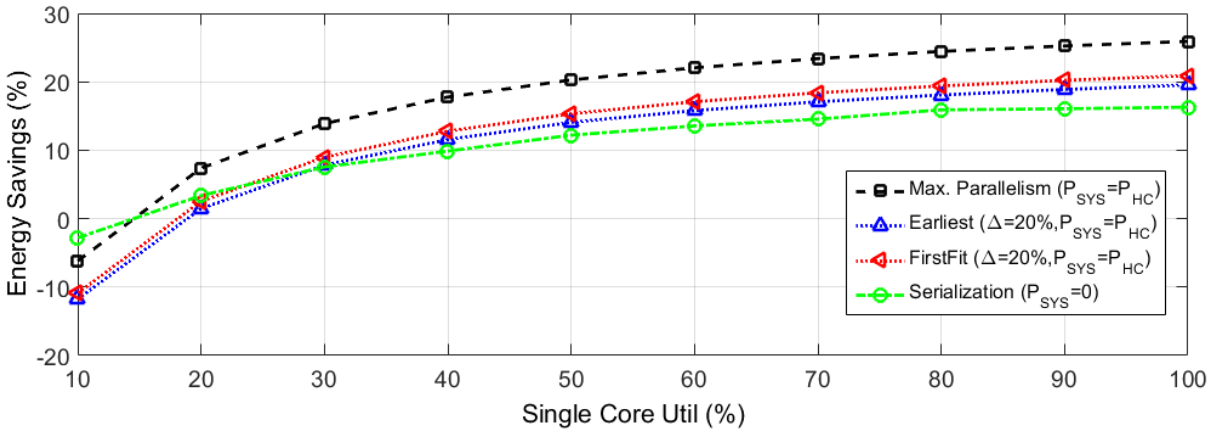


Figure 9: Simulation results under homogeneous frequencies ($F_{HC} = F_{LC} = 100M$)show the energy savings for several allocation policies. Three parallel policies are shown, as well as the serialization policy. Note that serialization is for $P_{SYS} = 0$, which leads to higher savings compared to parallelization for $P_{SYS} = P_{HC}$ at low utilizations. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

the *LRU* policy. When $\Delta = 50\%$, one task is substantially longer than the other, meaning that the load will not be evenly distributed. As expected, the energy savings are substantially lower than ideal, maximum parallelization (in black). The red lines show the energy savings for the *First Fit* policy. As mentioned in the previous section, this policy uses task-set knowledge to both minimize the active system energy and maximize the *Light* core's utilization. When there is high variance, it can lead to additional savings of 7% with respect to *LRU*.

Task serialization: As was seen in Section V, if the system power is *under* the threshold value $P_{\Delta,SYS} < (P_{\Delta,HC} - P_{\Delta,LC})$, the most energy-efficient allocation policy is to allocate all tasks to the *Light* core. For this experiment, system power set to zero, the other power values were kept the same. The green line in Figure 9 shows the energy savings of serialization as a function of the single-core utilization. It can be seen that for $U_{SC} \geq 15\%$, the dual-core platform has energy savings, reaching up to 15% for $U_{SC} = 1$.

C. Heterogeneous frequency simulation results

In this section, we evaluate the allocation algorithms proposed in Section VI, under the assumption that the *Light* core cannot be clocked to the maximum frequency. The result is a heterogeneous system with a performance penalty when compared to our previous homogeneous case. Depending on the granularity of the frequency scaling, the maximum safe operating frequency might not be reached. As a result, we use pessimistic values of $F_{HC} = 100$MHz, $F_{LC} = 50$MHz to quantify our savings, noting that in more realistic scenarios, the actual savings will be somewhere between our optimistic homogeneous frequency case, and our pessimistic heterogeneous frequency case. For our experiments, we once again begin with synthetic tasks to sweep the entire utilization range. For a given period, we instantiate a queue of $I = 100$ iterations, each activating $N = 2$ independent tasks. Their execution times were chosen to reach the desired utilization under different Δ values, using the same methodology as in the previous subsection. Each of these queues was then executed in both a single-core and a dual-core platform under two different scenarios: task serialization and task parallelization. We present the results as the energy savings with respect to the single-core, as defined in (7).

