Has Energy Surpassed Timeliness?
Scheduling Energy-Constrained Mixed-Criticality Systems

Marcus Völp*†
*School of Computer Science, Logical Systems Lab
Carnegie Mellon University
Pittsburgh, PA, USA
{mvoelp}@cs.cmu.edu

Marcus Hähnel†, Adam Lackorzynski†
†Institute for Systems Architecture, Operating Systems Group
Technische Universität Dresden
Dresden, Germany
{voelp, mhaehnel, adam}@os.inf.tu-dresden.de

***Abstract*—In the past, we have silently accepted that energy consumption in real-time and embedded systems is subordinate to time. That is, we have tried to reduce energy always under the constraint that all deadlines must be met. In mixed-criticality systems however, schedulers respect that some tasks are more important than others and guarantee their completion even at the expense of others. We believe in these systems the role of the energy budget has changed and it is time to ask whether energy has surpassed timeliness. Investigating energy as a further dimension of mixed-criticality systems, we show in a realistic scenario that a subordinate handling of energy can lead to violations of the mixed-criticality guarantees that can only be avoided if energy becomes an equally important resource as time.**

I. Introduction

Over the past three decades, many approaches have been proposed to minimize energy consumption under the constraint that all tasks meet their deadlines. For example, dynamic voltage and frequency scaling (DVFS) has been used to adjust core frequency and in turn also supply voltage to a point where the core is just fast enough to complete all assigned tasks in time [1]. After realizing that in some architectures returns from DVFS are diminishing [2], clock-gating and other circuit-disabling or sleep techniques have been used to increase the gap between activity times [3], keeping cores at maximum power until all real-time tasks complete. Similar approaches have been followed to optimize energy consumption of devices besides the CPU cores. Examples include the WIFI network [4], disks [5], and other IO devices [6]. However, in all these approaches, timeliness is the dominant criterion and energy plays only a subordinate role. In other words, the goal was to guarantee timely completion of all tasks and, while obeying this constraint, to do the best to conserve battery power.

In this paper, we re-evaluate the role of the energy budget in mixed-criticality systems [7], [8]. In these systems, tasks of different importance or criticality for the system's proper functioning are consolidated to facilitate resource sharing even between tasks of different criticality. Tasks are analyzed in more or less stringent ways resulting in more or less pessimistic but also more or less trustworthy estimates on the upper bounds of their execution times. As usual, we call these upper bounds the worst-case execution time estimates (WCETs) of a task at the individual criticality levels. The guarantees conveyed by mixed-criticality schedulers are to complete all tasks of a criticality level provided the WCET estimates at this level hold also for all higher criticality tasks. However, in case these more optimistic estimates cease to hold, the system is prepared to relocate resources to higher criticality tasks to ensure their timely completion even in the most extreme situations. If this happens, lower criticality tasks are dropped which means their timely completion is no longer guaranteed. One potential source for criticality levels are evaluation criteria such as DO-178C [9] or IEC 61508 [10] with their level-specific assurance requirements. However, mixed-criticality scenarios appear also beyond the realm of certification, for example as part of a safety analysis, whenever exceptional situations demand for the timely completion of a subset of tasks without disqualifying the real-time guarantees of the others in less extreme situations (e.g., from hard to soft real-time). It is these systems where we ask ourselves whether it is still justified to unconditionally spend energy to execute lower criticality tasks.

In many systems, energy is easily replenishable and therefore subordinate to timeliness. For example, although refueling airplanes is not as easy, the safety margins for ensuring a safe landing can easily sustain the energy demands of electrical systems, provided the supply of these systems remains operational. However, what happens if the power drops to a point where it is no longer possible to sustain the worst-case energy demand of all tasks simultaneously? Clearly, maintaining real-time communication is important for coordinating the approach to a nearby airport but it is much less important than maintaining control over the aircraft. It is therefore quite natural to sacrifice the timeliness of the former in favor of the latter when such an exceptional situation arises. Satellites are examples where weight, costs and other constraints let us dimension the power supply such that even in normal situations it is not possible to sustain all tasks simultaneously [11]. Clearly loosing the TV signal is unfortunate but much less severe than loosing the satellite in case an advanced attitude and orbit control subsystem has to ignite the propulsion system to avoid collisions.

After providing some background on mixed-criticality scheduling, we show in Section III first with an artificial example and then in a realistic benchmark that subordinate handling of energy can break mixed-criticality guarantees in energy constrained systems. Section IV discusses consequences of allowing the energy demands of higher criticality tasks to surpass timeliness guarantees offered to lower criticality ones. We continue in Section V by translating our findings into a first scheduler for energy-constrained mixed-criticality systems. Section VI presents our evaluation results of this scheduler with randomly generated task sets.

II. Mixed-Criticality Systems

The goal of mixed-criticality scheduling [7], [8] is to integrate real-time tasks of different criticality into the same system. Each task $\tau_i \in T$ is characterized by a criticality level $l_i \in L = \{0, 1, \ldots, l_{max}\}$ and a vector of worst-case execution time estimates C_i. For each criticality level $l \leq l_i$, this vector denotes an upper bound on the task's execution time that is trustworthy with regards to the requirements placed to tasks at this level. As usual, we assume that these estimates become more pessimistic and hence more trustworthy with increasing criticality level (i.e., $C_i(l) \leq C_i(h)$ for $l < h$) and that execution budgets are enforced. That is, the system prevents the jobs $\tau_{i,j}$ of each task τ_i from consuming resources once they exceed $C_i(l_i)$. It is therefore safe to set $C_i(h) = C_i(l_i)$ for all criticality levels $h > l_i$. We denote the largest criticality level by l_{max} and the smallest by 0 and use the constants $LO = 0$ and $HI = l_{max}$ for better readability. Notice that the same hard guarantees are given to LO-criticality tasks, of course up to the trust placed in C_i. If a job $\tau_{i,j}$ exceeds the WCET estimate $C_i(l)$ of a criticality level $l \leq l_i$, we say it executes at criticality level $l + 1$. At this time, it consumes its excess budget $C_i(l, l+1)$ for criticality level $l + 1$, which we define more generally for any two criticality levels $l < h$ as $C_i(l, h) := C_i(h) - C_i(l)$. If, at some time t, l is the largest criticality level at which a job executes, we say the system is l-critical.

The framework for mixed-criticality scheduling does not assume a particular task model. However, to simplify the following discussion, it is beneficial to investigate energy-aware mixed-criticality scheduling in the light of a more specific task model: implicit deadline-constrained sporadic tasks.

Definition 1 (MC Tasks): Let T be a set of implicit deadline-constrained sporadic mixed-criticality tasks. We characterize each task $\tau_i \in T$ by the triple $\tau_i = (\Pi_i, C_i, l_i)$ where $\Pi_i = D_i$ is both the minimal interrelease time between successive jobs of this task and the deadlines of these jobs relative to their release times $r_{i,j}$. The parameters C_i and l_i are, as described above, the vector of WCET estimates and τ_i's criticality level.

The scheduling criterion and guarantee conveyed by energy-agnostic mixed-criticality systems is:

Definition 2 (MC-Schedulability): A set of tasks $T = \{\tau_1, \ldots, \tau_n\}$ is mixed-criticality schedulable if all jobs $\tau_{i,j}$ of tasks $\tau_i \in T$ receive $C_i(l_i)$ time in between $r_{i,j}$ and $r_{i,j} + D_i$, provided that all higher criticality jobs $\tau_{h,k}$ ($l_h > l_i$) require no more than $C_h(l_i)$ to complete.

