Energy Characterization of Embedded Real-Time Operating Systems

Andrea Acquaviva Luca Benini Bruno Riccó

DEIS - Universitá di Bologna
Bologna, ITALY 40136

Abstract

In this paper we propose a methodology to analyze the energy overhead due to the presence of an embedded operating system in a wearable device. Our objective is to determine the key parameters affecting the energy consumption of the RTOS allowing the development of more efficient OS-based power management policies. To achieve this target, we propose a characterization strategy that stimulates the RTOS both at the kernel and at the I/O driver level by analyzing various OS-related parameters. Our analisys focus in particular on the relationship between energy consumption and processor frequency characterizing the different functionalities of an RTOS, suggesting a way to develop effective OS-aware energy optimization policies based on variable voltage and frequency. Experimental results are presented for eCos, an open-source embedded OS ported and installed on a prototype of wearable device, the HP SmartBadgeIII.

1 Introduction and Motivation

Power consumption has always been a primary design constraint for most wearable devices. At the same time, these systems feature an ever increasing software and hardware complexity: a large number of heterogeneous applications must be supported while matching battery capacity. To handle complexity while ensuring modularity and fast time to market, designers are moving towards processor-based systems-on-chip (SoC) architectures instead of dedicated hardware like ASICs and DSPs. This class of architectures is well-suited to be managed by an embedded operating system, which introduces a software layer between applications and the underlaying hardware.

Clearly, in a power-constrained environment, a characterization of the energy impact of the RTOS is required, and the OS impact on power must be taken into consideration while designing the system. While a large amount of work has been done in the past to characterize performance, the energy overhead of RTOSes is not well-studied, even though it can strongly affect the effectiveness of power-aware design and power management strategies.

In this context, our work proposes a methodology to evaluate the energy overhead of embedded OSes. We use as a case study the eCos Real Time Operating System from Red Hat, and we apply our methodology to obtain a detailed energy analysis of the OS impact on a prototype wearable computer, the HP's SmartBadgeIII. In particular, we analyze key factors, like I/O data burstiness and thread switch frequency, that influence the energy overhead of operating system services and drivers.

Another important motivation for this work arises from recent developments in variable-frequency, variable-voltage processors and the related power management problems (e.g. voltage scheduling, frequency setting). These techniques require modifications of basic RTOS schedulers to account for the possibility of adjusting the voltage and speed level of the processor at run-time depending on the workload. For this reason, it is important to know how the power and performance of RTOS services and drivers change as a function of the CPU clock frequency. We performed energy characterization at different processor speeds by exploiting the frequency-setting capabilities of the StrongARM 1100, the processor core of the SmartBadge.

The remainder of the paper is organized as follows. We survey related work in Section 2. An overview of the system, both hardware and software, is provided in Section 3. In Section 4 we describe in detail how to characterize a real-time operating system from the energy viewpoint, while in section 5 we provide experimental results for the characterization framework. Section 7 concludes the paper.

2 Related Work

The problem of characterizing the energy profile of a real-time embedded operating system arises from the increasing complexity of the software architecture of modern wearable embedded systems. Moreover, operating system

energy-behavior play an important role for power optimization strategies. In the past, indeed, some researchers investigated the possibility of a cooperation between applications and OS in order to achieve an energy efficient tuning of the system resources [13][14][7].

Other authors investigated various opportunities to improve the energy efficiency of an embedded operating system. For example Vahdat et al. in [21] propose potential energy-improvements for each functionality, like interprocess communication, memory allocation, CPU scheduling, while Lebeck et al. in [11] proposed a memory paging technique that aims at putting as many memory components as possible in power-down mode. Lorch and Smith in [12] suggested heuristic techniques to put the processor in low-power states when identifying idle conditions. Benini et al. in [5] designed a workload monitoring tool for supporting effective dynamic power management policies.

In addition, a considerable amount of work has been done in the area of energy efficient scheduling for real time operating systems [8][9][10][15][16][18][20][3]. In spite of this, the need of an energy characterization of an RTOS is a relatively new concern. Indeed, researchers in the past have focused mainly on the performance of RTOSes [19][17]. The first attempt to assess the energy overhead of an embedded OS is reported in [6]. This work analyses the power profile of a commercial RTOS by running two applications and evaluating the power consumed by the operating system calls. Power analyzis are carried out on a instruction-set simulator of the Fujitsu SPARClite processor with instruction-level power models. In this work, the authors show by means of two examples that power can be saved by optimizing how the RTOS services are used by the applications.

