EcoMobile: Energy-aware Real-time Framework for Multicore Mobile Systems[1]

B. Liu, Y. Wen, F. Liu, Y. Ahn and A. Cheng

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report Number UH-CS-11-09

October 23, 2011

Keywords: mobile system, energy-aware, real-time scheduling, multithread, multicore processor.

Abstract

In this paper, we propose a design and development plan to build an energy-aware framework for multicore mobile systems which extends Android with two features it lacks: fine-grain power management functionality and real-time support. We also provide three novel energy-aware real-time multicore scheduling algorithms to reduce both dynamic and leakage power consumptions. These algorithms address leakage power consumption more than other existing algorithms since leakage power is considered more dominant in low-power multicore architectures. Both DCS and thermal-aware task allocation are utilized to avoid overheating and unnecessary leakage.

[1]This work was supported in part by the Department of Computer Science at the University of Houston, Texas, U.S.A..

EcoMobile: Energy-aware Real-time Framework for Multicore Mobile Systems

B. Liu[1], Y. Wen, F. Liu, Y. Ahn and A. Cheng

Abstract

In this paper, we propose a design and development plan to build an energy-aware framework for multicore mobile systems which extends Android with two features it lacks: fine-grain power management functionality and real-time support. We also provide three novel energy-aware real-time multicore scheduling algorithms to reduce both dynamic and leakage power consumptions. These algorithms address leakage power consumption more than other existing algorithms since leakage power is considered more dominant in low-power multicore architectures. Both DCS and thermal-aware task allocation are utilized to avoid overheating and unnecessary leakage.

Index Terms

mobile system, energy-aware, real-time scheduling, multithread, multicore processor

I. Introduction

Mobile real-time (RT) systems have been increasingly applied in many areas such as sensor network, unmanned airplanes, unmanned vehicles, satellites, medical equipments and billions of personal mobile devices [5, 8, 13, 18]. Enabled by smaller and more complex integrated circuits (IC) [25], multicore mobile devices are prevailing for both hard real-time and soft real-time mobile applications. A monitoring system example for the hospital [13] requires support for both hard real-time applications such as sending alarms for abnormal heart beats or brain signals from the emergency room, and soft real-time applications such as sending reminders of medicine and meals to nurses. Another example is in disaster control and rescue. A robot who finds a human being needs to send SOS signals in a guaranteed interval, while it can send signals of “operating normally” after deadlines now and then without disastrous damages. Android, as the fastest growing mobile OS, does not have real-time support required by mobile RT systems today, and its various applications are getting adapted to satisfy diverse computing needs of industries, homes and personal usage. One of the reasons is because the Linux kernel, as the foundation of Android, is a time sharing system. As a result, there is no deterministic response or predictable run time of the tasks. To address this issue in Android, two features are keys to integrate. One is a low cost preemption strategy which is necessary to guarantee response time. The other is an efficient resource allocation strategy which can guarantee immediate resource access.

Besides RT support, energy efficiency is of paramount importance for mobile systems due to the increasing complexity of architectures such as multicore and hardware multithreading techniques [26]. Power density directly transforms into heat. The temperature in modern integrated circuits increases dramatically due to smaller feature size (such as the recently announced 28nm technology [25], higher packing density, and rising power consumption [6]), and therefore it is critical to tackle thermal issues in all levels of the system design. Thus, a full set of power management features exposed from hardware to the applications is necessary, such as turning on and off functional logics. Android only supports a coarse-grain power management to control the CPU, screen and keyboard with "wake locks" through the Android application framework and native Linux libraries.

In our paper, we propose a framework to extend both RT support and fine-grain power management into Android from hardware to applications. Specifically, four features are extended: power management, interrupt handling, resource allocation and performance monitoring.

