# Energy-Aware Standby-Sparing on Heterogeneous Multicore Systems

Abhishek Roy, Hakan Aydin
Department of Computer Science
George Mason University
Fairfax, Virginia 22030
aroy6@gmu.edu, aydin@cs.gmu.edu

Dakai Zhu
Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas 78249
dakai.zhu@utsa.edu

## ABSTRACT

Standby-sparing systems where one processor is used as primary while another one is deployed as spare have been used to provide high reliability to real-time embedded systems. To reduce the energy consumption, the primary uses DVFS while the spare employs DPM to postpone the backup tasks. In this paper, we re-visit the problem for heterogeneous multicore systems that include both high-performance and low-power cores. We identify and address the two main dimensions of the problem, namely, what type of core to use as the primary or backup, and how to make frequency assignments on the primary to maximize energy savings.

## 1. INTRODUCTION

Reliability is an increasingly important design dimension for real-time embedded systems including those deployed for safety-critical applications in the areas of industrial control, automotive, and high-confidence medical systems. In such systems, provisions must be made to tolerate both *transient* and *permanent* faults. Transient faults are primarily caused by electromagnetic interference and cosmic rays (including alpha particles) [10]. They manifest in the form of *soft errors* (or *single event upsets (SEUs)*) that appear in the processor core logic or memory subsystems, leading to incorrect computations. Moreover, with increased technology scaling and the use of aggressive low-power design techniques such as near-threshold voltage operation, the CMOS circuits become more vulnerable to transient faults [7, 15]. By their nature, the transient faults are short-lived, and it is often possible to invoke an alternative back-up task after such a fault is detected on a specific task [2]. The permanent faults, on the other hand, are often the results of manufacturing defects, aging, or adverse temperature/environmental conditions. It is often necessary to have redundant hardware, in the form of extra processing units, in order to tolerate the permanent faults [10].

A popular approach that allows to tolerate both transient and permanent faults is the *standby-sparing* system [4]. In a standby-sparing system, one processor (called *primary*) executes the main tasks, while a back-up copy for each task is allocated to a second *(spare)* processor. The spare processor is kept in low-power state thanks to the Dynamic Power Management (DPM) technique, unless a fault is detected on the primary processor. In case of a permanent fault, the spare completely takes over the execution of the workload. For transient faults, the spare (re-)executes only the back-up copies of the affected task(s).

In *energy-aware* standby-sparing configuration, the primary processor uses Dynamic Voltage and Frequency Scaling (DVFS) to execute the main tasks at reduced processing frequencies to save power. The spare processor uses only DPM; it delays the back-ups as much as possible in order to cancel them dynamically to save power, if the corresponding main task(s) complete without errors. Thus, a common problem addressed in the literature is to determine the voltage/frequency assignments for main tasks allocated to the primary processor for various task models [4, 5, 8, 9]. A frequent theme in this line of research is to minimize the overlap between the primary and back-up copies of the tasks to reduce the energy consumption of the spare, because it is not always possible to delay the back-ups by significant margins, in view of the task deadlines [5, 9, 13].

The existing energy-aware standby-sparing studies focus exclusively on homogeneous dual-core systems [4, 5, 9]. Recently, *heterogeneous (asymmetric)* multicore systems have been attracting much attention due to their ability to exploit power and performance trade-offs for varying workloads [11, 12]. In those systems, some "big" cores offer high-performance but exhibit high-power consumption profiles. The power-efficient ("little") cores, on the other hand, have modest performance characteristics. A well-known example is ARM big.LITTLE systems that combine high performance out-of-order (e.g., ARM Cortex-A15) cores with power-efficient in-order (e.g., ARM Cortex-A7) cores [1]. In such systems, the "big" cores are activated when high-performance is needed, while "little" cores can be sufficient for modest load levels in order to save energy.