Our work differs from [6] in many aspects. First our purpose is to fully characterize an RTOS independently from the running application. Thus, we develop a methodology ad-hoc experiment to evaluate from an energy perspective OS kernel services and drivers as a function of those that we identified as energy-sensitive factors. As an example, we tested the I/O drivers by changing data burstiness. Moreover, we perform these experiments at different frequencies, in order to enable designers to address the side effects of frequency-setting policies. As an additional difference, we performed measurements on real hardware, by an experimental set-up described later in the paper.

3 System Overview

In this section we describe the target system for our experiments. The hardware is the SmartBadgeIII, a prototype of wearable devices from Hewlett-Packard Laboratories, while the OS is eCos, a real time embedded operating system from Red Hat that we ported to the target platform.

3.1 The Hardware Platform

In this paper we deal with wearable devices. These systems are often composed by a System-on-Chip, containing a processor core, and external chips like power supply regulators, sensors, audio and video CODEC. The SmartbadgeIII, our case study, is equipped with the StrongARM 1100 processor[2], which integrates in the same chip an ARM core, memory management unit, data and instruction cache, interrupt and DMA controller, and many I/O controllers like UART, audio and LCD. The system has a small external memory footprint, 1MByte of FLASH and 1MByte of static RAM. The external audio codec on-board chip interfaces to the CPU with a multimedia communication protocol.

3.2 RTOS overview

The operating system that we analyze in our work is eCos, an open source, real-time configurable operating system, targeted to deeply embedded applications. This OS is highly modular, and it allows easy adaptation and installation on different kinds of embedded platforms while meeting memory space requirements. Indeed it has a small memory footprint, in a range from 10 to 100KBytes (depending on the configuration). In addition, eCos is compatible with Linux through the EL/IX software layer, that is a set of common system calls. ECos can be considered a real time operating system because it can be configured to provide structures able to managing alarms, timers and counters.

The overall structure of the OS consists of an hardware abstraction layer (HAL) which encapsulates all the architecture-dependent features, that interfaces to the kernel and the device driver layers. The kernel software layer is implemented in a architecture-independent way and provides the thread management and communications functions, while the devices management layer is composed by a high-level interface which is protocol dependent and a low-level architecture/platform dependent structure.

In order to install this RTOS on the SmartBadge, we modified the HAL structure to adapt it to the memory architecture and the driver layer to interface with the on-board peripherals. In particular a serial interface is needed in the early phase of the porting work in order to build a ROM monitor which acts as boot-loader and supports remote debugging.

4 Characterization Strategy

The characterization framework can be divided in three phases. In the first we analyze kernel services, like thread

management and synchronization. In the second we evaluate the energy efficiency of the I/O drivers, while in the last phase we compare the energy consumption obtained when running an application under the RTOS with the stand-alone version.

An RTOS is useful in an embedded real-time device for several reasons. One of these is that it creates a multi-threading environment and it provides time management functions like alarms and counters. These features are available through calls to kernel functions. In very complex systems with many real-time applications running simultaneously, the energy overhead imposed by these calls may become sizable.

We faced the problem of characterizing the energy cost of kernel services by evaluating the energy spent by the system call as it is, independently from the type of workload imposed by the application running. In order to carry out this measurement, we evaluated the energy consumed by each single system call.

In the first phase also we analyze how the energy cost of these calls is affected by tunable parameters. Since usually the OS is managing multiple processes we are interested in analyzing the overhead that arises when it switches between threads as a function of the switching frequency. For this reason we run two CPU-intensive threads (matrix multiplication), which maximize contention for CPU cycles and do not give any opportunity for context switching on IO-blocked processes. We compare the energy spent by running the threads in a serialized way to that spent when the two threads alternate on the CPU. This evaluation is made at different switching frequencies and at various clock speeds. Since we want to isolate the energy overhead due to context switching, we impose very small matrix dimensions, so that the cache contains both the threads worksets. Indeed, if this is not true, there is an additional energy cost due to cache-misses, which is an architecture-dependent effect.