Task parallelization: As was seen in Section VI-C2, whenever P_{SYS} and F_{LC} hold the inequality $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC+SYS}) > P_{\Delta,HC}$, the most energy-efficient allocation policy is to parallelize the load in proportion to the frequency ratio. Figure 10 shows the energy savings as a function of the single-core utilization. Just as in the homogeneous case, this allows us to compare and evaluate the efficiency of the dual-core vs single-core platform. We have implemented two different task allocation policies: *LRU* and *First Fit*, as discussed in Section VI-C2. 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 lower savings. The red and blue lines show the energy savings for the *First Fit* and *LRU* policies, respectively. Because the reduced *Light* frequency has a very significant performance penalty, these results are meant to show the bounds on the energy consumption. In the case of *First Fit*, task-set knowledge guarantees that for the maximum system sleep time, the *Light* core's activity is also minimized. With *LRU*, there is no guarantee, and in the worst-case scenario, the *Light* core's activity will be maximized. As these results show, the possible energy savings with *First Fit*, up to 10% with $U_{SC} = 1$, can turn into energy losses with *LRU*, which loses at least 8% in the best-case scenario.

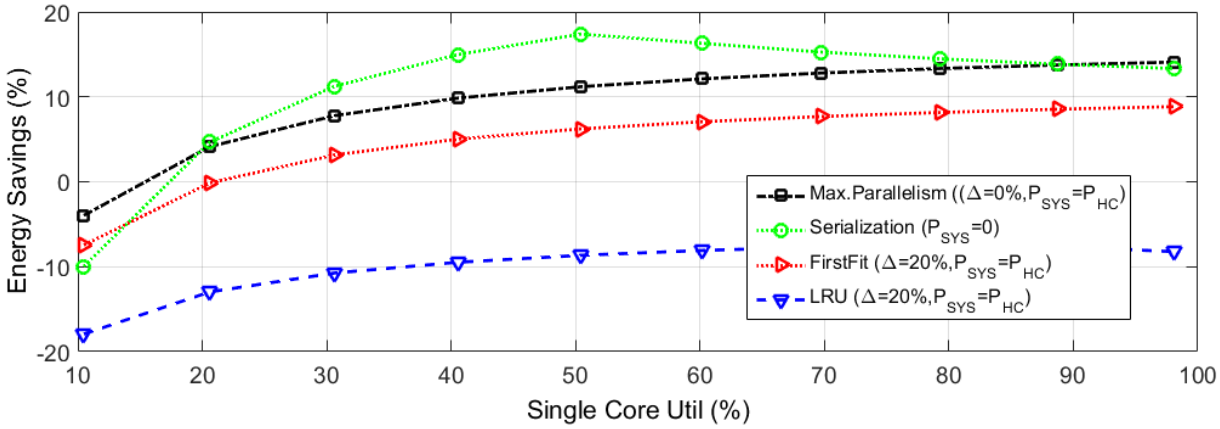


Figure 10: Simulation results under heterogeneous frequencies ($F_{HC} = 100M, F_{LC} = 50M$) show the energy savings for several allocation policies. The two proposed parallel policies are shown, as well as the serialization policy. Note that serialization is for $P_{SYS} = 0$, which leads to higher savings compared to parallelism for $P_{SYS} = P_{HC}$ in a medium utilization range. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

Task serialization: As was seen in Section VI-C1, whenever P_{SYS} and F_{LC} hold the inequality $\frac{F_{MAX}}{F_{LC}}(P_{\Delta,LC+SYS}) < P_{\Delta,HC}$, the most energy-efficient allocation policy is to allocate all tasks to the *Light* core. For this experiment, the system power was set to zero. The green line in Figure 10 shows the energy savings of serialization as a function of the single-core utilization. It can be seen that for $U_{SC} \geq\sim 15\%$, there are energy savings. However, after $U_{SC} = 50$, the energy savings start to decrease. This happens because of the reduced frequency, which extends computation time beyond the period. This accumulates extra sleep energy from the *Heavy* core, and decreases the savings. It should also be noted that even though serialization can achieve energy savings for most of the utilization range, the dual-core latency will always be larger than the single core. This does not necessarily mean a deadline miss. If the application has a utilization less than 50%, this additional latency can be tolerated and serialization can still provide energy savings. Since the main focus of our work is energy efficiency, when the reduced frequency is below a critical performance value chosen by the developer, $F_{LC} < F_{CRIT}$, our policy is to serialize to the *Heavy* core. While this would sacrifice the energy efficiency, the performance will be guaranteed to be equal to the single-core case in all circumstances.