In this paper, we re-visit the energy-aware standby-sparing problem in the context of emerging heterogeneous multi-core systems for real-time embedded applications. In addition to the problem of determining the frequency assignments on the primary, the new settings introduce a new dimension, namely, whether to use the power-efficient core as the primary or spare. We investigate both of these dimensions, propose solutions, and present a comprehensive experimental evaluation. Our results indicate that, unlike the homoge-



*DAC '17, June 18-22, 2017, Austin, TX, USA*



DOI: http://dx.doi.org/10.1145/3061639.3062238

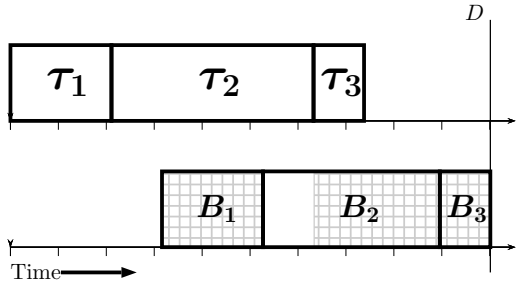



Figure 1: A standby-sparing system

neous case, allowing some overlaps between the primary and back-up copies *may* help to reduce energy, in particular when the high-performance core is used as the primary. We also show that, in general, designating the power-efficient core as the primary and high-power core as the spare is preferable, except when the system utilization is high. Our analysis is performed under the realistic assumption that the execution time and power consumption of different applications scale by different ratios on the big and little cores [12].

The rest of the paper is organized as follows. In Section 2, we present our system model and assumptions. In Section 3, we develop our proposed schemes for primary/spare role designation, and the task frequency assignment. We experimentally evaluate the proposed schemes in Section 4 before concluding in Section 5.

# 2. SYSTEM MODEL AND ASSUMPTIONS

## 2.1 Platform and Application model

We consider a single-ISA heterogeneous dual-core system that consists of a high-performance and power-hungry ("big") core, and a relatively slow but power-efficient ("little") core. The application consists of a set of $n$ real-time periodic tasks $\{\tau_1, \ldots, \tau_n\}$ that must complete their execution by a specific deadline $D$, which is also equal to the common period. This specific model is also called *frame-based* model in the literature [14, 15]. The worst-case number of cycles required by task $\tau_i$ is denoted by $C_i$. When executed at the frequency $f$, the task may require up to $W_i = C_i/f$ time units to complete. Due to the architectural differences (such as in-order versus out-of-order processors), the same task may require different number of cycles and hence, different execution times, on each type of cores. Throughout the paper, we will use the superscripts $LP$ and $HP$, respectively, to denote the values of these variables for the low-power and high-power cores (e.g., $C_i^{LP}$, $W_i^{LP}$, $C_i^{HP}$, $W_i^{HP}$). The maximum processing frequency on low-power and high-power cores is denoted by $f_{max}^{HP}$ and $f_{max}^{LP}$, respectively.

We employ the energy-aware standby sparing technique on the dual-processor platform [5,9]. One processing core is designated as the *primary* and the other one as the *spare*, as shown in Figure 1. Throughout the paper, we show the primary processor at the top, and the spare processor at the bottom in the figures. When a task ($\tau_i$) is allocated to the primary core, a back-up copy ($B_i$) is also allocated to the spare core. The main and backup tasks follow the same execution order on their respective processors. The primary core uses the DVFS [3] technique to save energy. The spare core is DPM-enabled; it remains in low-power state as much as possible, delaying the execution of the back-up tasks while still meeting the deadline (Figure 1). Whenever a main task completes, the *acceptance* (or, *sanity*) tests are performed [10] to check the existence of errors induced by transient faults. If there are no errors, the (remaining part of the) corresponding back-up task on the spare processor is cancelled dynamically. For example, in Figure 1, $B_1$ and $B_3$ are completely cancelled, because the corresponding main tasks ($\tau_1$ and $\tau_3$) complete successfully before the scheduled start times of $B_1$ and $B_3$. Similarly, a part of $B_2$ is cancelled as soon as $\tau_2$ completes successfully (we are using the dashed lines to indicate cancelled executions in the schedules). In case of a transient fault, the back-up task executes as specified in the spare schedule. If a permanent fault affects the primary processor, the spare copy executes the back-up tasks after that point. The back-up tasks are executed at the maximum processing frequency of the spare core. This energy-aware standby-sparing arrangement can tolerate a single fault of either processor, while enabling the recovery of transient faults affecting any subset of the main tasks on the primary processor [5].

## 2.2 Power Model

We model the dynamic power consumption of a task $\tau_i$ as $P_i(f) = a_i f^3 + \alpha_i$ where $a_i$ denotes the switching capacitance, $\alpha_i$ indicates the *frequency-independent* power consumption [9, 14], and $f$ is the processing frequency of the task. Due to the asymmetric nature of the dual-core system, the tasks exhibit different power consumption characteristics on different cores. Again, we are using the superscripts $HP$ and $LP$, to distinguish between the values of the task power consumption parameters on the high-power and power-efficient cores, respectively (e.g., $\alpha_i^{LP}$, $P_i^{HP}$).

