Real-Time Scheduling for a Virtual Simple Architecture (VISA) *

Kiran Seth, Aravindh Anantaraman, Eric Rotenberg and Frank Mueller
Departments of Computer Science/Electrical and Computer Engineering
Center for Embedded Systems Research
North Carolina State University, Raleigh, NC 27695
{ericro,fmuelle}@unity.ncsu.edu, phone: +1 (919) 513-2822 or 515-7889

ABSTRACT

Determining safe and tight upper bounds on the worst-case execution time (WCET) of hard real-time tasks running on contemporary microarchitectures is a difficult problem. Current trends in microarchitecture design have created a complexity wall: By enhancing performance through ever more complex architectural features, systems have become increasingly hard to analyze.

The objective of this work is to obviate the need to statically analyze complex processors by instead shifting the burden of guaranteeing deadlines – in part – onto the hardware. We propose a framework named Virtual Simple Architecture (VISA). The technique relies on a re-configurable processor design that supports two modes of operation, a complex mode and a simple mode. A WCET bound is statically derived for a task assuming the simple mode. At run time, the task is executed using the complex mode, and its progress is gauged to detect rarely occurring deadline misses of sub-tasks, in which case the processor switches to the simple mode and still finishes within the overall deadline.

The novelty of this paper is its generalization of VISA to cope with a *set of tasks* vs. a single task and to demonstrate that slack in a schedule can be safely and aggressively exploited by dynamic frequency/voltage scaling (DVS) for energy savings. Experiments with 7 C-lab-based tasksets show that a VISA-compliant complex pipeline consumes 12-47% less energy than an explicitly-safe simple pipeline.

1. INTRODUCTION

This work approaches timing analysis from a hardware/software co-design angle. We propose a novel microarchitectural design technique named Virtual Simple Architecture (VISA). VISA provides two processor pipeline models: (1) a complex, out-of-order, superscalar pipeline with dynamic branch prediction and (2) a simple, in-order, scalar pipeline with static branch prediction; both models support split instruction and data caches. The complex pipeline model, which may in fact include additional state-of-the-art microarchitectural features, is deemed to be unsafe in the sense that we cannot provide safe WCET bounds. The simple pipeline model, in contrast, allows us to derive safe and tight WCET bounds with existing static timing analysis tools. In practice, an existing complex pipeline can be reconfigured into a simple pipeline through gating without considerable overhead in design or die space, but this aspect is beyond the scope of this paper. Thus, the two pipeline models are really two modes of execution on a common hardware substrate, a *complex mode* and a *simple mode* [1].

2. VISA FRAMEWORK

The VISA framework works as follows. A WCET bound is derived for a task assuming the simple mode. However, at run time, the task is executed using the complex mode. The complex mode must be considered unsafe since, strictly speaking, the WCET bound does not apply to it. In practice, the complex mode is much faster, but we need to confirm this dynamically. This is achieved by dividing the task into multiple smaller sub-tasks and gauging their progress. Sub-tasks are assigned soft deadlines, called checkpoints, based on their latest allowable completion time assuming simple mode. Continued safe progress in complex mode is confirmed as long as sub-tasks meet their checkpoints. If a checkpoint is missed, the processor is reconfigured to operate in simple mode, thereby bounding execution time explicitly.

Our method is illustrated in Fig. 1-3 for a task with four sub-tasks and their actual execution times (arrows). Fig. 1 shows the conventional, conservative approach in which complexity is disabled in order to be explicitly safe. The task is executed entirely in simple mode. Dashed vertical lines in this figure indicate the WCETs of sub-tasks, which are very tight in the example.

Fig. 2 shows our *speculative* approach where the task is executed in the unsafe, complex mode. Notice that the checkpoint for sub-task 1 (chk1) lines up with the *start time* of sub-task 1 in the non-speculative case (Fig. 1); other sub-task checkpoints are similar. This leaves enough time to finish sub-task 1 plus execute sub-tasks 2, 3, and 4 in the simple mode if sub-task 1 misses its checkpoint in the complex mode. Moving chk1 any later would not be safe due to WCET bounds. Thus, extra time must be budgeted relative to the non-speculative case. While the sub-tasks complete well in advance of their checkpoints, we cannot statically prove it. Hence, substantial dynamic slack is created in the schedule. This opens up significant opportunities for lowering frequency/voltage to exploit excess slack and save power/energy, and/or scheduling additional tasks.