D. Real-world case studies

The results shown so far have used synthetic tasks. While these results give important insight to the energy saving trends, it is important to demonstrate energy savings in real-world applications. To this end, we have evaluated the energy savings of different microcontroller applications running on a Heavy-Light platform under both optimistic and pessimistic conditions. Our first application is an open-source Inertial Navigation System (INS) project [25]. This application implements zero-velocity-update INS which determines the position of an object in a space from the input of inertial sensors (i.e. accelerometers). There is a small degree of task-level parallelism present in the filtering, detecting, and correcting stages of the position and velocity calculations. This application serves as a reference for low parallelism. Our second application is an MP3 decoder which, as detailed in [26], can have a substantial degree of parallelism in certain sub-tasks. Finally, our second case study is the Pix4Flow camera module [27], which is part of the broader PixHawk project [28]. This module has a Cortex M4, which executes a lightweight computer-vision algorithm called optical flow, 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. For these experiments, we used the core and system power numbers as described in Table II, where parallelization is optimal. Furthermore, we tested these applications under both homogeneous and heterogeneous frequency configurations. In the former, $F_{HC} = F_{HC} = 100\,\text{MHz}$, while in the latter $F_{HC} = 100\,\text{MHz}$ and $F_{LC} = 50\,\text{MHz}$. The results of these experiments can be seen in Figure 11. It is important to notice that as the degree of parallelism increases, so do the energy savings. As expected, the homogeneous frequency case produces the best results, since there is no performance penalty during parallelization. The energy savings range from around 5% in the INS application to almost 20% in the Pix4Flow application in the homogeneous case. For the heterogeneous frequency case, even with a pessimistic F_{LC} value, the savings are in the [2,11]% range.

Framework overhead: As with any task allocation framework, there are overheads associated to an actual implementation. We define overhead to be the amount of active time spent doing anything other than executing tasks. More specifically, the time spent registering task arrivals, checking for task dependencies, determining the optimal policy, and communication mechanisms between the master core and the energy booster. In our simulation platform, we can calculate this by subtracting the task execution times from the measured active time. From all our experiments, these are in the tens of microseconds per iteration, depending on the number of tasks. In the INS benchmark, for example, 3 dependent tasks had an overhead of around 50 μs.

VIII. Conclusions

In this work, we have analyzed the energy efficiency of next generation microcontroller platforms composed of one *Heavy* core fabricated with *worst-case* design margins, and a more energy-efficient *Light* co-processor with reduced design margin. We have presented a lightweight task allocation framework that exploits the available power heterogeneity to minimize the platform's total energy consumption according to the power ratios of a platform's components, namely the system peripherals and the *Light* core's operating frequency. We have derived in detail the optimal load balancing policies, which depend on the *Light* core's operating frequency, and validated our task allocation policies with an instruction-accurate simulation platform. Real-world case studies show that compared to a single-core platform, our policies can save up to 20% of total energy with a maximum *Light* core operating frequency, and almost 11% with a reduced frequency. Post-silicon measurements confirm the robustness of the proposed approach, since the probability of a *Light* core operating at a maximum frequency is over 99% for most common environmental conditions.

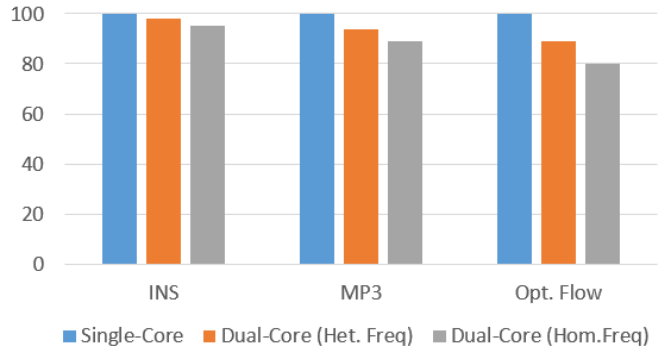


Figure 11: Total energy savings with shadow co-processor in homogeneous and heterogeneous configurations.

IX. Acknowledgments

This work has been funded by NXP Semiconductors, FP7 project PHIDIAS (g.a.318013), SNF project "Transient Computing Systems" (200021_157048), and ETH Zürich Grant funding.

References

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

[2] Y. Pu, J. Echeverri, M. Meijer, and J. de Gyvez, "Logic synthesis of low-power ics with ultra-wide voltage and frequency scaling," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–2.

[3] *LPC5410X Product Data Sheet*, NXP Semiconductors, 2015, rev 2.2.

[4] A. Gomez *et al.*, "Dynamic energy burst scaling for transiently powered systems," in *Proc. DATE Conf.* EDA Consortium, 2016, pp. 349–354.