Another important aspect of real-time operating system in an embedded context is I/O support, allowing application designers to interface with peripherals at an high abstraction level disregarding hardware details. The drawback is the complexity of the additional software that may lead to additional energy costs. This motivates the second phase of our methodology, which consist in setting up a number of benchmarks targeted to stimulate the device drivers and to find out the main factors affecting the energy consumption. Since a possible optimization framework may act on I/O buffer dimensions and processor frequency, we measure the energy variations due to different levels of data burstiness and clock speeds. In addition, we have examined the case of the device contention. As a tunable parameter here we consider the frequency switching between two competitors.

In the third phase of the characterization framework we run an application that stresses the I/O drivers and we measure energy consumption. Then we run a different version of the same application, built for running in a stand-alone way, without any RTOS support. This experiment allow us to evaluate the overall RTOS overhead. The results of our characterization are shown and commented in next section.

5 Experimental Results

Our experimental set-up consists of a hardware and a software component. The current absorbed by the Smart-Badge flows through a I/V conversion board that provides voltage values proportional to the absorbed current to a data acquisition board (DAQ). The DAQ communicates to a PC where a LABVIEW program controls the measurement framework. To obtain energy consumption values we need to measure both the current and the execution time of the programs. For this reason we used an accurate software trigger. Indeed the DAQ board allows an external signal to start and stop the measurement. We provide this signal by driving a general-purpose input/output (GPIO) pin of the StrongARM1100, which can be programmed by writing a control word on a memory-mapped CPU register. We verified on the DAQ specification that the delay introduced by the DAQ board on the trigger signal with respect to the analog inputs is 50ns, a value that is negligible in our context. Only one instruction is needed to start and stop the measurement. The LABVIEW software is responsible to combine power and time informations to give energy values.

5.1 Kernel Services

The relative average switching overhead is shown in table 1 for a fixed clock frequency value (103.2MHz). The percentage values reported in this table indicates how the energy cost increases as the switching frequency increases. We measured the energy consumption needed to perform 2 millions matrix multiplications where the matrix dimensions are 210 x 10. The reason for the small matrix size, as explained in the previous section, is to minimize the impact of cache conflicts between the two processes. The results show that increasing the context switching frequency from zero (no switching) to 10KHz does not affect the energy consumption in a significant way. Repeating the experiment at different processor clock frequencies leads similar results, with the notable exception of the minimum available processor frequency (59MHz). At this frequency the fastest context switching cannot be supported, and the system malfunctions.

From this experiment we conclude that context switch is very efficient from an energy viewpoint. However, it must be considered that we choose the benchmark in order to evaluate the overhead of the pure OS-related context switch, disregarding the cache-related energy variations that arise if

we increase the matrix dimensions, as shown in table 7 and explained later. It is also important to remember that, even though context switching does not affect much energy consumption at all processor clock frequencies, the total energy needed to carry out the computation is strongly impacted by clock frequency.

In figure 1 we reported the results of the experiment described above for different clock frequency values, by maintaining the context switching frequency fixed (maximum value). The shape of the plot shows that the energy consumption decreases as the clock speed increases up to 20%. This is due to the CPU-dominated nature of the workload (when I/O is dominant, the behavior is different [1]). This result can be easily explained by considering that the energy is the product between the average power and the total execution time. If we consider that the steady state current (and hence the power) profile obtained when running this experiment is almost flat since the processor does not access the external bus, the energy cost of thread switching is proportional to the time spent besides the not-switching case. For this reason, when the processor speed increases, the total execution time decrease, so the number of switches decrease and the total time spent due to the kernel calls decrease even if the time needed to perform a single switch decrease.

Besides context switching, we also performed an experiment in order to evaluate the overhead of each single kernel call. The results are shown in table 6. The testing parameters are shown in table 5. We made several calls for each kernel function, then in table 6 we have reported also the minumum and maximum energy values. In the table we reported experimental results related to both the minumum and the maximum available processor clock speed. In accordance to what we observed in the thread switching experiment, also in this case the energy cost is smaller at higher frequencies with a few exceptions. In fact, the reduction in execution time due to the increase of the processor speed doesn't affect the energy contribution proportional to the frequency, but reduces the the cost of the static component. This table can be used by application designers to estimate the cost of various OS calls in their code without resorting to detailed measurements.