Fig. 3 shows a task *misprediction* for sub-task two indicated by an X at the checkpoint. In missing the checkpoint, its execution time using the complex mode is no longer boundable, so the processor switches to simple mode to explicitly bound the execution time of the unfinished sub-task and remaining sub-tasks (dashed arrows indicate simple mode). Here, we conservatively bound the residual execution time of the unfinished sub-task using its WCET since we cannot know how much of the sub-task was completed in the complex mode. The implication is that the checkpoint for a sub-task is based on the cumulative WCETs of all later sub-tasks *plus WCET of the sub-task itself*, ensuring enough time to safely complete remaining work in simple mode. The overhead is incurred whether or not a misprediction occurs, as shown in Fig. 2-3. Thus, the price for speculation is a higher worst-case utilization. But the net benefits are substantial, and overheads can be minimized by increasing the number of sub-tasks as we will show.

Previous schemes either bar the use of complex processors or disable complex features during run-time as a means for bounding WCET. The VISA framework is a radical departure in that it allows a hard real-time task to be speculatively executed on unsafe systems without sacrificing safety. Moreover, static worst-case timing analysis of the complex mode is circumvented by dynamically confirming its WCET is bounded by WCET of the simple mode. In this way, VISA

*This work was supported in part by NSF grants CCR-0208581, CCR-0310860 and CCR-0312695.

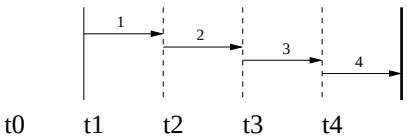

Fig. 1: Non-speculative (Conservative) Execution

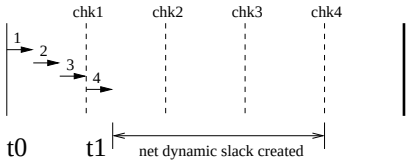

Fig. 2: Speculation w/ Complex

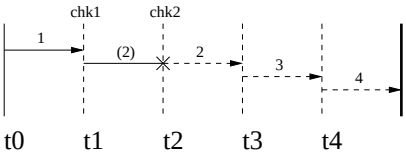

Fig. 3: Speculation w/ Complex and Misprediction (Sub-Task 2)

overcomes the complexity wall mentioned earlier.

3. TIMING SAFETY OF VISA

Progress of a task in complex mode is continuously monitored. If progress is deemed unsafe, the processor switches to the simple mode. Next, we describe a general methodology for guaranteeing safe operation in the VISA framework.

Setting Checkpoints: Timeliness of the complex mode is gauged by dividing a task into smaller sub-tasks. Each sub-task is assigned a soft deadline, called a checkpoint. If a checkpoint is missed, continued execution of the task in the complex mode is unsafe. The pipeline is reconfigured to operate in the simple mode until the end of the task. To guarantee safety in spite of a missed checkpoint, enough time must be budgeted between the checkpoint and the task's WCET (derived from static timing analysis) to

1. reconfigure the pipeline to operate in simple mode,
2. complete the sub-task i whose checkpoint was missed, in simple mode, and
3. execute the remaining sub-tasks in simple mode.

Item 1 is a fixed overhead while item 3 can be safely bounded by the worst-case execution time of the remaining sub-tasks. Regarding item 2, we do not have a tight and safe bound on the time needed to complete the unfinished portion of sub-task i, since worst-case timing analysis is done for the sub-task as a whole. Fortunately, item 2 is safely (although loosely) bounded by the WCET of the entire sub-task i. Thus, the checkpoint for sub-task i relative to the beginning of the task is as follows.

$$checkpoint_i = (WCET - overhead - \sum_{k=i}^{s} WCET_{k,f}) \quad (1)$$

Equation 1 corresponds to the remaining WCET within a task. This time between the checkpoint ($checkpoint_i$) and the WCET determined by static timing analysis includes switching overhead ($overhead$) plus the maximum time to execute the unfinished sub-task i and all subsequent sub-tasks up to s in simple mode. $WCET_{k,f}$ is the WCET of sub-task k at frequency f in simple mode.