[5] M. Thielen *et al.*, "Human body heat for powering wearable devices: From thermal energy to application," *Energy Conversion and Management*, vol. 131, pp. 44–54, 2017.

[6] K. Jeong, A. B. Kahng, and K. Samadi, "Quantified Impacts of Guardband Reduction on Design Process Outcomes," in *9th International Symposium on Quality Electronic Design (isqed 2008)*. IEEE, 2008, pp. 790–797.

[7] A. B. Kahng, R. Kumar, and J. Sartori, "Recovery-Driven Design: Exploiting Error Resilience in Design of Energy-Efficient Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 404–417, 2012.

[8] A. Gomez, C. Pinto, A. Bartolini *et al.*, "Reducing energy consumption in microcontroller-based platforms with low design margin co-processors," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015.

[9] R. Kumar *et al.*, "Processor power reduction via single-ISA heterogeneous multi-core architectures," *Computer Architecture Letters*, vol. 2, no. 1, pp. 2 – 2, 2003.

[10] A. a. Eltawil, M. Engel, B. Geuskens *et al.*, "A survey of cross-layer power-reliability tradeoffs in multi and many core systems-on-chip," *Microprocess. Microsyst.*, 2013.

[11] U. R. Karpuzcu, A. Sinkar *et al.*, "EnergySmart: Toward Energy-efficient Manycores for Near-Threshold Computing," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 542–553.

[12] F. Paterna, A. Acquaviva, A. Caprara *et al.*, "Variability-aware task allocation for energy-efficient quality of service provisioning in embedded streaming multimedia applications," *Computers, IEEE Transactions on*, vol. 61, no. 7, pp. 939–953, July 2012.

[13] A. Rahimi, A. Marongiu, P. Burgio *et al.*, "Variation-tolerant openmp tasking on tightly-coupled processor clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 541–546.

[14] D. Bortolotti, A. Bartolini, and L. Benini, "An ultra-low power resilient multi-core architecture with static and dynamic tolerance to ambient temperature-induced variability," *Microprocess. Microsyst.*, vol. 38, no. 8, Part A, pp. 776 – 787, 2014.

[15] F. Chaix *et al.*, "Variability-aware Task Mapping Strategies for Many-Cores Processor Chips," in *2011 IEEE 17th International On-Line Testing Symposium*. IEEE, 2011, pp. 55–60.

[16] A. Gomez *et al.*, "Sf3p: A framework to explore and prototype hierarchical compositions of real-time schedulers," in *Proc. RSP Symp.* IEEE, 2014, pp. 2–8.

[17] J. Chen and L. John, "Efficient program scheduling for heterogeneous multi-core processors," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009.

[18] D. He and W. Mueller, "A heuristic energy-aware approach for hard real-time systems on multi-core platforms," *Microprocess. Microsyst.*, 2013.

[19] J. Mei *et al.*, "Energy-aware preemptive scheduling algorithm for sporadic tasks on dvs platform," *Microprocess. Microsyst.*, vol. 37, no. 1, pp. 99–112, Feb. 2013.

[20] A. Gupta, S. Im *et al.*, "Scheduling Heterogeneous Processors Isn'T As Easy As You Think," in *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '12. SIAM, 2012, pp. 1242–1253.

[21] T. S. Muthukaruppan *et al.*, "Hierarchical power management for asymmetric multi-core in dark silicon era," *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, 2013.

[22] A. Gomez *et al.*, "Wearable, energy-opportunistic vision sensing for walking speed estimation," in *Proc. SAS Symp.* IEEE, 2017, pp. 1–6.

[23] R. Marau *et al.*, "Performing flexible control on low-cost microcontrollers using a minimal real-time kernel," *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 2, pp. 125–133, 2008.

[24] D. Bortolotti, C. Pinto, A. Marongiu *et al.*, "VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013.

[25] J.-O. Nilsson, I. Skog, P. Handel, and K. Hari, "Foot-mounted INS for everybody - an open-source embedded implementation," in *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*. Ieee, 2012, pp. 140–145.

[26] W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A Practical Approach to Exploiting Coarse-Grained Pipeline Parallelism in C Programs," in *IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007.

[27] Pix4FLow. (2016) Pix4flow smart camera module website. [Online]. Available: http://pixhawk.org/modules/px4flow

[28] L. Meier *et al.*, "Pixhawk: A system for autonomous flight using onboard computer vision," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2992–2997.