Also, three novel energy-aware real-time scheduling algorithms for multicore systems are presented which utilize all the extended features. Dynamic scaling techniques, including Dynamic Voltage Scaling (DVS) and Dynamic Circuits Scaling (DCS), are used to reduce both dynamic power consumption and leakage power consumption. The design premise of DCS is to reduce the leakage power through reducing the number of running circuits. The ratio of leakage power consumption gets higher in low-power multicore architectures because of two reasons: 1) they significantly increase the number of circuits to produce

[1]This work was supported in part by the Department of Computer Science at the University of Houston, Texas, U.S.A..

the same throughput at lower clock frequencies; 2) dynamic power consumption is a cubic function of frequency (Equation (6)) while leakage power consumption is a linear function of frequency (Equation (7) (8)). At the same time, a simple thermal model (Equation (10)) is frequently used to guide task allocation by the scheduler. A wide range of studies for energy-aware real-time scheduling on multicore systems have been delivered in a decade [2, 3, 6, 7, 9, 11, 12, 15, 17, 21]. However, most work does not focus on reducing leakage power consumption. The closed work we found is [17] which power off the whole cores when necessary. Our work is different in two aspects: 1) we use our thermal model to guide the task allocation to avoid overheating; 2) we use fine-grained control on functional logics to power off unnecessary logics first, then power off the whole core.

We leverage the performance monitoring unit (PMU) on the hardware instruction level, which can help us to obtain performance statistics of each application offline. The design premise of DVS is to extend the execution time of tasks running on lower clock frequencies. Thus, the most energy-efficient DVS is Look-ahead DVS [16] which highly utilizes slacks. The more accurate the execution times of the tasks we can predict, the higher slack utilization we can get. Performance monitoring (in Section 3.2) can provide such an insight into the applications.

The rest of this paper is organized as follows: Section 2 describes our framework architecture based on Android. Section 3 describes our system design in detail, including the power management, performance monitoring, interrupt handling and resource allocation mentioned above. And Section 4 proposes our energy-aware real-time scheduling algorithms in general. Finally, Section 5 lists the task breakdowns and project timelines for implementation.

II. System architecture

Android has a layered architecture as illustrated in Figure 1. We choose to implement all our extended features in a library for the following reasons. First, it can access Linux kernel, which makes it practical to expose hardware functionalities with low cost implementation of interrupts, resource allocation and power management. Second, they are not intrusive modifications to the kernel, which makes the method portable to other systems. Finally, it provides the flexibility for developers to choose real-time scheduling or others in the same system.

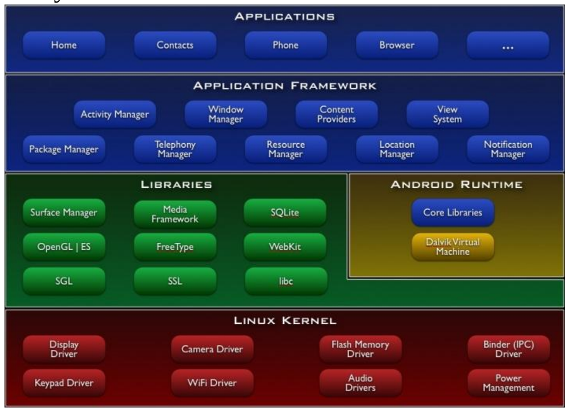

Fig. 1. Android system architecture. It describes the components in each layer of Android system.

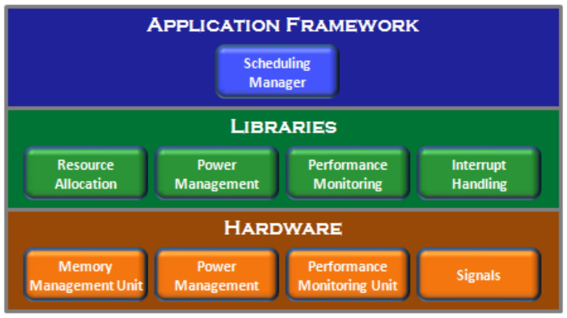

Fig. 2. EcoMobile system architecture. It describes the layers of EcoMobile system and the components in each layer.