Detecting Missed Checkpoints: A hardware cycle counter, called the watchdog counter, is used to detect missed checkpoints. The watchdog counter is memory-mapped so that it can be read and written via load and store instructions, respectively.

The watchdog counter is initialized with the number of cycles between the start of a task and its first checkpoint upon start of the first sub-task as $\lfloor checkpoint_1 \cdot f \rfloor$, where f is the processor frequency. At the beginning of each new sub-task i, the watchdog counter is incremented by the number of cycles between $checkpoint_{i-1}$ and $checkpoint_i$. Thus, sub-task i adds $\lfloor (checkpoint_i - checkpoint_{i-1}) \cdot f \rfloor$ cycles to the watchdog counter. This effectively advances the interim deadline enforced by the watchdog counter to the next checkpoint. Meanwhile, hardware autonomously decrements the watchdog counter by one every cycle. If the watchdog counter reaches zero, it means the current sub-task missed its checkpoint. In this case, an exception is raised indicating that a checkpoint was missed. If a missed-checkpoint exception occurs while executing in complex mode, then the pipeline is drained and re-configured to operate in simple mode.

Selecting a WCET for Use by Scheduler: To pad a task's WCET, any amount of padding will work (i.e., there is no correctness constraint). However, if the amount were too small, the chance of mispredicting would be high. Conversely, if it were too large, the worst-case utilization would be over-inflated with respect to no speculation. Thus, a good heuristic is needed to select a padded WCET for scheduling. We use the following heuristic: Enough padding (and no more) should be provided such that no checkpoints would be missed if the processor being checked were the simple mode itself based on the following observation. If our pipeline were running at the *exact* pace of the simple mode, then any timeliness checks would be met. Hence, if we had a faster pipeline that introduced slack, we would be allowed to slow it down, either by lowering frequency or sharing resources with other threads. The slowdown is bounded on the lower end by the speed of the original simple mode.

The above criterion is met if the padding is set equal to the maximum WCET among all sub-tasks seen in Fig. 4-5. The example assumes the timing of the processor being checked matches the timing of simple mode exactly. Therefore, the padding should be set such that all checkpoints are met. Checkpoint 3 is critical: By setting the padding to $WCET_3$ (the maximum WCET of the sub-tasks), checkpoint 3 is just met. Hence, the task's WCET provided to the scheduler is the sum of the individual sub-task WCETs plus padding equal to the maximum WCET among all sub-tasks:

$$WCET_f = \max_{1 \le k \le s} (WCET_{k,f}) + \sum_{k=1}^{s} WCET_{k,f} \quad (2)$$

Checkpoint Management in Multi-Tasking Systems: Up to this point, we have described a method for gauging progress of a task under the assumption that the task will not be interrupted by other tasks. However, in a multi-tasking environment, pre-emptions are common.

Handling pre-emptions is similar to precise handling of exceptions/interrupts. The state used to manage a task in the VISA framework is saved and later restored within its context. The following state is included in the context:

- The current pipeline mode (complex or simple).
- The contents of the watchdog counter.

When the task is resumed, the pipeline mode is restored if the previous task used a different pipeline mode. If the resumed task is in complex mode, then the watchdog counter is restored to its previous value and used to monitor further progress of the interrupted sub-task.

Accounting for Pre-emption Overheads: In multi-tasking real-time systems, the worst-case execution time of the task scheduler itself must be accounted for in schedulability analysis. Our scheduler's WCET includes selecting the next task based on EDF, applying a DVS scheduling algorithm, and saving/restoring register context in the case of pre-emptions. Safely accommodating scheduler overheads is independent of VISA, i.e., these overheads must be considered for VISA and non-VISA real-time systems alike. Nonetheless, there are several alternatives for accounting for the WCET of the scheduler.