Each core executes tasks in the *active* state, dissipating power as determined by the characteristics of the current task and processing frequency. When a core does not execute tasks, it remains in the low-power *(idle)* state. The low-power (idle) power consumption of the high-performance and low-power cores are denoted by $P_{idle}^{HP}$ and $P_{idle}^{LP}$, respectively. We assume those figures include the *static* power consumption of the corresponding core as well. Finally, the energy consumption during a time interval is given by the aggregate power consumption during the same interval.

The existence of the frequency-independent power component implies the existence of an *energy-efficient frequency ($f_{ee}$)* below which DVFS affects the overall energy consumption negatively [15]. The value of $f_{ee}$ can be analytically derived using standard techniques [15].

*Problem Statement.* In this paper, we address the following problem: Given a set of frame-based real-time tasks and a heterogeneous dual-core system, minimize the overall energy consumption while still meeting the deadlines, by determining:

1. which core should be designated as the *primary* and *spare* in the standby-sparing system, and,

2. what processing frequency assignments should be made to tasks on the *primary* core.

We investigate these two dimensions in Sections 3.1 and 3.2, respectively.

# 3. PROPOSED SCHEMES

## 3.1 Primary/Backup Role Assignment

The first design dimension is to whether to designate as the primary processor the high-performance core or low-power core. The two options are, therefore:

- **FasterP**: Use $\Pi^{HP}$ as primary, $\Pi^{LP}$ as spare
- **SlowerP**: Use $\Pi^{LP}$ as primary, $\Pi^{HP}$ as spare

It turns out that this decision can have a significant impact on the overall energy consumption of the system, depending on the workload. For instance, if a high-performance core is used as the primary core, ideally it should be slowed down through DVFS to reduce its energy consumption. However, that would extend the completion times on the primary, and the overlapped backup executions could significantly increase the energy consumption on the slow, power-efficient, spare processor. Conversely, if the low-power core is used as the primary, due to its modest performance, the task completion times would naturally shift, increasing the overlaps with the power-hungry spare processor.

Consider two tasks that run on a heterogeneous system where the normalized maximum frequencies are $f_{max}^{HP} = 1.0$ and $f_{max}^{LP} = 0.8$. Assume $P_{idle}^{HP} = 0.05$ and $P_{idle}^{LP} = 0.02$. For both tasks, $a_i^{HP} = 1.0$, $a_i^{LP} = 0.3$, $\alpha_i^{HP} = 0.1$ and $\alpha_i^{LP} = 0.03$. Table 1 gives task execution times (in $ms$), and energy consumptions ($E^{HP}$, $E^{LP}$) on both cores (in $mJ$) under respective maximum frequencies.

Table 1: Example Task Set 1

| | $W_i^{HP}$ | $W_i^{LP}$ | $E^{HP}$ | $E^{LP}$ |
|---|---|---|---|---|
| $\tau_1$ | 22 | 52 | 24.2 | 9.55 |
| $\tau_2$ | 10 | 24 | 11.0 | 4.4 |

When this task set is assigned to a homogeneous system consisting of two high-power cores ($f_{max} = 1.0$ for both cores) where the primary is slowed down as much as possible, the execution of the system is shown in Figure 2a. The last 21 time units of $B_1$'s execution are dynamically cancelled when $\tau_1$ executes successfully, but $B_2$ is executed fully. This yields an overall energy consumption of 29.5 mJ (in all the examples, we focus on the energy consumed in the fault-free execution sequence, because faults are rare events). But once the application is moved to the heterogeneous system, we can take advantage of the power-efficient execution on the little core. In the FasterP case (Figure 2b), the overlapped portion is larger than the one in Figure 2a, but the overall energy consumption decreases to 26.51 mJ, giving an improvement of 10.32%. This is due to the fact that the power-efficient core is used at its maximum frequency to execute the spare. When a SlowerP configuration is used (Figure 2c), we have the same amount of overlap as in the homogeneous case, but now the primary workload can be executed on a power-efficient core with maximum slow-down ($f_i = 0.61$). The energy consumption of the SlowerP system shows a 11.77% improvement compared to the homogeneous system. Now, if we slightly change the task set parameters and increase the $W_2^{HP}$ and $W_2^{LP}$ values to 13 and 31, respectively, we can compute that the execution on FasterP in Figure 2d yields a 10% energy savings compared to the homogeneous system, and SlowerP yields only a 7.6% improvement (not shown). In this case, the increased workload on a SlowerP configuration causes the high-power backup to start early and execute at its maximum frequency, therefore the FasterP configuration is more advantageous to execute backups on the low-power spare during the unavoidable overlap. This example shows that *the best primary/spare configuration to minimize the overall energy is dependent on the characteristics of the workload at hand.*