Summarizing, the results obtained in this first phase indicate that for an application characterized by a small data working set and by low peripherical activity, it's convenient to work at an high speed.

5.2 I/O Drivers

As explained in section 4, we evaluated the energy behaviour of I/O drivers both in a single and a multi-task environments. First, we evaluate the relative energy consumption of the RTOS audio driver by sending a block of data

f_{SWT}	Energy(mJ)
0	ref.
100Hz	+0.69%
2KHz	+0.74%
5KHz	+0.80%
10KHz	+1.22%

Table 1. Thread switch experiment: Energy variation due to different switching frequencies with a fixed clock frequency (103.2Mhz)

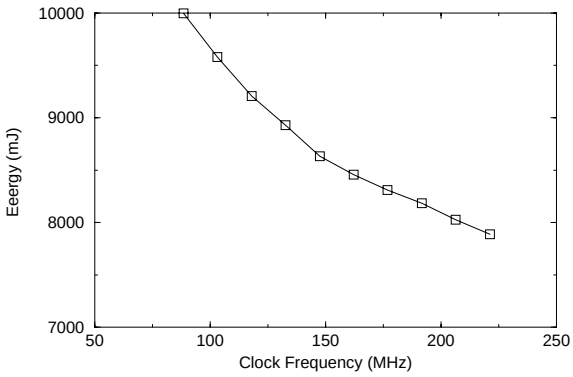

Figure 1. Thread switch experiment: Energy consumption for different clock frequencies at the maximum switching frequency

towards the audio channel. Table 2 shows the results of the experiment carried out for different levels of data burstiness and for a fixed clock frequency. When the burstiness is high, the CPU accumulates a large burst of data (with respect to the device's output buffer) before sending it to the device. We note that starting from the smaller burst size, the energy consumption decreases weakly. This is the result of two compensating contributions: the first is the energy overhead due to the higher number of calls to the driver primitives; the second, which lightly overcomes the first, is the energy saved avoiding additional idle cycles.

In effect, when the CPU sends data bursts that are large with respect to the size of the device output buffer, the CPU experience idleness when the output buffer is full. In such time intervals, it spends a non-negligible amount of energy by polling a synchronization variable. On the contrary, when the burstiness is comparable to the buffering capability of the device, idle intervals are reduced. Because a small level of burstiness allows better system responsiveness, it is convenient to organize the data in little bursts, if possible. From this results it comes out that we can do this without an additional energy cost.

We performed the same experiment by changing the

$BurstSize(bytes)$	Energy(J)
400	ref.
4000	-1.45%
40000	-1.74%

Table 2. Variation of the audio driver energy consumption due to different level of data burstiness at a fixed clock frequency

f_{SWT}	Energy(mJ)
0Hz	ref.
10Hz	+8.43%
100Hz	+10.0%
1KHz	+10.1%
10KHz	+10.5%

Table 3. Variation of the energy consumed by the audio driver in presence of device contention for different switch frequencies

clock frequency and we shown the results in Figure 2. In this case we observe a different behavior with respect to the CPU-bound application. Indeed, the energy consumption increases significantly as the clock speed increases. In effect, we notice a variation greater than 40% in the energy consumption from the minimum to the maximum clock frequency value. The reason is the energy wasted by the CPU during the idle intervals increases as the clock speed increases, because idle intervals are longer. The experimental result indicate that for an application characterized by a wide use of the external memory and the peripherals (e.g. data streaming) a lot of energy can be saved by setting the processor speed as lower as possible.

The last experiment we performed for the audio drivers is the measure of device-contention costs. We set-up two threads that alternatively access the audio driver with a certain switch frequency. In table 3 we show the results. The energy increases as the switching frequency increases, and the energy variation is higher with respect to the case of switching two CPU-bound threads.