Because of the guarantee to complete lower criticality tasks if no higher criticality task exceeds its lower WCET budget, situations may arise where the decision to drop a lower criticality task cannot yet be made. In these situations, not executing lower criticality tasks could risk missing their deadlines if all higher criticality tasks complete within low bounds. Following Niz et al. [12], we call this situation *criticality inversion* and the points in time where dropping can finally be decided *criticality decision points*.

Figure 1 depicts such a scenario for the sporadic task set $T = \{\tau_l, \tau_h\}$ with $\tau_l = (4, (1, 1)^T, LO)$ and $\tau_h = (8, (5, 7)^T, HI)$. Unconditionally giving τ_l priority over τ_h

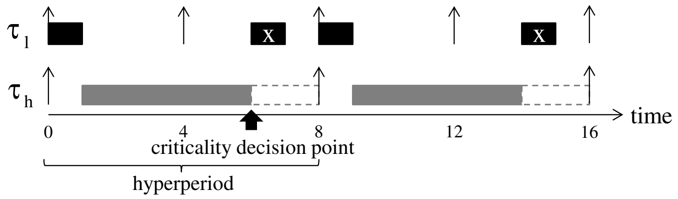

Fig. 1. Energy-agnostic mixed-criticality schedule of the tasks $\tau_l = (4, (1, 1)^T, LO)$ and $\tau_h = (8, (5, 7)^T, HI)$. In each of the two shown hyperperiods, the first job of τ_l has to receive a higher priority than τ_h to not risk missing its deadline in case τ_h completes within $C_h(LO)$. Vertical arrows mark releases and deadlines, solid bars the low-criticality execution budgets $C_i(LO)$ and dashed bars the excess budget $C_h(LO, HI)$ for τ_h. Jobs marked with an 'x' may have to be dropped to make available this excess budget if τ_h exceeds $C_h(LO)$.

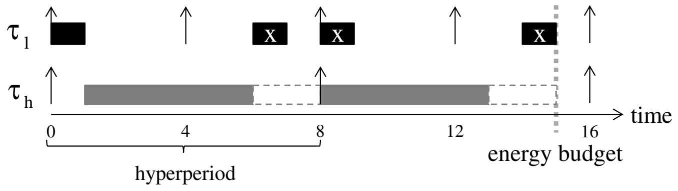

Fig. 2. Example schedule for energy constrained mixed-criticality systems. An energy-agnostic scheduler would prioritize the third job of τ_l over τ_h. However this could drain the energy budget $\hat{E}$ and cause the system to fail before completing τ_h. The vertical dotted line marks the depletion of $\hat{E}$.

could break the mixed-criticality guarantees if τ_h requires its entire high WCET budget of 7 time units and if both jobs of τ_l together execute for more than 1 unit time. On the other hand, prioritizing τ_h over τ_l risks missing τ_l's first job's deadline because τ_h may execute longer than $D_l - C_l(LO)$. Yet, if we allow criticality to be inverted and assign τ_l's first job a higher priority than τ_h but τ_l's second job a lower priority, the system is MC-schedulable. At time 6, after we have granted τ_h an execution budget of 5 time units, a criticality decision point of τ_h is reached. If τ_h completes before this point in time, τ_l's second job $\tau_{l,2}$ may well complete before its deadline. Otherwise, if τ_h has not completed by this time, the scheduler has to drop $\tau_{l,2}$ and relocate its resources to τ_h.

III. Time Dominance can Break Fixed-Criticality

We now demonstrate that subordinate handling of energy can break the mixed-criticality guarantees. We will first take a closer look at the artificial example from Figure 1. After that, we support our findings in Section III-B with a more realistic benchmark involving measure based worst-case execution time and energy estimates for a 16×16 matrix multiplication.

Our goal is to keep the system operational for a given *keep-up time t_{up}* and with a given *total energy budget $\hat{E}_{total}$*. After this time, we assume that either the mission critical part ended or that the budget is replenished to its former value. In this first work, we do not account for memory effects of the energy supply or other supply degrading causes.

A. An Artificial Example

Figure 2 shows an energy-aware schedule of the same task set as we have seen in Figure 1. The two tasks $\tau_l = (4, (1, 1)^T, LO)$ and $\tau_h = (8, (5, 7)^T, HI)$ are executed to a keep-up time of $t_{up} = 16$ units (2 hyperperiods) on a single core. To simplify the example, we assume here that the static energy for keeping this core operational for $t = 16$ time units has already been subtracted from the total energy budget $\hat{E}_{total}$, leaving us a dynamic budget of $\hat{E}_{dynamic} = 15$ energy units. We assume further that this dynamic energy budget is consumed at a rate of one unit whenever a task executes for one unit of time. Throughout this paper, we will use the terms static and dynamic energy only to describe application-level demands. That is, whenever energy is required to keep a resource operational, but not to actually execute on this resource, we attribute its demand to the static energy budget. Therefore, applications do not consume dynamic energy unless they execute[1].

Clearly, we cannot execute any mixed-criticality task set if the dynamic energy budget $\hat{E}_{dynamic}$ cannot sustain the execution for t_{up} time units at each criticality level. In our two level example, this means $\hat{E}_{dynamic}$ must be large enough to cover the high-criticality demand of τ_h of $2 \cdot 7 = 14$ energy units and the low-criticality demand of all tasks, $2 \cdot 5 + 4 \cdot 1 = 14$ energy units.

Figure 1 shows that executing the task set in an energy-agnostic way violates the mixed-criticality guarantees. After executing the first and third job of τ_l for 1 unit time each and the first job of τ_h for 7 time units, the dynamic energy budget is reduced to $15 - 2 \cdot 1 - 7 = 6$ units. This is one unit less than the high worst-case energy estimate $E_h(HI) = C_h(HI)$. The completion of τ_h can no longer be guaranteed in all circumstances. On the other hand, if we continue to drop τ_l's third and fourth job in the second hyperperiod as depicted in Figure 2, τ_h may well complete with the available energy budget.

For the same reason that we have already discussed in Section II, we cannot prioritize τ_l's first job lower than τ_h because then τ_l misses its deadline, even if both jobs of τ_h execute only for $C_h(LO)$. It is easy to see that the completion of all tasks can be guaranteed with the given energy budget if both jobs of τ_h do not exceed their worst-case execution time and energy estimates: τ_h's first job stopping before $C_h(LO) = 5$ leaves two energy units, of which one is used by τ_l's second job. Therefore, $15 - 2 \cdot 1 - 5 = 8$ energy units remain for the second hyperperiod. This budget allows us to continue executing τ_l's third job before τ_h's second job to guarantee the completion of both $\tau_{h,2}$ and, $\tau_{l,4}$ in case $\tau_{h,2}$ does not exceed $C_h(LO)$.

B. Towards a More Realistic Example

Having seen this violation of mixed-criticality guarantees in a hand-crafted artificial scenario, it is interesting to see whether such a situation may also occur in reality. We have therefore measured the execution time and energy consumption of several runs of a 16×16 matrix multiplication under two different stress situations to be able to repeat the above argument with observed worst-case execution times and energy estimates.

1) Experimental setup and measurement methodology: To obtain worst-case estimates on the energy consumption of the 16×16 matrix multiplication, we leverage the energy counters of Intel's Running Average Power Limiting (RAPL) facility [13] on an Intel Core i5-2400S Sandybridge quad-core system. We repeatedly execute the matrix multiplication on one core next to two different types of interfering cache flooders running on the other cores. The flooders interfere with the multiplication through the shared last-level cache L3. The first reads and writes memory as fast as possible to cause evictions from the private L1 and L2 caches of the core performing the matrix multiplication. This eviction is possible because in our benchmark system, the last-level cache is inclusive. That is, it holds all data cached in the core-local L1 and L2 caches and possibly more. Therefore, if a cacheline gets replaced in this last-level cache, private copies are flushed as well. In some rare situations however, additional unforeseen flushes may occur. To accommodate for these rare cases, our second type of flooder actively evicts the matrices by writing to the same memory locations and thereby invalidating possible copies in the remote caches. As the second type of flooder is more pessimistic, we regard the energy and execution time maxima with the first flooder as the results of a low-criticality analysis and the ones with the second flooder as our high-criticality estimates. All measurements were performed on a Gentoo [14] based Linux system, which we have built specifically to minimize interference with the measurements. Except for the base system, all other components have been removed and non-essential services were disabled[2].