## 3.2 Frequency Assignment on Primary

Once all the tasks are allocated to the primary processor and their respective copies to the spare processor, the next dimension is to determine the processing frequencies for the main tasks on the primary. Normally, one would want to exploit DVFS to slow down execution on the primary and save energy. But, slowing down the main copy of the task has the potential of increasing its completion time, and thereby increasing the overlap between the main task and its backup copy, resulting in a higher overall energy consumption. Moreover, through a cascading effect, such a decision could also shift the completion times of the following main tasks and further increase the energy consumption.

We will use a heterogeneous dual-core system with $f_{max}^{HP} = 1.0$, $f_{max}^{LP} = 0.8$, $P_{idle}^{HP} = 0.05$ and $P_{idle}^{LP} = 0.02$, arranged in a FasterP configuration, to demonstrate the execution scenarios and performance trade-offs of the proposed schemes. Table 2 shows two tasks $\tau_1$ and $\tau_2$, with their worst-case execution times (in $ms$) and energy consumptions (in $mJ$) on the low-power and high-power cores and under respective maximum frequencies. For both of these tasks, $a_i^{HP} = 1.0$, $a_i^{LP} = 0.6$, $\alpha_i^{HP} = 0.1$ and $\alpha_i^{LP} = 0.06$. In the following discussion, for brevity, all the $C$ and $f$ values refer to those on the primary (high-performance) core.

Table 2: Example Task Set 2

| | $W_i^{HP}$ | $W_i^{LP}$ | $E^{HP}$ | $E^{LP}$ |
|---|---|---|---|---|
| $\tau_1$ | 22 | 49 | 24.2 | 9.0 |
| $\tau_2$ | 13 | 29 | 14.3 | 5.3 |

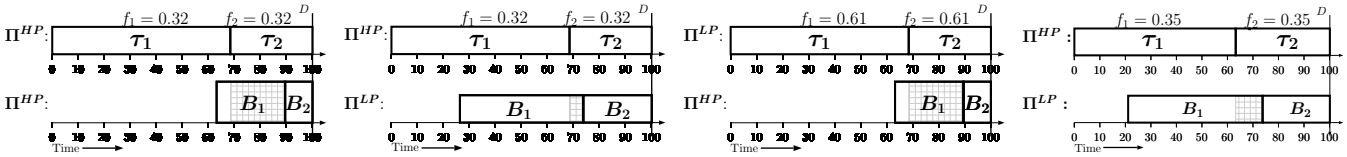
We propose three schemes for the frequency assignment to tasks on the primary core:

1. Static Frequency Assignment (Static)
2. Minimize Overlap for Current Task (MO)
3. Overlap-Aware Energy Minimization (OA)

Note that MO and OA are both dynamic/online schemes that adjust the processor frequency of the primary at runtime for maximum efficiency.

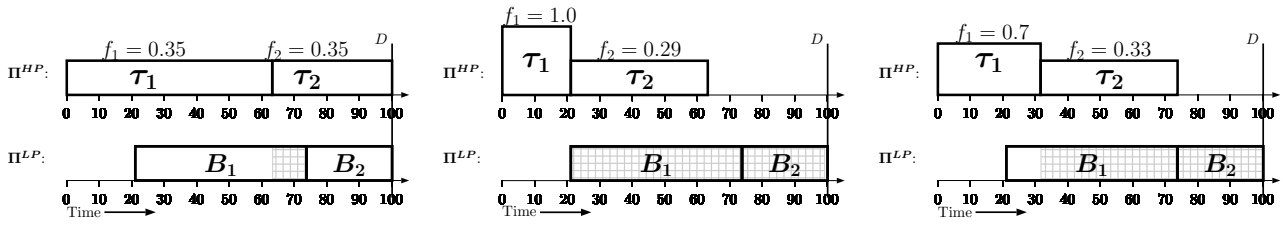
**Static Frequency Assignment (Static).** In this scheme, the frequency assignments to the tasks are done statically, considering the energy-efficient frequency as well as the minimum (uniform) frequency that guarantees the deadline. Specifically, task $\tau_i$ is executed at frequency $f_i = Max(f_i^{ee}, f_U)$, where, $f_U = \frac{\sum C_j}{D}$ is the minimum frequency that ensures meeting the frame deadline for all tasks.