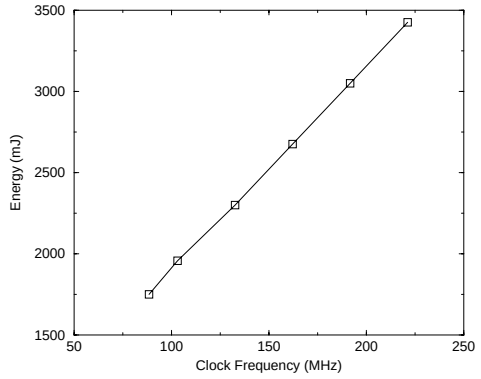

Figure 2. Energy consumption of the audio driver for different clock speeds at fixed data burstiness

	E 59MHz	E 132.7MHz	E 221.2MHz	code sz.(KB)
stdalone	7.634J	9.834J	12.430J	43.172
ecos	8.118J	13.20J	18.040J	55.540

Table 4. Comparison between the energy consumed by two version of the speech enhancer: OS based and stand-alone

5.3 Application Example: RTOS vs Stand-alone

As final experiment, we run the same application with and without the RTOS support. The application is an adaptive audio-noise canceler, which takes audio samples from the input, perform LMS filtering and sends the filtered samples towards the serial channel. In order to do the comparison, we built two versions of the filter. One version exploits the serial and audio drivers provided by eCos, the second accesses directly both the I/O channels. Since the application must play a fixed amount of audio data on the output, the execution time is the same for both versions and the energy measure is also a power measure. In Table 4 we summarize the results of the comparison. Near to the energy consumption values we show also the difference in code size. We note that the energy overhead of the OS is not due to the increased size of the application, but mainly to the driver's overhead.

From these measurements we conclude that event though some kernel services (e.g., context switching) are very efficiently implemented from an energy viewpoint, the presence of an OS (even a lightweight one, like eCos) can have significant impact on the energy consumption of an embedded systems. Our experiments show that most energy losses are due to device drivers and contention management for I/O resources. These OS routines are based on idle waiting (for performance reasons) which is very energy-inefficient.

$f_{CLK} = 59MHz$			$f_{CLK} = 221.2MHz$			
Avg Energy(μJ)	Min	Max	Avg Energy(μJ)	Min	Max	Function
15.86	12.63	38.94	3.82	3.53	12.53	Create thread
1.69	1.68	11.45	0.47	0.40	3.56	Yield thread [all suspended]
3.34	2.52	16.65	0.99	0.65	3.55	Suspend [suspended] thread
1.24	1.04	5.35	0.40	0.34	2.74	Resume thread
1.96	1.63	4.90	0.60	0.50	1.56	Set priority
0.83	0.15	3.42	0.27	0.00	1.43	Get priority
7.24	4.76	44.89	1.74	1.22	9.87	Kill [suspended] thread
1.55	1.48	4.31	0.45	0.40	1.34	Yield [no other] thread
5.01	3.42	14.72	1.33	0.83	2.54	Resume [suspended low prio] thread
1.18	1.04	2.53	0.35	0.25	0.63	Resume [runnable low prio] thread
2.71	1.93	11.44	0.75	0.56	2.44	Suspend [runnable] thread
1.55	1.48	4.31	0.45	0.40	2.75	Yield [only low prio] thread
1.33	1.04	4.01	0.37	1.22	3.34	Suspend [runnable $\rightarrow$ not runnable]
6.09	4.75	16.94	1.45	1.22	3.12	Kill [runnable] thread
5.80	3.41	27.20	5.43	0.86	7.65	Destroy [dead] thread
8.39	4.61	20.51	2.23	1.90	5.58	Destroy [runnable] thread
29.20	21.70	60.80	7.06	5.68	13.54	Resume [high priority] thread
3.69	3.42	25.42	0.86	0.80	5.35	Thread switch

Table 6. Energy consumption of kernel functions at minimum and maximum clock frequencies:59MHz and 221.2MHz

Testing parameters	Value
Clock samples	32
Thread	50
Thread switches	128
Mutexes	32
Semaphores	32
Scheduler operations	128

Table 5. Testing parameters for the experiment related to table 6

f_{SWT}	Energy(J)
0Hz	14.662
100Hz	29.330
1KHz	30.108
2KHz	31.696
5KHz	23.190
10KHz	24.651

Table 7. Energy cost of thread switching in presence of cache-related effects

6 Cache Related Effects in Thread Switching

In order to evaluate the cache-related effects that manifest themselves when two thread that are switching contend for the cache, we performed the same matrix multiplication experiment described in the previous section, but with a larger matrix size, 250x250, in such a way that the data cache does not hold the whole thread work-set. The results presented in table 7 show that in this case the cost of the switching is much higher. Indeed the energy consumption increases not only because the execution time of each thread is higher, but also because there is an increase in the average power consumption due to the memory external access related to cache-misses arising at each context switch. Even though these effects are not caused by the presence of the OS, they should be considered when developing energy-efficient scheduling strategies.