2) Results: Figures 3 and 4 show the results of our measurements after flooder side effects have been unfolded (see the Appendix for details on this unfolding). Shown are the maximum energy consumption on the y-axes and the maximum execution time on the x-axes that we observed over in a total of

[1]Dynamic energy may still be consumed at the transistor level (e.g., to refresh the DRAM cells storing an application's data while it is not running). This energy would constitute static energy in our setting.

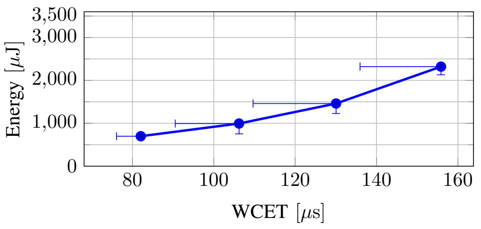

Fig. 3. Low-criticality analysis: observed maximum execution time and energy for a 16×16 matrix multiplication with increasing interference through cache flooders on other cores. The leftmost data point is the undisturbed case. Subsequent data points show the observed worst case estimates with 1, 2 and 3 cache flooding cores. The vertical lines represent the range of the worst-case values over 20 runs.

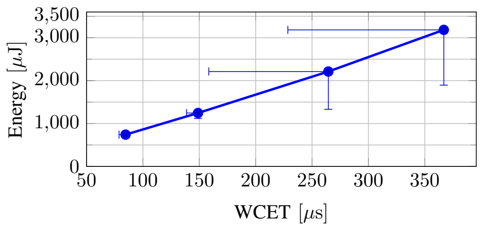

Fig. 4. High-criticality analysis: same experiment as in Figure 3. The cache flooders explicitly invalidate the cache by writing to the memory regions in which the matrices are stored.

[2]Please notice that the choice of our setup was motivated by the availability of the RAPL energy counters and our wish to demonstrate in a realistic scenario that subordinate handling of energy can break mixed-criticality guarantees. In particular we do not advocate Intel Sandybridge as a real-time system nor do we claim that our analysis methods provide strong worst case estimates for these kind of systems.

Task Set parameters for $T = \{\tau_l, \tau_h\}$

	τ_l	τ_h
criticality level	$l_l = LO$	$l_h = HI$
period	$\Pi_l = 230\,\mu$s	$\Pi_h = 460\,\mu$s
WCET estimate	$C_l(LO) = 85\,\mu$s	$C_h(LO) = 156\,\mu$s $C_h(HI) = 376\,\mu$s
WC energy	$E_l(LO) = 742\,\mu$J	$E_h(LO) = 2320\,\mu$J $E_h(HI) = 3183\,\mu$J

System parameters

keep-up time	$t_{up} = 920\,\mu$s
overall energy budget	$\hat{E}_{total} = 11,000\,\mu$J
static power	$P_{static} = 3.65\,$W

TABLE I. PARAMETERS FOR THE REALISTIC SCENARIO

80 runs with 1000 matrix multiplications each. Every 20 runs, we increased the number of interfering cores from zero (no interference) to three (interference from all available cores).

As expected, the interference with the matrix multiplication increases with an increasing number of flooder cores. Notice the low difference in Figure 3 between no interference and full interference. We see a factor of only 1.9 for the execution time and of about 2.7 for the energy consumption. We expected a much larger difference due to the fact that all 16×16 matrices fit completely into the L1 cache. While one core performed the matrix multiplication, the flooders on the other cores accessed memory in the order of the size of the L3 cache to actively evict these matrices from the cache. We therefore assumed that the matrix multiplication will experience mostly misses in the L1 cache. However, in the x86 cache performance counters, we saw that this was not the case. We attribute this behavior and the resulting small differences to Intel's Advanced Smart Cache logic [15]. This logic seeks to maintain an approximately fair share of the L3 cache space for all cores. Rather than evicting the other core's cachelines, the flooders were stalled evicting their own cachelines and leaving the matrices cached.

In some rare cases, the performance counters showed outliers. Although we did not attempt to systematically trigger these rare cases, we want to accommodate for the effect of the Smart Cache logic failing. The second flooder fulfills this purpose by interfering with the matrix multiplication more directly. Figure 4 shows the worst-case estimates, which result from stalling the multiplying core upon missing in the cache. We see a much larger difference between the undisturbed case and the worst case with three concurrent cache-flooders. Energy consumption and execution time is increased by a factor of 4.3.

For our scenario, we take the results obtained with the first flooding method as low-criticality analysis method and the one with the second method as high-criticality analysis. The differences between these two models at full interference are $863\,\mu$J or a factor of 1.37 in terms of energy consumption and of $211\,\mu$s or a factor of 2.35 in terms of worst-case execution time.

3) A realistic scenario: We run one matrix multiplication as a HI task and a second one with shorter period as LO. Both are executed for a keep-up time of $t = 920\,\mu$s, which corresponds to two hyperperiods lasting $460\,\mu$s each. The overall energy budget for this time is $\hat{E}_{total} = 11,000\,\mu$J. Similar to C_i, we arrange the estimates of a task's worst case energy demands into a vector E_i. Table I summarizes the task and system parameters. For the low-criticality task τ_l, we assume an interference free storage location for the matrices. Scratch pad memory and locked-down cache partitions [16] are examples of such locations.

We have measured a static system power demand of $P_{static} = 3.65\,$W for powering up the package. To obtain the energy budget $\hat{E}_{dynamic}$, we subtract from $\hat{E}_{total}$ the static energy required to keep the system up for a period of length t_{up}. That is, $\hat{E}_{dynamic} = \hat{E}_{total} - P_{static} \cdot t = 11000\,\mu\text{J} - 3.65\,\text{W} \cdot 920\,\mu\text{s} = 7642\,\mu$J. Notice, the longest time that the system is guaranteed to be idle is $\Pi_h - 2 \cdot C_l(LO) - C_h(LO) = 460\,\mu\text{s} - 2 \cdot 85\,\mu\text{s} - 156\,\mu\text{s} = 134\,\mu$s when all tasks complete within their low execution time estimates $C_i(LO)$. Turning off the entire package for such short times is intractable. We may therefore disregard the busy periods of the scheduler and keep the tasks' resources up for the entire keep-up time t_{up}.

The discussion now follows the same line of argumentation as our artificial example: we have to prioritize $\tau_{l,1}$ over τ_h because the slack that remains before $\tau_{l,1}$'s deadline (i.e., $\Pi_l - C_l(LO) = 145\,\mu$s) does not suffice to reach $\tau_{h,1}$'s criticality decision point after $C_h(LO) = 156\,\mu$s. But this prevents us from repeating the schedule in the second hyperperiod with the same priority assignment as for the first because the energy budget that remains after executing τ_l's first and third job and τ_h's first can be as low as $\hat{E}_{dynamic} - 2 \cdot E_l(LO) - E_h(HI) = 7642\,\mu\text{J} - 2 \cdot 742\,\mu\text{J} - 3183\,\mu\text{J} = 2975\,\mu$J. This is not enough to guarantee the completion of τ_h's second job[3].