We present an approach very convenient in two respects:

1. It implicitly accounts for the worst-case number of scheduler invocations (bounded by two per task).

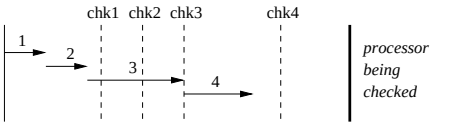

Fig. 4: Speculation w/ Complex

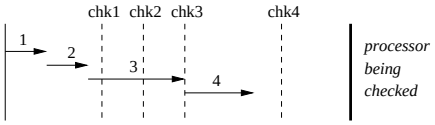

Fig. 5: Non-speculative w/ Padding Heuristic

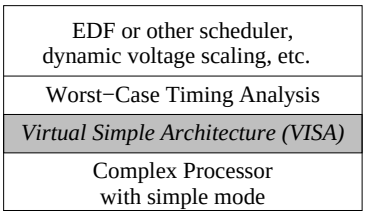

Fig. 6: Abstraction Layers

2. It enables the task scheduler to execute in complex mode without first delineating it into sub-tasks, setting interim checkpoints, etc., as is done with other tasks.

Our solution is to consider the task scheduler to be part of the first sub-task and the last sub-task of each task in the task-set. The reason the scheduler does not need to be partitioned is that, effectively, it is no longer a task, but rather a part of a sub-task in another task. To summarize, the overhead of the task scheduler is accommodated by padding the WCETs of the first and last sub-tasks (for each task in the task-set), as follows:

$$WCET_{i,f} = WCET'_{i,f} + WCET_{scheduler,f} \ for \ i = 1, s \quad (3)$$

$WCET'_{i,f}$ is the original WCET of the ith sub-task before padding it with the scheduler overhead.

VISA System Design: We have described VISA as a processor with two modes of operation. In a more general sense, VISA is an interface specification that, when adhered to by the hardware, provides a convenient abstraction for software (much like an instruction-set architecture, or ISA). The VISA is a timing specification for a hypothetical simple pipeline that is convenient for static timing analysis. Also, the complex processor embeds a simple mode of operation that directly implements the VISA specification (we are also investigating other less literal ways of ensuring VISA-compliance).

System design involves defining a VISA, and then implementing a static timing analyzer and a complex processor, the latter two compliant with the VISA. In defining the VISA, we considered the capabilities of current timing analysis tools. We also considered how a simple mode of operation is likely to be accommodated within a typical dynamically scheduled superscalar processor and, in particular, the complex processor used in this paper. The three layers — VISA, processor, and timing analyzer — are shown in Fig. 6 and are described in detail elsewhere [1].

4. VISA MULTI-TASKING SCHEDULING

In the VISA framework, the WCET for a task is derived on the basis of execution on a *virtual simple architecture*, since the simple architecture is within the capability of contemporary static timing analysis tools. At run-time, worst-case timing analysis is intentionally undermined by executing on a different, more complex architecture than was initially assumed, with the intent of accelerating execution of the task. The complex architecture is backed by a simple mode of execution that matches the timing of the virtual simple architecture used to derived the WCET of the task. The checkpoint mechanism described in Section 3 provides a means for evaluating whether or not the complex architecture is at least as timely as the virtual simple architecture. A missed checkpoint indicates the complex architecture is not making satisfactory progress, and that there is still enough time to meet the WCET constraint if the simple mode of execution is invoked immediately.

The difference between a taskset running on an explicitly-safe simple pipeline and the same taskset running on a VISA-compliant complex pipeline is that, in the latter case, the WCETs for tasks are padded to support the checking mechanism (as explained previously in Section 3). Thus, the taskset on a simple pipeline will have a lower worst-case utilization, which is revealed as static slack in the schedule. The significant advantage that the VISA-compliant complex pipeline has over the explicitly-safe simple pipeline is with respect to dynamic slack. Usually, the complex pipeline outperforms the simple pipeline by a considerable margin. Hence, checkpoints are rarely missed and, in fact, tasks finish much earlier than they would have on the simple pipeline. This creates significant dynamic slack that can be exploited using DVS, far outweighing the relatively small advantage that the simple pipeline has initially in terms of lower worst-case utilization (see Section 6).

The VISA framework also safely handles pre-emptions, as covered in Section 3. When a task is pre-empted, the contents of the watchdog counter and the current pipeline mode are saved as part of the task state. If the pre-empted task was executing in the complex mode, then when the task is later resumed, the pipeline is reconfigured to the complex mode (if the last task was in the simple mode) and the watchdog counter value is restored. Thus, we can continue gauging progress of the task in the complex mode where we left off. On the other hand, if the task was pre-empted while it was in the simple mode (due to a prior missed checkpoint), it is executed in the simple mode when it is resumed again. By resuming the task in the simple mode, it is guaranteed that the task will not exceed its WCET. Thus, any scheduling algorithm can be used unmodified within the VISA framework, as it ensures that no task will exceed its WCET that is presented to the scheduling algorithm.