Figure 2 describes how we fit our library into the Android architecture. The layered architecture allows us to fit in the Android architecture easily and enable each layer to be extended with new functionalities. To fulfill our requirements of low cost implementation, we need to access the hardware through the Linux kernel. To make the library portable, we make the minimum modification to the kernel and we abstract all the hardware functionalities directly exposed to the application. This abstract layer allows us to extend support to various multicore architectures without changing the applications. Portability and low cost implementation are tradeoffs in the same design. However, such a design is achieved. For example, studies about

POSIX® real-time extensions [14] show that portability can be implemented efficiently. Thus, the library is a practical choice for us.

The scheduling manager sits on the Application Framework layer. It calls the services provided by libraries and Linux kernel, including our extended features, and cooperates with services provided by other modules in this layer such as resource manager. It provides energy-aware real-time scheduling algorithms as well as a demonstration about how other uses can utilize the extended features.

III. System design

In this section, we describe each in more detail. Regarding resource allocation, our first focus is memory management due to its influence on performance. However, other resources such as high throughput I/O and other peripheral devices can be added in the similar way.

A. *Power management*

In modern architecture, power management ICs (PMIC) are often provided on system-on-a-chip devices. A PMIC may include battery management, voltage regulation, and charging functions. In our hardware abstract layer, functionalities to expose are 1) voltage regulation to change IC voltages, 2) frequency regulation to change clock speed, and 3) functionalities to power on and power off whole or partial ICs. For example, Cortex-A9 has resets to power off the whole MPCore, individual CPUs, individual functional logics, individual SIMD logics of MPE, and so on.

Regarding power-aware scheduling, there are two kinds of strategies to reduce energy consumption. One is Dynamic Voltage Scheduling (DVS) or Dynamic Voltage Frequency Scheduling (DVFS) which lowers dynamic power consumption. The other aims at reducing leakage power consumption for CPUs with turning off ICs or the whole unit.

B. *Performance monitoring*

One difficulty for real-time scheduling algorithms to meet deadlines is to accurately predict task execution time. Worst Case Execution Time (WCET) is usually used in all scheduling decisions. Performance Monitoring Unit (PMU) is increasingly used in both single core and multicore architectures to gather statistics on the operations of processor and memory system, which helps analyze bottlenecks and predict a task's execution time more accurately. Since the overhead to use PMU is high, performance monitoring functionalities are designed to use offline with multiple runs of applications.

Depending on the architecture, a different number of events and registers can be provided by the PMU. We mainly gather 5 statistics: 1) total number of floating point operations *totalFP*; 2) total number of operations *totalP*; 3) total number of misses in L1 instruction cache *L1IMiss*; 4) total number of misses in L1 data cache *L1DMiss*; 5) total number of misses in L2 cache *L2Miss*. These five numbers can describe how well the memory hierarchy works and estimate the total execution time for unknown problem sizes. For example, Equations (1 – 5) can be used to predict task execution time with curve fitting mechanisms based on multiple offline runs of a floating operation intensive application.

$$ExeTime\left(psize\right) = \left(A + B + C + D\right) \big/ CPUClock \tag{1}$$

$$A = k_0 \times \left(totalFP\left(psize\right) \big/ FpPipelineNum\right) \times FPRpeatRate \tag{2}$$

$$B = k_1 \times L1DMiss\left(psize\right) \times L1MissPenalty \tag{3}$$

$$C = k_2 \times L1IMiss\left(psize\right) \times L1\mathrm{I}\,MissPenalty \tag{4}$$