On the spare core, the backups are activated as late as possible and they run at the maximum frequency. Figure 3a shows the execution of the tasks in Table 2 under the Static scheme. The tasks on the primary core are slowed down to the maximum level to save energy, which results in a large overlapped execution with the spare core, and ultimately yields an overall consumption value of 40.54 mJ.



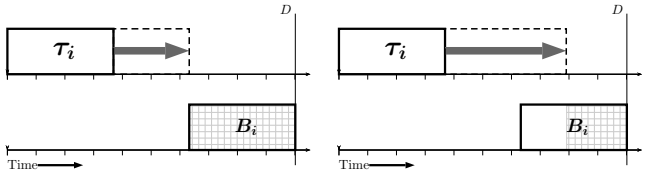

(a) Homogeneous System (b) ($f^{HP}_{max} = 1.0, f^{LP}_{max} = 0.8$) with FasterP configuration (c) ($f^{HP}_{max} = 1.0, f^{LP}_{max} = 0.8$) with SlowerP configuration (d) ($f^{HP}_{max} = 1.0, f^{LP}_{max} = 0.8$) with modified tasks (FasterP)

Figure 2: Executions under different configurations




(a) Static Frequency Assignment (b) Minimize Overlap (MO) scheme (c) Overlap-Aware (OA) scheme

Figure 3: Executions under different schemes




(a) MO-scheme (b) OA-scheme

Figure 4: Frequency assignment with MO and OA schemes

**Minimize Overlap for Current Task (MO).** In this scheme, when a main task on the primary ($\tau_i$) becomes ready to run, we attempt to choose a frequency that would allow the task to complete *before* its backup copy gets activated (Figure 4a). That is, $f_i = Min(f^{max}, f_i^*)$ where $f_i^* = \frac{C_i}{r_i - t}$, $r_i$ is the latest time that the backup copy of $\tau_i$ can be activated without violating any deadlines, $t$ is the current time, and $f^{max}$ is the maximum frequency of the primary core. In case $f_i$ is less than $f_i^{ee}$ or $f_U$, we update $f_i$ as $Max(f_i, f_i^{ee}, f_U)$.

This scheme makes sure that the activation of the current task's backup copy is avoided whenever possible. Figure 3b shows the execution of the tasks in Table 2 under this scheme. The task $\tau_1$ runs at its maximum frequency in order to avoid any overlapped execution, which gives $\tau_2$ enough time to avoid any overlap even when it executes at its energy efficient frequency ($f_2^{ee} = 0.29$). We have found that the overall energy consumption under this scheme is about 33.4 mJ—17.5% less than the one under Static.

**Overlap-aware Energy Minimization (OA).** The MO scheme opts to avoid an overlapped execution whenever it can (Figure 4a). However, we have found that in some cases, allowing some overlapped execution can yield better overall energy performance (Figure 4b). For instance, in a FasterP configuration, the energy gain due to additional slowdown on the primary *may offset* the extra energy consumption due to the overlapped execution on the power-efficient spare.

In this scheme, for a given task, first the energy consumption $E_{MO}$ obtained with the frequency suggested by the MO scheme is computed. Then the frequency that minimizes the energy consumption of the main task and its backup copy, *under the assumption that the overlapped execution will happen,* is obtained, and the resulting energy consumption $E_{OA}$ is evaluated. Finally the solution that yields the lower energy consumption between these two options is selected.