Frequency Speculation: Idle time can be reclaimed in the context of VISA through dynamic frequency/voltage scaling (DVS). The novelty of DVS in the context of VISA is its potential to aggressively scale frequencies *without* having to observe WCET bounds in the *complex mode*, as explained in the following. Our sub-task checking concept provides the key benefit: (1) If we scale frequencies conservatively in the complex mode such that no checkpoints are missed, then the task will complete in complex mode but may complete prior to its latest possible completion time. (2) If we scale frequencies very aggressively in the complex mode, an early checkpoint may be missed so that the current sub-task and all subsequent ones execute in the simple mode at the original (higher) frequency to meet its deadline. This potentially results in higher energy consumption and completion prior to the latest possible point in time. The latter case may be due to actual execution times diverging from WCETs. The objective is to find the lowest frequency that does not result in missed checkpoints in order to lower energy consumption as much as possible.

In summary, any task execution with DVS on top of VISA is subject to choosing a frequency pair: a *speculative* frequency for the complex mode and, *independent* of that, a *safe* frequency for the simple mode. The safe frequency depends on the WCET bound derived for the virtual simple architecture and may be derived through arbitrary DVS real-time scheduling schemes. The speculative frequency should

be chosen as low as possible while still avoiding missed checkpoints to result in the best energy savings.

Note that the frequency-pair approach further demonstrates the transparency that VISA provides. In terms of transparency, the safe frequency is derived for our real-time system in the same way it is derived for conventional real-time systems, since it is based on the WCET abstraction. Furthermore, this safe frequency is speculatively undermined beneath the VISA layer until a checkpoint is missed, at which time the WCET constraint is explicitly enforced by switching to the simple mode and the safe frequency.

5. EXPERIMENTAL FRAMEWORK

A multi-tasking environment has been modeled using a cycle-accurate simulator using the SimpleScalar toolset, supplemented by the Wattch power models [2], which was enhanced with DVS support. The architectural simulator supports the complex and simple-fixed execution modes of VISA on GCC-compiled benchmarks and takes the scheduling code into account. We use the look-ahead DVS real-time scheduling algorithm for scheduling tasks and setting the safe frequency [4]. *Simple-fixed* uses this safe and efficient frequency. The safe frequency is disregarded for the complex mode in favor of more aggressive and unsafe scaling (frequency speculation). We combine the next task's WCET with the *average execution time* of the complex mode (at the peak frequency) to calculate the speculative frequency for the next task.

We use six different benchmarks from the C-lab real-time benchmark suite [3], shown in Table 1. For each task, we experiment with two different sub-task selections. Fine-grain sub-task selection yields twice as many (and, hence, smaller) sub-tasks than coarse-grain sub-task selection, shown in the first two columns of Table 1. We created various tasksets by combining the benchmarks above. Each taskset has three tasks, whose names and periods are indicated in Table 2. Task periods were selected to achieve a worst-case utilization of 1.0 for *complex* with coarse-grain sub-task selection.

Tab. 1: Benchmark description.

bench-mark	# sub-tasks		WCET (ms)	Padded WCET (ms)		Average execution time (ms) @ 1 GHz	
	fine-grain	coarse-grain		fine-grain	coarse-grain	*simple-fixed*	*complex*
adpcm	16	8	3.35	3.71	4.1	2.43	0.64
cnt	10	5	0.16	0.18	0.19	0.07	0.02
fft	10	5	0.59	0.66	0.73	0.36	0.06
lms	20	10	0.19	0.20	0.21	0.17	0.04
mm	20	10	2.25	2.36	2.47	2.10	0.66
srt	20	10	3.53	3.71	3.89	1.82	0.55

Tab. 2: Taskset description.