Notice that we cannot simply divide up the dynamic energy budget equally among hyperperiods. To schedule one job of both τ_l and τ_h, an energy budget of $\hat{E}_{split} = E_l(LO) + E_h(HI) = 742\,\mu\text{J} + 3183\,\mu\text{J} = 3925\,\mu$J is required. However, the per hyperperiod share of $\hat{E}_{dynamic}$ is only $\hat{E}_{dynamic}/2 = 3821\,\mu$J. In case the energy budget is large enough to allow for such a splitting, we say the system is *energy balanced*. It is easy to see that a feasible energy-agnostic mixed-criticality scheduler is also feasible for an energy constrained system if the worst case energy consumption in a hyperperiod is below or equal to $\hat{E}_{split}$. In the following, we therefore focus our attention to energy-imbalanced systems but emphasize the need for a worst case energy analysis for the balanced case.

IV. COPING WITH ENERGY AS DOMINANT RESOURCE

The previous section has shown us two examples where subordinate handling of energy violated the mixed-criticality guarantees formulated in Definition 2. The violation resulted from a fundamental difference between scheduling time and scheduling energy: From a time scheduling point of view, consuming part of the excess budgets for higher criticality tasks or completing tasks early takes effect immediately. That is, tasks of the same hyperperiod may have to be dropped to not risk missing a deadline or, in case a task stopped early, background tasks could be scheduled or tasks that would otherwise have been dropped. On the other hand, if a task consumes part of the excess energy budget early on, the effect may well be delayed to short before the keep-up time.

[3]For completeness, the low-only case ($4 \cdot E_l(LO) + 2 \cdot E_h(LO) = 4 \cdot 742\,\mu\text{J} + 2 \cdot 2320\,\mu\text{J} = 7608\,\mu$J) and the high case in the second hyperperiod case ($E_l(LO) + 2 \cdot E_h(HI) = 742\,\mu\text{J} + 2 \cdot 3183\,\mu\text{J} = 7108\,\mu$J) are both covered by $\hat{E}_{dynamic} = 7642\,\mu$J.

Likewise, early savings are carried along all the way and allow us to schedule additional tasks that would otherwise have to be dropped.

The above examples have also shown us that mixed-criticality scheduling for energy-constrained systems must consider energy and timeliness as equally important measures and that equally splitting the energy budget among all remaining hyperperiods does not necessarily result in a feasible schedule. In the next section, we transform these results into a feasible energy-aware mixed-criticality scheduler. However, before that, let us extend our task model to incorporate energy consumption and clarify what it means for task sets to be schedulable in energy-constrained mixed-criticality systems.

A. *Mixed-Criticality Tasksets for Energy-Constrained Systems*

We say a mixed-criticality system is *energy-constrained* if it has to remain operational for a given keep-up time t_{up} with a given limited energy budget $\hat{E}_{total}$. We now characterize the tasks of a set of tasks $T = \{\tau_1, \ldots, \tau_n\}$ as four-tuples $\tau_i = (\Pi_i, C_i, E_i, l_i)$ where Π_i, C_i and l_i are as described in Section II the minimal interrelease time, the vector of worst-case execution time estimates and the criticality level of the task. The parameter E_i is a vector of worst-case estimates for the dynamic energy consumption of the task. That is, for some criticality level l, $E_i(l)$ is an estimate of the maximum dynamic energy that τ_i will consume while executing for at most $C_i(l)$ time units. If power consumption can be limited (e.g., with the limiting facility in Intel's RAPL), the estimate $E_i(l)$ need to be trustworthy only up to level l. In this case it safe set $E_i(h) = E_i(l_i)$ for all criticality levels $h > l_i$. Otherwise, $E_i(l)$ must be trustworthy at the highest criticality level l_{max}. We assume in the following that the static energy was already subtracted from the total energy budget $\hat{E}_{total}$, leaving us the dynamic energy budget $\hat{E}_{dynamic} = \hat{E}_{total} - \hat{E}_{static}$. Notice that $\hat{E}_{static}$ and likewise E_i must include the energy demand of all resources a task requires.

B. *Mixed-Criticality Schedulability for Energy-Constrained Systems*

We now define the criteria when a task set T is mixed-criticality schedulable in an energy-constrained system.

Definition 3 (MC-schedulability (energy constrained)): A set of mixed-criticality tasks $T = \{\tau_1, \ldots, \tau_n\}$ is schedulable in an energy-constrained system with keep-up time t_{up} and dynamic energy budget $\hat{E}_{dynamic}$ if each job $\tau_{i,j}$ of a task in T is granted $C_i(l_i)$ in between $r_{i,j}$ and $r_{i,j} + D_i$ provided that

1) all jobs of higher criticality tasks τ_h ($l_i < l_h$) complete before $C_h(l_i)$ and
2) the energy budget is large enough to meet the demand of all these tasks:

$$\sum_{\tau_h \in T | l_i \leq l_h} \left\lceil \frac{t}{\Pi_h} \right\rceil E_h(l_i) \leq \hat{E}_{dynamic}. \qquad (1)$$

Notice the change from $<$ to $\leq$ in Equation 1. This is to reflect that $\hat{E}_{dynamic}$ must be large enough to also meet the energy demand of l_i-criticality tasks. Like before (Definition 2), we did not include any claim in Definition 3 stating whether or when to resume giving guarantees to dropped tasks. Of course it is desirable to return to lower criticality levels as quickly as possible to quickly regain full functionality.

V. Towards Energy-Aware Mixed-Criticality Schedulers

Let us first focus on the class of work-conserving fixed-job-priority schedulers. From Baruah and Vestal [17] we know that sporadic tasks are MC-schedulable if they are MC-schedulable at their synchronous arrival sequences (SAS). In such a sequence, all tasks τ_i have their first job released at time 0 and subsequent jobs follow Π_i apart. In [18], Baruah et al. showed that among all fixed job priority scheduling algorithms those following the Own Criticality Based Priority Assignment (OCBP) yield the smallest speedup bound. That is, if a task set T is MC-schedulable then OCBP produces a priority assignment for scheduling T on a processor that is s_L times faster and no other job-level fixed priority (JFP) algorithm, including EDF, yields a smaller bound. Therefore, by augmenting OCBP, we obtain a family of scheduling algorithms that, as far as timing is concerned, is optimal in this class of fixed-job-priority scheduling algorithms. Our goal is to identify in this family the OCBP priority assignment that yields the lowest worst-case energy demand because this demand directly translates into the size and weight of the required battery and into the time the system remains operational.

A. *Admission*

To guarantee operation up to the system keep-up time t_{up}, we have to ensure that the energy budget $\hat{E}_{dynamic}$ is able to sustain LO execution but also possible transitions to higher criticality levels. Although desirable from a usability perspective, we do not have to guarantee returns to lower criticality levels. For the admission, it is therefore sufficient to consider only monotonic increases of the criticality level. Unfortunately, any job may cause such a transition and, as a result, the dropping of lower criticality jobs. Of course, calculating the energy demand for all combinations of $0 - l_{max}$ jobs transitioning to higher criticality levels in between time 0 and t_{up} is not feasible in practice. We therefore approximate the system's overall energy demand from upper bounds for individual hyperperiods of length HP and show in Section V-A2 how to derive these bounds for work-conserving OCBP schedulers with reasonable effort.