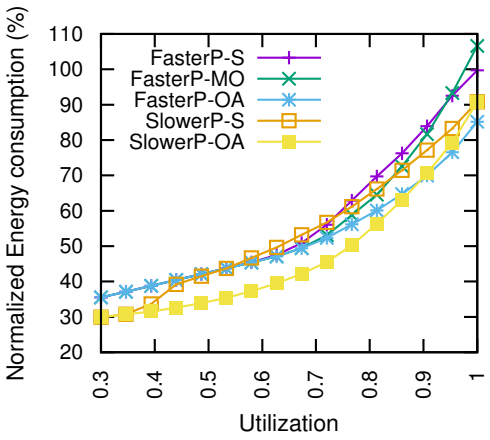
Specifically, the frequency $f_i$ that minimizes the task's energy consumption *in case of overlap with the backup* is obtained by solving the following optimization problem:

$$\begin{aligned} &\text{minimize} && E^{Pr}(f_i) + E^{Bk}(f_i) \\ &\text{subject to:} && f_i^U \leq f_i \leq Min\{f_i^*, f_{max}\} \end{aligned}$$

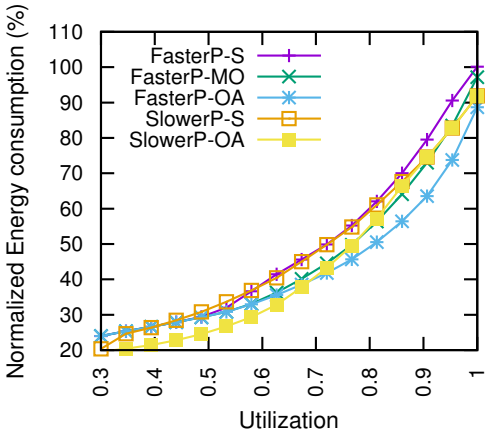
Above, the frequency $f_i$ is restricted between two values, $f_U$ which is the lowest frequency that guarantees the deadline, and $f_i^*$ is the frequency that enables the task to complete at the start time of its backup copy (see the description of the MO scheme). $E^{Pr}(f_i) = P_i^{HP}(f_i) * (C_i^{HP}/f_i)$ is the energy consumed on the primary core, and $E^{Bk}(f_i) = P_i^{LP}(f_{max}^{LP}) * (\frac{C_i^{LP}}{f_i} - (r_i - t)) + P_{idle}^{LP} * (r_i - t)$ is the energy consumed by the backup[1]. The above optimization problem can be solved in constant time, because the objective function is convex and has only one variable.

When there are multiple tasks, there is a need to assign execution windows to each of the tasks to determine the frequency suggested by the OA scheme. In fact, the slack time $S$ (remaining time to finish all the incomplete tasks, which is $D - \sum C_i$) can be distributed among the incomplete tasks to determine individual (pseudo-) task deadlines. We found out that sharing the slack time based on the tasks' relative workload is a quite useful strategy. Therefore, each task $\tau_i$ is assigned a portion of the slack time, $S * \frac{C_i}{\sum C_j}$, and $\tau_i$ must finish execution within this time. The steps are

[1]These equations assume a FasterP configuration.




(a) $f_{max}^{LP} = 0.7$




(b) $f_{max}^{LP} = 0.9$

Figure 5: Impact of utilization

repeated for each task iteratively.

Figure 3c shows the execution of the tasks in Table 2 under the OA scheme. $\tau_1$ executes at low frequency, which results in some overlap with $B_1$. However, the increase in energy consumption due to the overlap is much less compared to the energy savings obtained by reducing frequency on the primary. This scheme yields an overall energy consumption of 26 mJ —35% and 22.2% improvement over Static and MO schemes, respectively.

# 4. EXPERIMENTAL EVALUATION

We evaluated the performances of the proposed schemes by simulating the execution of large number of synthetic task sets in a discrete event simulator. The dual-core systems we simulated have a high-performance core with normalized frequency $f_{max}^{HP} = 1.0$ and a low-power core with normalized frequency $f_{max}^{LP}$ varying in the range $[0.7, 0.9]$.

For a given target total utilization ($\sum \frac{C_i}{D}$), the number of tasks $n$, and a deadline $D$, we have used the *RandFixedSum* [6] algorithm to generate uniformly distributed individual task execution times ($\{C_i\}$ values) on the low-power core. The constants $a_i^{HP}$ and $\alpha_i^{HP}$ are set to 1.0 and 0.1 respectively. Similarly, $P_{idle}^{HP} = 0.05$ and $P_{idle}^{LP} = 0.02$.