$$D = k_3 \times L2Miss\left(psize\right) \times L2MissPenalty \tag{5}$$

In the above equations, $k_0 \sim k_3$ are constants and vary for different architectures. They are obtained through curve fitting functions. $psize$ is the problem size. For example, for an FFT, problem size can be the size of one dimension of the matrix. $CPUClock$ is the frequency. $FpPipelineNum$ and $FpRpeatRate$ are platform specific features. Specifically, $FpPipelineNum$ is the number of floating point pipeline on the measured core. And $FpRpeatRate$ is the repeat rate of the pipeline, which is the number of clock cycles that occur between the issuance of one instruction and the issuance of the next instruction to the same execution unit. The miss penalty is the difference between the latencies of the current level and next level in memory hierarchy, because misses in cache incur data to be fetched from the next level in the memory hierarchy. It is to be noticed that we assume all the data is loaded into memory in the above equations because we do not consider stalls incurred in the lower level of memory hierarchy such as I/O, hard drives and tapes.

C. *Interrupt handling*

An essential characteristic of a real-time operating system is that its response time is deterministic. Interrupt handling in the current Linux kernel cannot support priority scheduling and bounded execution time. The reason is because low priority hardware interrupts can block high priority threads, i.e. Priority Inversion. Hardware and software interrupts are running in parallel. Device drivers start their interrupt service routines right after an interrupt is received, while software interrupts are handled by the kernel. Low priority hardware interrupts can steal resources from high priority threads with technologies such as Direct Memory Access (DMA). This is not an issue for Linux because it is a time sharing operating system. However, to fully support Priority scheduling, all interrupts should be served in order of their priorities in real-time systems. Thus, two steps are designed in our system: 1) all the interrupts are directed to a central control system with their priorities; and 2) threads with corresponding priorities are created to serve the interrupts. This means the scheduler schedule all the threads based on their priorities. However, one question needs to be answered first: how are we going to assign the priorities to interrupt service routines? Our answer is to assign the same priority to both its task and service routines. Related works are LynxOS [20], RTLinux [1, 22, 28], RTAI [29] and Xenomai [31].

D. Memory management

A time sharing operating system such as Linux favors fairness in the design. So is resource allocation in Linux kernel. Regarding memory management subsystem, this means it is possible that periodic tasks need to warm up caches at the beginning of every period. Also, it is not guaranteed for high priority tasks to get more resources when they need them or to keep the resources they already have. One solution is to lock the memory pages assigned to high priority tasks.

Virtual memory locking and unlocking are supported by many modern architectures. For example, mlock (), unlock (), mlockall () and maniacal () were introduced in kernel 2.6.9 and later versions. The problem to avoid with memory locks is priority inversion because low priority tasks can lock their frames in RAM, which incurs throttling for high priority tasks. Our solution for schedulable task sets is: 1) to enable memory locks for periodic tasks only; 2) to allow memory locks for the highest priority tasks through its lifetime and disable others. That means we need a central control in memory management subsystem to control locks in the above way.

IV. ENERGY-AWARE REAL-TIME MULTICORE SCHEDULING

We propose three novel energy-aware real-time scheduling algorithms in our scheduling manager based on the above features enabled through hardware to the library. We target both dynamic and leakage power consumptions. To reduce dynamic power consumption, DVS is used. To reduce leakage power consumption, two techniques are used: Dynamic Circuit Scaling (DCS) and thermal-aware task allocation.

Our DCS is used in a similar way to calculate an optimal number of cores to run as in [17]. However, our low level power management functionalities allow us to power off the functional logics when they are idle. For example, the SIMD logic of MPE is not in use when a sensor is sending and receiving signals, so we power it off but the core is running. This fine-grain control adds more complexity to our algorithms (compare with DCS in [17]).

We first address our thermal models, and then integrate them in existing real-time multicore scheduling algorithms [4]: Partitioned Earliest Deadline First (P-EDF) which is best for hard real-time applications; Clustered Earliest Deadline First (C-EDF) which is the best for soft real-time systems; and Staggered PD^2 (S-PD^2) which is among the best for both.