1) Admitting task sets based on hyperperiod energy bounds: Let $E^{HP}(l, h)$ be an upper bound on the energy required to complete all jobs in a single hyperperiod when starting execution at criticality level l and ending in the same hyperperiod with criticality level h ($l \leq h$). Let further k be the criticality level that generates the highest demand in one hyperperiod when no criticality change happens (i.e., $E^{HP}(k, k) = \max_{l \in L} E^{HP}(l, l)$). If now, as usual, the number $n_{HP} = \left\lceil \frac{t_{up}}{HP} \right\rceil$ of hyperperiods that fill up the system keep-up time t_{up} is large compared to the number of criticality levels, the upper bound of the overall system energy demand is dominated by the hyperperiods that remain at criticality level k. However, because we have chosen $E^{HP}(k, k)$ only to be the maximum energy demand of those hyperperiods that do

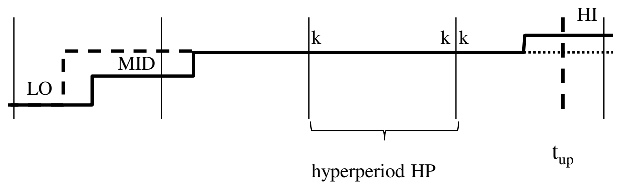

Fig. 5. Admission test based on hyperperiod energy bounds. Excluding transitional hyperperiods, the system energy demand is greatest ($E^{HP}(k,k)$) while it remains at level k. The worst case combination for the entire keep-up time of the system is therefore bounded by the worst case combination to reach k and to remain there or possibly exit to a higher criticality level (HI). For a simpler analysis, we combine the two step increase from LO via MID to k into a single step increase LO to k (dashed line) by adjusting $E^{HP}(LO,k)$. We consider the energy demand for a full hyperperiod even if t_{up} falls in between two simultaneous releases of all tasks.

not change criticality, we must also consider the stepwise or immediate increase from 0 to k over at most k hyperperiods at the beginning and from k to l_{max} over at most $l_{max} - k$ hyperperiods at the end. We do so by systematically coalescing the energy demand of multi-hyperperiod sequences into an upper bound for the single hyperperiod energy demand. More precisely, if we find a sequence of $m \leq k$ criticality levels $l_0, \ldots l_m$ with $l_0 = LO$ and $l_m = k$ such that

$$\sum_{i=0}^{m-1} E^{HP}(l_i, l_{i+1}) > E^{HP}(LO,k) + (m-1)E^{HP}(k,k) \quad (2)$$

we adjust $E^{HP}(LO,k)$ to reflect the additional costs of this gradual increase. That is, we set

$$E^{HP}(LO,k) \leftarrow \sum_{i=0}^{m-1} E^{HP}(l_i, l_{i+1}) - (m-1)E^{HP}(k,k). \quad (3)$$

Figure 5 illustrates this procedure graphically.

With a similar adjustment of all $E^{HP}(k,h)$ for $k \leq h \leq l_{max}$), the worst case energy demand for $n_{HP} > 2$ hyperperiods is

$$\begin{aligned} E^T \;=\; & E^{HP}(LO,k) + (n_{HP} - 2)E^{HP}(k,k) + \\ & \max_{k \leq h \leq l_{max}} E^{HP}(k,h) \end{aligned} \quad (4)$$

and the system is feasible from an energy point of view if $E^T < \hat{E}_{dynamic}$. All that remains now is to compute the initial bounds $E^{HP}(l,h)$ of hyperperiod energy demands.

2) Computing hyperperiod energy bounds: To compute hyperperiod energy bounds for work-conserving fixed-job-priority schedulers, we rely on the specific priority assignments produced by OCBP. Figure 6 Lines 1 – 18 show the C-like pseudo code of OCBP. Given a set of jobs $\mathbb{J}$ and the maximum index $n = |\mathbb{J}|$, OCBP picks one job j_k after the other and checks the feasibility of scheduling $\mathbb{J}$ if j_k receives the lowest priority π_k. It does so by picking an arbitrary priority assignment for all other jobs in $\mathbb{J}$ (e.g., earliest release time first) and simulating the schedule assuming that all jobs receive $C_i(l_k)$ time (ignoring their deadlines). If in this simulation sufficient time remains in between the release r_k of j_k and its deadline ($r_k + D_k$), OCBP continues with $\mathbb{J} \setminus \{j_k\}$ and tries to find a priority assignment for all remaining jobs.

As OCBP operates on jobs rather than tasks, we must first transform the sporadic tasks in T into the jobs these tasks

```
1  bool check_ocbp($\mathbb{J}$, $n$) {
2    for (k = 0...n) {
3      if ($\mathbb{J} == \emptyset$) return true;

5      // skip job indices we already considered
6      if ($j_k \notin \mathbb{J}$) continue;

8      // check feasibility
9      assign $j_k$ priority $\pi_k = |\mathbb{J}|$;
10     pick arbitrary priorities for $\mathbb{J} \setminus \{j_k\}$;
11     simulate schedule assuming $C_i(l_k)$
12       and ignore deadlines;

14     if ($C_k(j_k)$ time remains in $[r_k, r_k + D_k]$) {
15       $\mathbb{J}$ = $\mathbb{J} \setminus \{j_k\}$;
16       k = 0;
17       continue;
18     }
19   }
20   return false;
21 }

23 bool augmented_ocbp($T$) {
24   transform $T$ into the set of jobs $\mathbb{J}$;
25   n = $|\mathbb{J}|$;

27   // guide search
28   sort $\mathbb{J}$ by increasing criticality levels;
29   sort jobs $j_k$ of the same criticality level
30     by decreasing energy weight $E_k(l_k)\frac{C_k(l_k)}{\Pi_k}$;

32   // OCBP
33   return check_ocbp($\mathbb{J}$, $n$);
34 }
```

Fig. 6. C-like pseudo code of augmented OCBP algorithm.

will issue in their synchronous arrival sequence. We do so by setting the release r_k of the job $j_k = \tau_{i,j}$, $j = 0, 1, \ldots$, to $r_k = j\Pi_i$. To obtain an energy optimal OCBP priority ordering, we present jobs to OCBP by first sorting them in order of increasing criticality level and then the jobs of the same criticality level in order of decreasing energy weight (see Equation 5 below).

For better readability, we have adjusted the job indices in the above paragraph and in the following and refer to task parameters through their job indices. For example, we write $l_k = l_i$ for the criticality level of the job $j_k = \tau_{i,j}$. In the following, we explain the intuition for presenting OCBP jobs in this particular order.

a) Criticality Monotonic First: If jobs are sorted by increasing criticality level, the OCBP algorithm in Figure 6 will first check the feasibility of criticality monotonic priority assignments. For work-conserving schedulers, such assignments are preferable because lower criticality jobs run only if no higher criticality job is active (i.e., released and not yet finished). Therefore, if a higher criticality job j_h gets released, we will first see its criticality points before we continue lower criticality jobs.

It is easy to see that a criticality monotonic priority assignment yields a lower worst-case energy demand when compared to a non-monotonic assignment where the relative priorities of equally critical jobs is preserved: There is no chance for simultaneously active lower prioritized but higher criticality jobs to cancel the higher prioritized lower criticality jobs, which is trivially possible if priorities are criticality monotonic.