7 Conclusions

In this paper we presented a characterization methodology and a detailed analysis of energy consumption for RTOSes. We presented extensive measurements and a complete characterization for a case study, namely the eCos operating systems, and the prototype wearable computer HP SmartBadgeIII. Our work indicates that the knowledge of the energy behaviour of the RTOS is important for the effectiveness of power management policies based on voltage and frequency scaling and suggest how to improve them, by taking into cosideration the energy behaviour of the different parts of an RTOS.

References

[1] A. Acquaviva, L. Benini, B. Riccó, "An Adaptive Algorithm for Low-Power Streaming Multimedia Processing," *Design, Automation and Test in Europe Conference*, pp. 273–279, March 2001.

[2] Advanced RISC Machines Ltd., Advanced RISC Machines Architectural Reference Manual, *Prentice Hall, New York*, July 1996

[3] F. Bellosa, "Endurix: OS-Direct Throttling of Processor Activity for Dynamic Power Management," *Technical Report TR-I4-99-03, University of Erlangen*, June 1999.

[4] L. Benini, G. De Micheli, "System-Level Power Optimization: Techniques and Tools," *ACM, TODAES*, Vol. 5, No. 2, pp. 115–192, April 2000.

[5] L. Benini, A. Bogliolo, S. Cavallucci, B. Riccó, "Monitoring System Activity for OS-Directed Dynamic Power Management," *IEEE International Symposium on Low Power Electronic and Design*, pp. 185–190, Aug 1998.

[6] R. P. Dick, G. Lackshminarayana, A. Raghunathan, N. K. Jha, "Power Analysis of Embebbed Operating Systems," *Design and Automation Conference*, pp. 312–315, – 2000.

[7] M. Flinn, M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Application" *ACM SOSP*, pp. 48–63, December 1999.

[8] I. Hong, M. Potkonjak, M. B. Srivastava, "On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processors," *ICCAD*, pp. 653–656, November 1998.

[9] C. M. Krishna, Y. H. Lee, "Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low-Power in Hard Real-Time Systems," *RTAS*, May 2000.

[10] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processor," *ISLPED*, pp. 197–202, August 1998.

[11] A. Lebeck, X. Fan, H. Zeng, C. Ellis, "Power Aware Page Allocation," *ACM ASPLOS*, pp. 105–116, June 2000.

[12] J. Lorch, A. J. Smith, "Reducing Processor Power Consumption by Improving Processor Time Management in a Single-User Operating System," *MOBICOM*, pp. 143–154, 1996.

[13] B. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. R. Walker, "Agile Application-Aware Adaptation for Mobility," *ACM SOSP*, pp. 276–287, 1997.

[14] B. Noble, "System Support for Mobile, Adaptive Applications," *IEEE Personal Communications*, pp. 44–49, February 2000.

[15] T. Okuma, T. Ishihara, H. Yasuura, "Real-Time Task Scheduling for a Variable Voltage Processor," *DAC*, pp. 176–181, June 1998.

[16] Y. Shin, K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *DAC*, pp. 134–139, June 1999.

[17] L. Thiele, S. Chakraborty, M. Naedele, "Real Time Calculus for Scheduling Hard Real-Time Systems," *IEEE International Symposium on Circuits and Systems*, pp. 101–104, May 2000.

[18] I. Weiser, B. Welch, A. Demers, S. Shenker, "Scheduling for Reduced CPU Energy," *SOSDI*, pp. 13–23, November 1994.

[19] K. Weiss, T. Steckstor, W. Rosenstiel, "Performance Analysis of an RTOS by Emulation of an Embedded System," *IEEE International Workshop on Rapid System Prototyping*, pp. 146–151, 1999.

[20] F. Yao, A. Demers, S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Annual Foundation of Computer Science*, pp. 374–382, October 1995.

[21] A. Vahdat, A. Lebeck, C. Ellis, "Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency," *ACM SIGOPS European Workshop*, 2000.