A. Thermal models

There are two kinds of energy-aware techniques to address two kinds of power consumptions on ICs: dynamic power consumption and leakage power consumption [3, 6, 7, 11, 12, 15, 21]. The dynamic power consumption is widely modeled as (6):

$$P_{dynamic} = C \cdot V^2 \cdot f \qquad (6)$$

where C is the collective switching capacitance, V is the power supply voltage and f is the operation frequency.

The leakage power consumption grows exponentially, and the dominant types will be sub-threshold leakage and gate leakage [7, 11, 12]. For low-power multicore architecture, leakage power can be approximated by a linear function in the operating temperature ranges of ICs with roughly 5% error [11, 15]. Thus, we use the leakage power function defined in (7):

$$P_{leakage} = \alpha \cdot \Theta(\tau) + \beta \qquad (7)$$

where α, β are constants and $\Theta(\tau)$ is a function of time τ for the operation temperature.

Thermal models derived from (6) and (7) have different forms in a wide range of studies. One of the most influential studies is about the relationship between a function unit's temperature and its frequency formalized as a super-linear function in [19], and a simplified temperature function defined in (8):

$$\Theta(\tau)=\left(\Theta(\tau_0)-a\cdot f_C^{\alpha}/b\right)e^{-b(\tau-\tau_0)}+a\cdot f_C^{\alpha}/b \qquad (8)$$

where a,b,α are curve fitting constants, and $\Theta(\tau_0)$ is the operation temperature at a start point τ_0 when the system is in an equilibrium state. The system is in the equilibrium state before all the tasks arrive, and we can derive $\Theta(\tau_0)$ from another simplified function for the operation frequency defined in (9):

$$f(\tau)=\left(b\cdot\theta_C/a\right)^{1/\alpha} \qquad (9)$$

where a,b,α are the same fitting constants as in (8).

The linear relationships of temperature and frequency expressed in (8) and (9) are simplified with fixing one or the other [19]. That is, (8) is derived with a constant frequency f_C and (9) is derived with a constant temperature θ_C . Since temperature keeps changing during run-time, we estimate it on the first derivative of (8):

$$\Theta'(\tau)=-b\cdot\left(\Theta(\tau_0)-a\cdot f_C^{\alpha}/b\right)\cdot e^{-b\cdot\Delta\tau} \qquad (10)$$

where a,b,α are the same as in (8).

B. Scheduling algorithms

As mentioned above, we integrate energy-aware techniques in three real-time multicore scheduling algorithms. Thus, we have Energy-aware P-EDF (EP-EDF), Energy-aware C-EDF (EC-EDF) and Energy-aware S-PD2 (ES-PD2). In our implementation, a cluster of cores is grouped based on the fact that they share Level 2 cache or not. Thus, the task migration is very light because only Level 1 Instruction and Data caches need to clean. A more loosely coupled grouping can be based on if the cores are sharing the same bus of memory access. In this case, no memory access is required and the task can be migrated through a bus on board such as a cache transfer bus.

Table 1 describes the steps of algorithm EC-EDF which migrates tasks before their release times if necessary. $\theta_{current}$ is initialized with the environmental temperature. $\theta_{expected}=\theta_{current}+\Delta\theta$ where $\Delta\theta$ is defined in Equation (10).

TABLE I
STEPS OF EC-EDF

```
Initial state: All cores are switched off

While task queue is not empty?
        applies DCS to calculate the number of cores $N$ to use;
        choose the smalled $N$ tuples ($\theta_{current}$, $\theta_{expected}$) among
all the cores;
        update each $\theta_{expected}$ of the chose cores with $\Delta\theta$;
        if $\theta_{expected}$ lower than threshold temperature?
                assign task $T_i$ to the chosen core;
                detect unused logics and power off;
        else
                add task $T_i$ back to queue and set this
                core to cooling status;
                Add a new core to $N$ tuples;

cooling: power off the core;
```

ES-PD^2 can be implemented in the same way. However, P-EDF has fixed mapping between tasks and cores, thus only DCS is applied.