Please notice that we did not claim that criticality

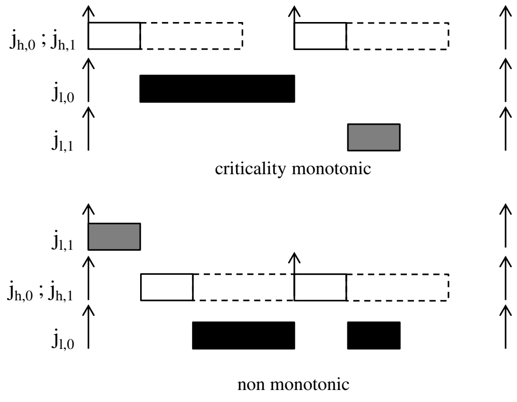

Fig. 7. Criticality monotonic and non-monotonic schedule of the job set $\mathbb{J}$.

monotonic assignments are in general less energy demanding. The counterexample in Figure 7 illustrates the need for our additional constraint that relative priority ordering must be preserved and, more indirectly, why we sort equally critical jobs by their energy weight. Let $\mathbb{J}$ consist of the following four jobs characterized by release time, period, criticality level, execution time vector and energy vector: $j_{h,0} = (0, 4, HI, (1, 3)^T, (1, 3)^T)$, $j_{h,1} = (4, 4, HI, (1, 3)^T, (1, 3)^T)$, $j_{l,0} = (0, 8, LO, (3, 3)^T, (6, 6)^T)$ and $j_{l,1} = (0, 8, LO, (1, 1)^T, (1, 1)^T)$. All jobs require 1 unit energy for executing over 1 time unit except $j_{l,0}$, which requires two energy units for one unit time. A criticality monotonic priority assignment $\pi_{h,0} > \pi_{h,1} > \pi_{l,0} > \pi_{l,1}$ yields a maximal energy demand of 9 if no job exceeds $C_i(LO)$; 6 if the first high job exceeds this value and 10 if the second high job does so. Hence, $max(9, 6, 10) = 10$ energy units. However, prioritizing $j_{l,1}$ over all other jobs gives us a demand of only $max(9, 7, 9) = 9$ energy units. The reason for these energy savings lie in the fact that we have first prioritized the energy heavy task $j_{l,0}$ over $j_{l,1}$ and then inverted this relative priority ordering. Had we prioritized $j_{l,1}$ over $j_{l,0}$ in the first place, we would not have seen any reduction in the worst case energy demands. The strategy for obtaining a worst-case energy optimal schedule is to keep energy heavy tasks as low prioritized as possible.

b) Minimizing Energy Weight: To characterize how energy heavy a task is, we introduce the energy weight of a job as

$$w_k := E_k(l_k) \frac{C_k(l_k)}{\Pi_k} \tag{5}$$

By presenting OCBP jobs of the same criticality level sorted in order of decreasing energy weight, it first searches for feasible priority orderings where in a work-conserving scheduler energy heavy tasks execute as late as possible.

c) Putting it all together: Finally, we obtain the energy bound $E^{HP}(l, h)$ from the above OCBP priority assignments, which we call EA-OCBP, by simulating the schedule for all combinations of jobs transitioning gradually or immediately from l to h. For this simulation, we assume execution times of $C_i(l_k)$ and energy demands of $E_i(l_k)$ for as long as the system executes at criticality level k. Notice, because the number of distinct criticality levels is typically small, computing and storing for each combination of criticality levels l, h (where $l \leq h$) the worst-case energy bound $E^{HP}(l, h)$ is time consuming but within feasible bounds for an off-line analysis. Once we have computed these bounds, we make use of them both for the admission and to decide in our online scheduler when it is safe to return to lower criticality levels.

B. Online Scheduling

With the admission test in place, we also have an easy way of returning to a lower criticality level l, provided we have performed the adjustment of $E^{HP}(l, k)$ that we have described in Section V-A1 for all criticality levels $LO \leq l < k$. We modify an online job-level fixed priority scheduler to keep track of the dynamic energy $E_{dynamic}(t)$ that the system has consumed up to time t. If appropriate sensors are available, $E_{dynamic}(t)$ can be drawn directly from these sensors after subtracting static energy parts. Otherwise, if the system is not equipped with suitable sensors, an upper bound on the consumed energy can be obtained by accumulating worst-case energy estimates for the criticality levels jobs execute in. More precisely, at the point in time when a job $\tau_{i,j}$ (executing at criticality level l) exceeds $C_i(l)$, we add $E_i(l, l+1)$ to the estimate $E_{dynamic}(t)$.

From the admission we know that gradual and immediate increases of criticality levels cannot violate the mixed-criticality guarantees. When at some time t $(0 < t < t_{up})$ we consider switching back from the current criticality level m to a lower criticality level l, we therefore have to ensure that sufficient energy remains to complete at this level l or at any higher level the system may transitions to. We do so by assuming pessimistically that a new hyperperiod starts at t. If the remaining energy budget $\hat{E}_{remaining}(t) = \hat{E}_{dynamic} - E_{dynamic}(t)$ can now sustain switching from l to k and from k possibly to some higher level $h \geq k$ such that $E^{HP}(k, h)$ is maximized, it is safe to return to l. With the exception of two corner-case situations, this is the case if

$$\hat{E}_{remaining}(t) > \\ E^{HP}(l, k) + (\left\lceil \tfrac{t_{up} - t}{HP} \right\rceil - 2) E^{HP}(k, k) + E^{HP}(k, h) \tag{6}$$

The first corner case occurs if it is not feasible to return to a lower level than k. In this case, assuming the system will remain k-critical for the majority of all hyperperiods gives only a rough but safe approximation of the worst case energy demand. The second corner case is when $t_{up} - t \leq 2HP$. In this case, we have to consider all criticality levels between l and HI and check whether

$$\hat{E}_{remaining}(t) > \max_{l \leq m \leq h \leq HI} (E^{HP}(l, m) + E^{HP}(m, h)) \tag{7}$$

C. Beyond Work-Conserving Fixed-Priority Scheduling

Giving up the requirement that jobs are scheduled in a work conserving manner, we may further improve on the worst-case energy demand by exploiting slack time to further defer the execution of energy heavy lower criticality jobs. In [19], Baruah et al. improve on the speedup bound by allowing jobs to obtain more than one priority. Similarly, Vestal [7] suggests splitting high prioritized jobs into smaller ones to maintain criticality monotonic assignment. An elaborate discussion of these dynamic priority schemes has to remain future work, however, we expect that the general approach of executing energy heavy low criticality jobs as late as possible remains applicable also in these multi-priority settings.

VI. Evaluation

We have evaluated our approach by simulating the scheduling of randomly generated task sets and compared the energy budgets required to complete energy-agnostic OCBP schedules (*OCBP*) and energy-aware schedules (*EA-OCBP*).

Task set generation follows the approach taken in [20]. We use UUniFast [21] by Bini and Buttazzo to generate tasks with a uniformly distributed high-criticality utilization $u_i(HI)$ in the range $[0, 1]$. Periods range between $\Pi_{min} = 2$ and $\Pi_{max} = 30$. To guarantee the presence of tasks with small as well as large periods, we follow Davis and Burns [22] and generate periods using a binning technique, which partitions the available range into bins of exponentially growing width. From those bins we draw periods in a round robin fashion. To obtain the vector C_i, we set $C_i(HI) = u_i(HI) \cdot \Pi_i$ and $C_i(l-1) = C_i(l) \cdot f$ for $l \leq HI$. Here, f is the criticality factor, which we vary in our experiments between 0.6 and 0.8. For each lower criticality level, we use the utilization

$$u = \sum_{i=1}^{|T|} \frac{C_i(l) - C_i(l-1)}{\Pi_i} \tag{8}$$

that remains when subtracting high excess budgets to create new tasks for criticality level $l-1$ using the same method. For our experiments we limited ourselves to two and three criticality levels with an equal amount of tasks in each level. The total number of tasks was chosen randomly between 4 and 10. We also varied the execution time and energy ratio r from $r_A = 1:1$ (unbalanced) to imbalanced ratios, which take the criticality level of a task into account ($r_B = 1 : (l+1)$ and $r_C = 1 : 2(l+1)$). Imbalanced ratios are justified because high criticality tasks are typically much simpler and require less expensive resources than for example a low video player exercising the full streaming instruction set. For our experiments we limited the hyperperiod to be simulated to 200 time units to reduce simulation time. Task sets that did not adhere to this limit were discarded. This was necessary due to the amount of jobs created for long hyperperiods, and the fact that OCBP derived algorithms have a complexity of at least $\mathcal{O}(n \cdot (n-1))$ (see [23]).