It is known that the execution time and power consumption on the high-power core for different tasks scale *by different ratios* when executed on a low-power core [12]. Therefore, for each task $\tau_i$, we define a time-scaling factor $tscale_i = \frac{C_i^{LP}}{C_i^{HP}}$, and a power-scaling factor $pscale_i = \frac{P_i^{LP}}{P_i^{HP}}$. The measurements reported in [12] suggest that $1.4 \leq tscale_i \leq 2.3$ and $1.4 \leq 1/(pscale_i * tscale_i) \leq 2.1$. We generated $tscale_i$ and $pscale_i$ values randomly in these ranges, thereby obtaining execution times and power characteristics of tasks on both types of cores.

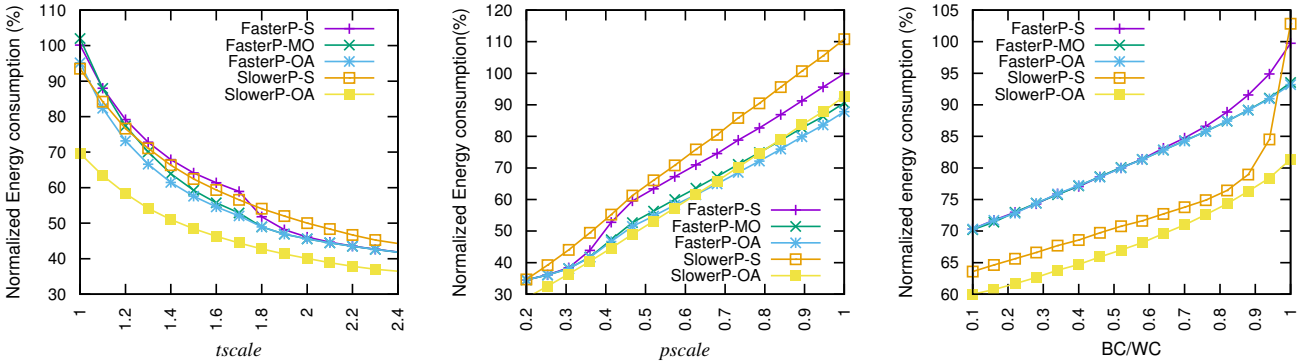
The two core role assignment options (FasterP vs SlowerP), and three frequency assignment schemes (Static, Minimize Overlap (MO), and Overlap-Aware (OA)) give six different combinations shortened as **FasterP-S, SlowerP-S, FasterP-MO, SlowerP-MO, FasterP-OA** and **SlowerP-OA**. Each of the generated task sets is executed by all these six schemes and the results are reported. The common frame period/deadline $D$ is 100ms. For each data point shown in the plots, we computed the average of 3,000 task sets, each containing $n = 10$ tasks.

The trends indicate that for SlowerP configuration, practically in almost all cases, the Overlap-Aware scheme has chosen to minimize the overlap, as suggested by the Minimize Overlap (MO) scheme. This is to be expected, because creating an overlap with the back-up running on the high-performance spare core almost invariably hurts the energy savings in the SlowerP configuration. Therefore, the results we report will not show SlowerP-MO (whose performance is identical to that of SlowerP-OA). However, as we will see, FasterP-MO and FasterP-OA may perform quite differently.

**Impact of utilization.** Figure 5a and 5b show the impact of the system load on two different platforms with $f_{max}^{LP}$ set to 0.7 and 0.9, respectively; $f_{max}^{HP} = 1.0$ in both cases. The X-axis shows the utilization with respect to the low-power core. The results are normalized with respect to the energy consumption of FasterP-S at $U = 1.0$. As expected, the energy consumption of all the schemes increases with the increasing system load. In these plots, the OA (and MO) schemes in the SlowerP group perform best for low load settings, but FasterP-OA starts to outperform when the system load is heavy. For low-load, SlowerP is better because it can use the energy-efficient low-power core to execute all the primary workload, and the overlaps with the back-ups on the high-power spare can be mostly avoided. For heavy load, the situation changes – the spare core generally turns on earlier to meet the deadline requirements of backups. Since the spare always runs at maximum speed, having a slower, power-efficient core helps to reduce energy, and FasterP configurations prove more advantageous.