V. Project management

We choose to build our platform on Android 3.1 which is the latest version to support the multicore architectures [23]. Although Honeycomb branch (3.0 and afterwards) targets on larger screen devices such as tablets, the reason that we believe we can build it on dual-core smartphones which can run Gingerbread and earlier branches (2.3 and earlier) is because Android is backward compatible. Thus, we select Motorola Atrix 4G with Cortex-A9 (dual-core) as our first build device [27]. Our development languages are hardware specific assembly languages, C/C++ and Java. Assembly language [24] and C/C++ are for hardware support and hardware abstract layer for performance concern [30], while Java is for library layer development.

The task breakdowns and timeline are described in Figure 3. The following are the details of each task:

- Design phase
  - Define attributes and operations of hardware unit we need in hardware abstract layer (HAL)
  - Define APIs in the library layer
  - Cortex-A9 architecture analysis and Linux kernel support
- Implementation
  - HAL and unit testing
  - Library and unit testing
  - Scheduling algorithms and unit testing
  - Demo applications and unit testing
- System Integration Test
  - From installation to each unit test case run

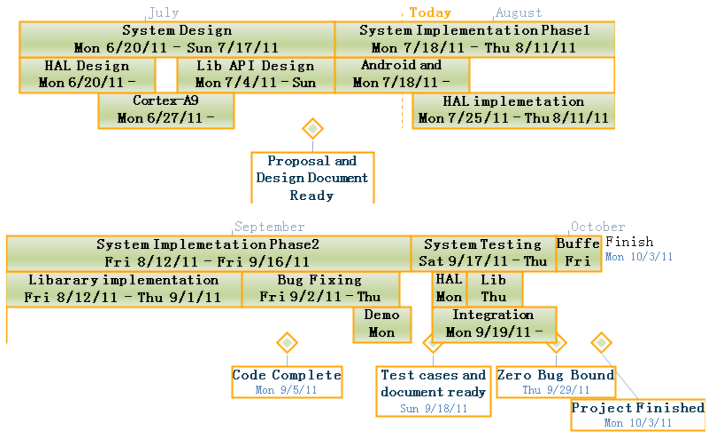

Fig. 3. The task breakdowns and timeline.

References

[1] M. Barabanov, “A Linux Based Real-Time Operating System”, Master Thesis in Computer Science, New Mexico Institute of Mining and Technology, Socorro, New Mexico, June 1, 1997.

[2] D. Bautista, J. Sahuquillo, H. Hassan, S. Petit, J. Duato, “A simple power-aware scheduling for multicore systems when running real-time applications”, IEEE International Symposium on Parallel and Distributed Processing, Miami, FL, April 2008, pp. 1- 7.

[3] S. Borkar, “Design Challenges of Technology Scaling,” IEEE Micro, July-August 1999, pp. 23-29.

[4] B. B. Brandenburg, J. M. Calandrino and J. H. Anderson, “On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study”, RTSS ’08 Proceedings of the 2008 Real-Time Systems Symposium, Washington, DC, USA 2008.

[5] T. W. Burger, “A Comparison and Analysis of Software Development Platforms for Embedded Computer-Controlled Medical Devices”, Medical Electronics Design Feature article, October 1, 2000.

[6] T. Chantem, R. P. Dick and X. S. Hu, “Temperature-Aware Scheudling and Assignment for Hard Real-Time Applications on MPSoCs,” in Proc. IEEE Design, Automation and Test in Europe (DATE), 2008, pp. 288-293.

[7] T. Chantem, R. P. Dick and X. S. Hu, “Temperature-Aware Scheudling and Assignment for Hard Real-Time Applications on MPSoCs,” in Proc. IEEE Design, Automation and Test in Europe (DATE), 2008, pp. 288-293.

[8] L. Dignan, “Mobile phone sales surge, component shortages to extend to second half of 2011”, ZDNet, US Edition, February 9, 2011.