With randomly generated task sets, picking a fixed keep-up time t_{up} and a fixed (dynamic) energy budget $\hat{E}_{dynamic}$ gives only a limited insight into the performance of our algorithm because only a few tasks will require close to $\hat{E}_{dynamic}$. We have therefore evaluated our approach in terms of the energy budget that is necessary to sustain the generated task set.

For this evaluation we calculated the maximum energy consumption for all possible criticality level transitions for the generated tasks when scheduled with our algorithm and compared the results to the schedules obtained from presenting OCBP an unsorted job list. We found, that for a simple energy ratio inversely proportional to the criticality level of the task (higher criticality means lower ratio down to 1:1) our algorithm required up to 17% smaller budgets than an unsorted OCBP scheduler, while still guaranteeing schedulability as defined in Definition 3. The detailed energy comparison for

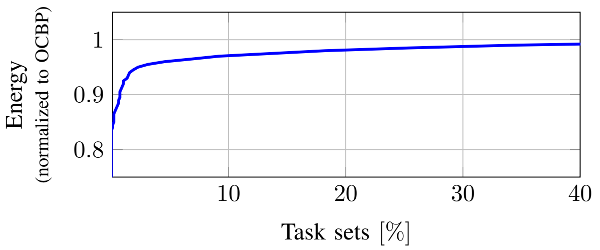

Fig. 8. Energy budget required for different task sets relative to OCBP

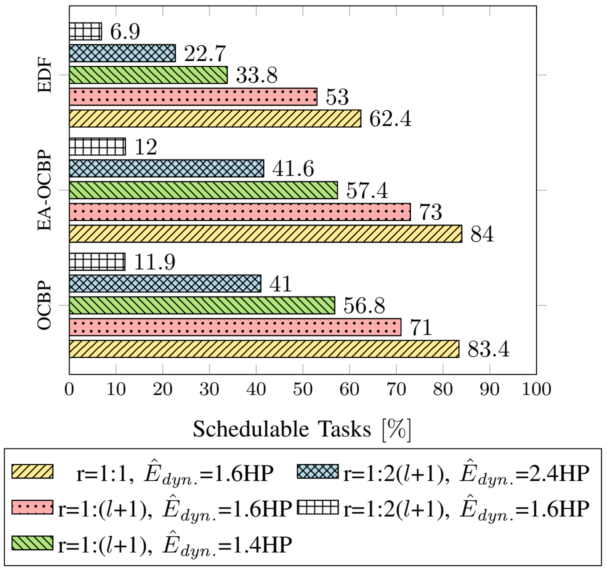

Fig. 9. Percentage of schedulable task sets

this scenario is shown in Figure 8. The y-axis gives the EA-OCBP energy budget required to guarantee energy constrained MC-schedulability, normalized to the energy budget required for stock OCBP. The x-axis gives the percentage of task sets EA-OCBP can sustain when given the fraction of the energy budget denoted by the y-value. We evaluated 5,000 task sets for two criticality levels and 1,000 task sets for three levels.

Figure 9 shows the schedulability comparisons for several scenarios. Figure 10 shows the maximum schedulable utilization graph of a simulation where we fixed the energy budget to 1.6 hyperperiods with a simulation time of two hyperperiods. Our algorithm was able to schedule all task sets that were schedulable with standard OCBP and in addition 46,000 instances of those task sets where OCBP failed due to energy constraints. This resulted in a schedulable utilization of 71% of for OCBP and 73% for EA-OCBP.

VII. Related work

To the best knowledge of the authors, this is the first paper approaching energy-aware real-time scheduling from the perspective that energy has surpassed timeliness. Yet, naturally, there is a large body of work on subordinate handling of energy. For example, we have already seen in the introduction how DVFS [24]–[27] and clock-gating [3] based approaches optimize energy consumption while meeting all deadlines. Fu et al. [28] and Völp et al. [29] propose to safe energy by dynamically consolidating real-time tasks to fewer cores than the admission requires.

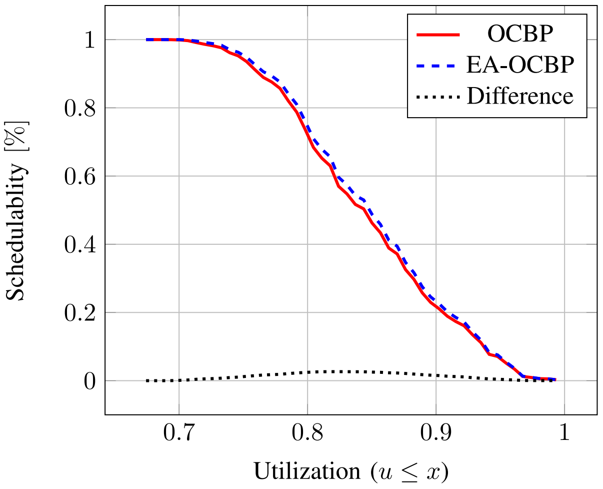

Fig. 10. Schedulable utilization of OCBP vs. EA-OCBP

Guan et al. [30] address the run-time complexity and hence the energy use of the mixed-criticality scheduler itself. With the exception of deciding when to return to lower criticality levels, our scheduler has no significant energy overheads when compared to traditional fixed-job-priority schedulers.

Rusu et al. [31] take an economy driven approach in their investigation of energy-aware real-time scheduling. Like our approach, they focus on energy constrained systems where both time and energy are critical resources. However, unlike our approach, their primary objective is to complete the most important/valuable applications even if this results in dropping unimportant (or in our case low criticality) applications before a high criticality task has exceeded their low WCET estimate. By correlating the task version k of Rusu's approach with a criticality level, our approach could serve as a replacement for their static analysis (which provides a guarantee for the worst-case scenario). The additional benefit is our guarantee of a minimal value for all tasks if the resource estimates for the low task versions suffice to complete the higher versions of a task.

The idea to present tasks to OCBP in a pre-sorted manner was first exercised by Baruah, Burns and Davis [23, Theorem I] to reduce the run-time complexity of OCBP.

VIII. Conclusion

In this paper, we have made the case for energy surpassing timeliness in energy-constrained mixed-criticality systems. Based on a realistic scenario, we have shown that subordinate handling of energy is no longer justified when, like in mixed-criticality systems, some tasks are more important than others. Rearranging the order in which jobs are presented to the own criticality based priority (OCBP) assignment algorithm, we turn OCBP into a mixed-criticality scheduler for energy constrained systems, achieving up to 17 % improvement in terms of worst-case energy demand. Although, admittedly, only a few systems operate at a point where energy is constrained such that mixed-criticality guarantees are at risk, we are confident that the trend towards autonomous, mobile and dependable systems will increase this number of energy-constrained mixed-criticality systems.

Appendix
Obstacles Measuring Worst-Case Energy Consumption with RAPL

With the Sandy Bridge processor line, Intel has introduced an on-board power measurement and limiting facility called RAPL [13]. Contrary to performance-counter-based methods, RAPL is based on a manufacturer-provided model of the processor that is directly built into the CPU and calibrated during start-up. The update frequency and hence the precision interval of the RAPL-provided energy counters is one millisecond, although it is possible to obtain results for much shorter executions if one can tolerate executing these code pieces in a specifically tailored setup [32]. Although as the name suggests, RAPL is also equipped to limit power consumed over a configurable time window, this facility has been locked down on the benchmark system we used. We have therefore limited our investigation of the real-life counterexample to obtaining accurate worst-case estimates on the consumed energy and not on enforcing the energy budget.