task-set	$task_1$		$task_2$		$task_3$		*simple-fixed*	*complex* fine-grain	*complex* coarse-grain
	name	P_1 (ms)	name	P_2 (ms)	name	P_3 (ms)	U_{max}	U_{max}	U_{max}
T1	lms	1.02	cnt	0.47	fft	1.81	0.84	0.92	1.0
T2	cnt	1.89	lms	0.51	srt	7.78	0.90	0.95	1.0
T3	lms	0.51	mm	4.94	cnt	1.89	0.90	0.95	1.0
T4	adpcm	8.14	cnt	0.47	lms	2.05	0.84	0.92	1.0
T5	lms	0.68	fft	2.42	mm	6.18	0.88	0.94	1.0
T6	mm	4.94	srt	12.97	lms	1.02	0.91	0.96	1.0
T7	adpcm	6.79	srt	12.97	lms	2.05	0.86	0.93	1.0
T8	adpcm	8.14	srt	9.72	cnt	1.89	0.86	0.93	1.0
T9	adpcm	10.18	srt	9.72	fft	3.62	0.85	0.93	1.0
T10	adpcm	8.14	mm	12.36	srt	12.97	0.86	0.93	1.0

6. RESULTS

Figure 7 shows the energy savings of *complex* relative to *simple-fixed*, for both coarse-grain and fine-grain sub-task selection. The complex processor saves energy, ranging from 12% (T9) to 47% (T2) energy savings. This is due to the fact that *simple-fixed* runs at an average frequency of about 475 MHz to 650 Mhz (depending on the taskset), as shown in Figure 8. On the other hand, Figure 8 shows that *complex* runs at a comparatively lower average frequency, ranging from about 200 MHz to 300 MHz (depending on the taskset). Note that the taskset with the largest difference in frequency between *complex* and *simple-fixed* (T2: 375 MHz) also yields the most energy savings (48%). In these experiments, no checkpoints were missed in the case of *complex*. This means that all tasks were executed using the complex mode and the simple mode was not used at all. The complex mode and speculative frequency are highly reliable, as we expected. But as *complex* is not provably reliable, the simple mode is needed to ensure WCET constraints are met in any potential scenario. Our experimental results with injected mispredictions are beyond the scope of this paper.

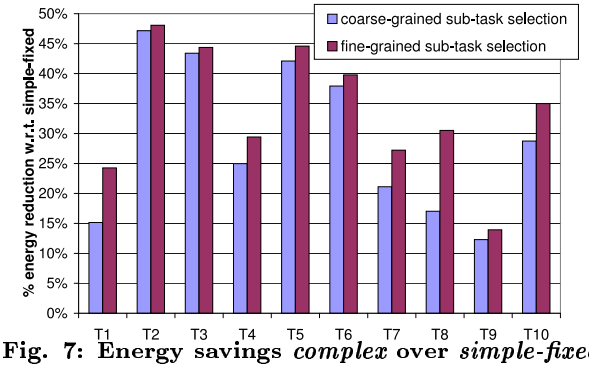

Fig. 7: Energy savings *complex* over *simple-fixed*.

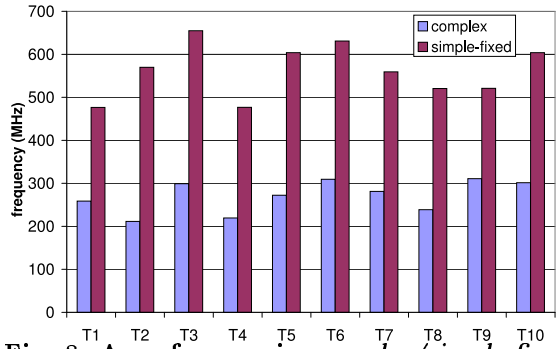

Fig. 8: Avg. frequencies *complex/simple-fixed*

7. CONCLUSION

The novelty of this paper is its generalization of VISA to scheduling a set of tasks and to demonstrate that slack in a schedule can be safely and aggressively exploited by dynamic frequency/voltage scaling (DVS) for energy savings. Experiments with 7 C-lab-based tasksets show that a VISA-compliant complex pipeline consumes 12-47% less energy than an explicitly-safe simple pipeline.

8. REFERENCES

[1] A. Anantaraman, K. Seth, K. Patil, E. Rotenberg, and F. Mueller. Virtual simple architecture (VISA): Exceeding the complexity limit in safe real-time systems. In *International Symposium on Computer Architecture*, pages 250–261, June 2003.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, pages 83–94, June 2000.

[3] C-Lab. Wcet benchmarks. Available from http://www.c-lab.de/home/en/download.html.

[4] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles*, 2001.