[9] Nathan Fisher, Jian- Jia Chen, Shengquan Wang and Lothar Thiele, “Thermal- Aware Global Real- Time Scheduling on Multicore Systems,” in Proc. IEEE Real- Time and Embedded Technology and Applications Symposium, San Francisco, CA, April 2009.

[10] J. Leung, J. Whitehead, “On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks”, Performance Evaluation, Vol. 2, No. 4, pp. 237-250, 1982.

[11] Y. Liu, R. P. Dick, L. Shang and H. Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in Proc. IEEE Design, Automation and Test in Europe (DATE), Mar. 2007, pp. 204–209.
[12] Y. Liu, H. Yang, R. P. Dick, H. Wang and L. Shang, "Thermal vs Energy Optimization for DVFS-enabled Processors in Embedded Systems," in Proc. IEEE 8th International Symposium on Quality Electronic Design (ISQED), 2007.
[13] J. Moon, "Choosing the right OS for your medical device", Medical Electronics Design Feature article, March 16, 2011.
[14] K. M. Obenland, "The Use of POSIX in Real-time Systems, Assessing its Effectiveness and Performance", MITRE Technical Paper, September 2000.
[15] G. Paci, P. Marchal, F. Poletti and L. Benini, "Exploring "temperature-aware" design in low-power MPSoCs," in Proc. IEEE Design, Automation and Test in Europe (DATE), 2006, pp. 1-6.
[16] P. Pillai, K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems", Proceeding 18th ACM Symposium Operating Systems (SOSP '01), pp. 89-102, 2001.
[17] E. Seo, J. Jeong, S. Park, J. Lee, "Energy Efficient scheduling of Real-Time Tasks on Multicore Processors", IEEE Transactions on Parallel and Distributed Systems, Vol. 19, No. 11, November 2008, pp. 1540 – 1552.
[18] H. J. De La Vergne, C. Milanesi, R. Cozza, A. Gupta, CK Lu, T. H. Nguyen, A. Zimmermann, "Competitive Landscape: Mobile Devices, Worldwide, 2Q10", Gartner report, Gartner, August 11, 2010.
[19] S. Wang and R Bettati, "Delay Analysis in Temp.-Constrained Hard Real-Time Systems with General Task Arrivals," in Proc. IEEE Real-Time System Syposium (RTSS), IEEE Press, Dec. 2006, pp. 323-334.
[20] W. Weinberg, "Meeting Real-Time Performance Goals with Kernel Threads", Real-Time Magazine, RTOS Update (1) – 97q2, 1999.
[21] Y. Xie and W.-L. Hung, "Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor Systems-on-Chip Design," Journal of VLSI Signal Processing, 45(3), 2006, pp. 177-189.
[22] V. Yodaiken, "The RTLinux Manifesto", the 5th Linux Conference Proceedings, 1999.
[23] Android 3.0 (Honeycomb branch) Platform Highlights, http://developer.android.com/sdk/android-3.0-highlights.html.
[24] Cortex-A Series Programmer's Guide, Version 1.0, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388e/index.html.
[25] IBM Press, "IBM Technology Alliance Announces Availability of Advanced 28-Nanometer, Low-Power Semiconductor Technology", IBM Press room, EAST FISHKILL, N.Y., April 16, 2009.
[26] Intel® Hyper-Threading Technology (Intel® HT Technology), http://www.intel.com/technology/platform-technology/hyper-threading/.
[27] Motorola ATRIX™ 4G, Technical Specifications, http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/Mobile-Phones/ci.Motorola-ATRIX-US-EN.alt.
[28] RTLinuxFree, http://www.rtlinuxfree.com/.
[29] RTAI – the RealTime Application Interface for Linux from DIAPM, https://www.rtai.org/.
[30] The Computer Language Benchmarks Game, http://shootout.alioth.debian.org/.
[31] Xenomai: Real-Time Framework for Linux, http://www.xenomai.org/index.php/Main_Page.