**Impact of *tscale*.** Figure 6a shows the impact of *tscale* for a moderately-loaded (62.5% on low-power core) system with $f_{max}^{LP} = 0.8$. It can be seen that for the entire region, SlowerP-OA performs distinctly better than others. This is because for this moderate load, the high-power backup core can be kept idle most of the time, and the low-power core will execute the workload avoiding backup activation. We can see that FasterP-S scheme performs worse for low *tscale* values, but improves as *tscale* increases. This is because higher *tscale* values imply a low-power core with large task execution times, and other techniques cannot do much to avoid significant overlapped executions.

**Impact of *pscale*.** In general, low *pscale* values imply




(a) Impact of *tscale* (b) Impact of *pscale* (c) Impact of workload variability

Figure 6: Impact of *tscale*, *pscale* and workload variability

increased energy-efficiency of the low-power core. For these cases, SlowerP-OA perform better for moderate load (62.5% on the low-power core with $f^{LP}_{max} = 0.8$), as can be seen in Figure 6b. But as *pscale* grows, executing the main tasks on the faster core becomes less problematic from the energy standpoint, and at some point FasterP-MO and FasterP-OA start to outperform the SlowerP group.

**Impact of Workload Variability.** The results reported so far correspond to the scenarios where every task takes its worst-case execution time. However, in practice, many tasks complete earlier, without presenting its worst-case workload. To investigate the impact of the workload variability, we define the ratio $BC/WC$ as the ratio of the *best-case execution time* to the *worst-case execution time*. During the experiments, the actual execution time of every task is randomly generated between its worst-case and $BC/WC \times$ worst-case execution time, using a uniform probability distribution.

Figure 6c shows the impact of $BC/WC$ ratio on a moderately loaded system (62.5% on the low-power core). With $BC/WC$, the energy consumption increases, because the system generates less and less dynamic slack that can be used to save energy. SlowerP-OA outperforms other schemes throughout the $BC/WC$ spectrum. FasterP schemes stay worse than the SlowerP group except when $BC/WC$ is very close to 1.0, at which point FasterP-MO and OA outperform SlowerP-S. In fact, FasterP schemes do outperform even SlowerP-OA at high $BC/WC$ values for highly loaded systems (the plots are not shown due to space limitations).

## 5. CONCLUSIONS

In this paper, we investigated the energy-aware standby-sparing systems on emerging heterogeneous multicore platforms. We found out that designating the high-performance or low-power core as the primary can make important differences in terms of energy savings. We also proposed three frequency assignment techniques for tasks on the primary. Our experimental results indicate that, in general, the schemes that assign frequency by carefully evaluating the level of overlap between the main and back-up tasks perform better than those that blindly ignore or avoid such overlaps.

**Acknowledgments:** This work was supported by the US National Science Foundation Awards CNS-1422709 and CNS-1421855.

## 6. REFERENCES

[1] ARM big.LITTLE Technology. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[2] H. Aydin, R. Melhem, and D. Mossé. Tolerating faults while maximizing reward. In *Proc. of the 12th IEEE Euromicro Conference on Real-Time Systems*, 2000.

[3] V. Devadas and H. Aydin. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proc. of the IEEE Green Computing Conference*, 2010.

[4] A. Ejlali, B. M. Al-Hashimi, and P. Eles. A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems. In *Proc. of ACM CODES+ISSS*, 2009.

[5] A. Ejlali, B. M. Al-Hashimi, and P. Eles. Low-energy standby-sparing for hard real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):329–342, 2012.

[6] P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Proc. of the Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2010.

[7] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[8] M. A. Haque, H. Aydin, and D. Zhu. Energy-aware standby-sparing technique for periodic real-time applications. In *Proc. of IEEE ICCD*, 2011.

[9] M. A. Haque, H. Aydin, and D. Zhu. Energy-aware standby-sparing for fixed-priority real-time task sets. *Journal of Sustainable Computing*, 6:81–93, 2015.

[10] I. Koren and C. M. Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[11] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proc. of ACM/IEEE DAC*, 2013.

[12] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. Power-performance modeling on asymmetric multi-cores. In *Proc. of IEEE CASES*, 2013.

[13] M. K. Tavana, M. Salehi, and A. Ejlali. Feedback-based energy management in a standby-sparing scheme for hard real-time systems. In *Proc. of IEEE RTSS*, 2011.

[14] B. Zhao, H. Aydin, and D. Zhu. Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints. *ACM Trans. on Design Automation of Electronic Systems*, 18(2):23, 2013.

[15] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of IEEE ICCAD*, 2004.