There are several obstacles which limit the use of RAPL for a fine grained estimation of task energy consumption. For instance, unlike performance counters [33], the RAPL energy counters are updated asynchronously to the execution. For this reason, it is not straight forward to multiplex these counters among threads unless thread switches are aligned to the millisecond update intervals. Moreover, counters so far exist only for the package domain (RAPL_PKG), the core domain (RAPL_PP0) and graphics core (RAPL_PP1) of the chip. The package domain aggregates the energy consumption of all devices in the package whereas the core domain measures the core energy consumption as an aggregate of all cores taken together. For these reasons, our benchmark was limited to a single core only with the other cores interfering. To obtain repeatable results, we have aligned the start of the matrix multiplication on the first core and the start of the interfering tasks with each other. After obtaining the raw results, we had to unfold the flooder energy to get the estimates for the matrix multiplication.

The following three-phased analysis provided us with reasonable approximations for these estimates. The first phase did not run any of our benchmarks but was used to determine the static power P_0 and the power of the uncore system P_{uncore}. For the static power all cores were left idle while we measured their energy consumption over a certain period of time. For the uncore power we ran an experiment, where we executed the same code on one, two, three and four cores while measuring the package power P_{pkg-i} for the respective core count. The cores did not access shared memory for this run. We calculated the costs of adding another core by averaging the total power consumption differences of the multiple core runs. The single core power reduced by this value is the resulting value of P_{uncore}. Please note that the uncore power does *not* include the static power. The equation to find the uncore power for n cores is $P_{uncore} = P_{pkg_1} - \frac{\sum_{i=1}^{n-1}(P_{pkg_{i+1}} - P_{pkg_i})}{(n-1)}$

We verified that the additional power required for adding cores after the first is constant and that it does not depend on the core ID. Table II shows the values we measured. For our setup the uncore power P_{uncore} was 2.27 W.

In the second phase we measured the system power P_{total}

Active cores (i)	0	1	2	3	4
Power P_i [W]	3.65	11.50	17.05	22.60	28.25

TABLE II. SYSTEM POWER CONSUMPTION FOR A VARYING NUMBER OF ACTIVE CORES

for the matrix multiplication with the cache flooders and in the third phase just for the cache flooders ($P_{total_{flooder}}$). The overhead of the flooders $P_{overhead}$ can be calculated as $P_{overhead} = P_{total_{flooder}} - P_0 - P_{uncore}$. The final energy consumption for the matrix multiplication E_{matrix} is $E_{matrix} = P_{total} - P_{overhead} * t_{matrix}$ where t_{matrix} is the duration of the matrix multiplication.

To obtain the values shown in Figures 3 and 4, we included the uncore energy in the values for the matrix multiplication. Notice however, that if a system schedules tasks on multiple cores it has to divide this power to all of them.

ACKNOWLEDGMENTS

This work was in part funded by the DFG through the collaborative research center "Highly Adaptive Energy Efficient Systems" (HAEC) and through the cluster of excellence "Center for Advancing Electronics Dresden", by the EU and the state Saxony through the ESF young researcher group "IMData" and by the NSF through grant CNS-0931985. Thanks to Claude-Joachim Hamann for many helpful comments.

REFERENCES

[1] A. Dudani, F. Mueller, and Y. Zhu, "Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints," in *Languages, compilers and tools for embedded systems: software and compilers for embedded systems (LCTES/SCOPES '02)*. New York, NY, USA: ACM, 2002, pp. 213–222.

[2] E. L. Sueur and G. Heiser, "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns," in *2010 Workshop on Power Aware Computing and Systems (Hot Power'10)*. Vancouver, Canada: Usenix.

[3] P. Baptiste, M. Chrobak, and C. Dürr, "Polynomial time algorithms for minimum energy scheduling," in *Proceedings of the 15th annual European conference on Algorithms*, ser. ESA'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 136–150.

[4] A. Mahapatra, K. Anand, and D. P. Agrawal, "QoS and energy aware routing for real-time traffic in wireless sensor networks," *Comput. Commun.*, vol. 29, no. 4, Feb. 2006.

[5] Y. Won, J. Kim, and W. Jung, "Energy-aware disk scheduling for soft real-time i/o requests," *Multimedia Systems*, vol. 13, no. 5-6, 2008.

[6] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic I/O device scheduling in hard real-time systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 7, July 2003.

[7] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium*. Tucson, AZ, USA: IEEE, December 2007, pp. 239–243.

[8] H. Li and S. K. Baruah, "An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems," in *RTSS*. IEEE Computer Society, 2010, pp. 183–192.

[9] *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Dec. 2011.

[10] *IEC 61508-ed2: Functional Safety*, International Electrotechnical Commission, Apr. 2010.

[11] B. S. Emmanuel, "Microcontroller-based intelligent power management systems (ipdms) for satellite application," *ARPN Journal of Engineering and Applied Sciences*, vol. 7, no. 3, pp. 377–384, March 2012.

[12] D. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Real-Time Systems Symposium*. IEEE, 2009, pp. 291 –300.

[13] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: memory power estimation and capping," in *16th Int. Symp. on Low power electronics and design(ISLPED '10)*. New York, NY, USA: ACM, 2010, pp. 189–194.

[14] "Gentoo homepage," 2013. [Online]. Available: http://gentoo.org

[15] J. Doweck, "Inside Inotel® Core Microarchitecture and Smart Memory Access," Intel Corporation, Whitepaper, 2006. [Online]. Available: http://software.intel.com/file/18374/

[16] J. Liedtke, H. Härtig, and M. Hohmuth, "OS-controlled cache predictability for real-time systems," in *Third IEEE Real-time Technology and Applications Symposium (RTAS)*, Montreal, Canada, Jun. 1997, pp. 213–223.

[17] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Proceedings of the 2008 Euromicro Conference on Real-Time Systems*, ser. ECRTS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 147–155.

[18] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *Computers, IEEE Transactions on*, vol. 61, no. 8, 2012.

[19] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Proceedings of the 19th European conference on Algorithms*, ser. ESA'11. Springer-Verlag, 2011, pp. 555–566.

[20] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *Computers, IEEE Transactions on*, vol. 58, no. 9, pp. 1250–1258, 2009.

[21] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1-2, pp. 129–154, May 2005.

[22] R. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *Computers, IEEE Transactions on*, vol. 57, no. 9, pp. 1261–1276, 2008.

[23] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011, pp. 34–43.

[24] J.-J. Che, C.-Y. Yang, and T.-W. Kuo, "Slack reclamation for real-time task scheduling over dynamic voltage scaling multiprocessors," in *Sensor Networks, Ubiquitous, and Trustworthy Computing*, vol. 1, June 2006, p. 8 pp.

[25] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings of the 42nd annual Design Automation Conference DAC '05*, New York, NY, USA, pp. 111–116.

[26] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686–700, 2003.

[27] W. Kim, J. Kim, Y. Sang, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *In Proceedings of Design Automation and Test in Europe*, 2002, pp. 788–794.

[28] X. Fu and X. Wang, "Utilization-controlled task consolidation for power optimization in multi-core real-time systems," in *RTCSA*, 2011.

[29] M. Völp, J. Steinmetz, and M. Hähnel, "Consolidate-to-idle: the second dimension comes almost for free," in *RTAS - WIP*, April 2013.

[30] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011.

[31] C. Rusu, R. Melhem, and D. Mosse, "Mulit-version scheduling in rechargeable energy-aware real-time systems," *IEEE Journal of Embedded Computing*, no. 3, June 2004.

[32] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 13–17, Jan. 2012.

[33] "perf: Linux profiling with performance counters," 2013. [Online]. Available: https://perf.wiki.kernel.org