Tool Supported Development of Energy-aware or Quality-aware Real-time Applications for Embedded Systems

Markus Ramsauer, Michael Coduro, Raphael Hoffmann
Chair of Computer Organization (Prof. Dr.-Ing. Werner Grass)
Faculty of Mathematics and Informatics, University of Passau, Germany
e-mail: {ramsauem,coduro,hoffmanr}@lrs.fmi.uni-passau.de

Abstract

Up-to-date mobile devices are executing an increasing number of real-time applications. Being very small and lightweight they provide only a very restricted amount of energy necessitating the developer to consider the energy consumption of an application. We present a novel approach that complements energy preserving hardware strategies and energy-aware (or quality-aware) operating system schedulers and aids the developer in implementing low energy (or high quality) applications. The application being augmented with power management code at compile time is optimized for minimal energy consumption (or maximal quality). While no real-time application is being run on the device the hardware strategies or operating system scheduler control the device speed and task execution. If a real-time application is being executed, the application itself takes control over the devices speed and its internal scheduling.

1 Introduction

Modern mobile devices are running various kinds of applications. There are office applications, e-mail clients and web browsers having no real-time constraints, and there are multi media applications like video and audio playback or recording that do have such constraints. Due to the size and weight constraints for mobile devices the energy resources are scarce and several strategies for saving energy have evolved, e.g. [1]. Hardware approaches are putting the processor into sleep mode or to a lower speed if it is idle. Software approaches (operating system schedulers) are trying to predict the processor load to choose the right speed. Although being very flexible, because they are able to handle almost all kinds of applications, they inherently have some drawbacks. Neither knowing how many nor what applications are to be executed, hardware strategies may be putting the processor to a too low speed mode or into sleep mode causing deadline misses. Operating system schedulers must not use too much resources and therefore they have to be simple but sufficiently flexible for handling a wide range of application types. Adding features such as deadline handling and data-dependency awareness to them makes them unnecessary complex for normal applications and leads to higher resource and energy demands.

Our solution complements these approaches, reliefs them from complex application models, and guarantees deadlines being met. Similar to [7] we propose building off-line optimized execution plans [3, 4] for complex applications with real-time constraints and data-dependencies, but unlike [7] who use the off-line results for their online scheduler we include them into the application. Compiling these plans into the applications enables them to handle internal scheduling and power management during their execution [2]. Scheduling and power management is done by the operating system scheduler and/or hardware strategies while the mobile device is executing non-real-time applications only. As soon as a real-time application is started it takes control of the processor and power management and schedules its internal tasks to met deadlines while saving energy. Processor control is given back to the hardware and/or operating system scheduler after the real-time application has finished. This allows building energy-aware or quality-aware applications with real-time constraints and data-dependencies that normally anticipate using runtime efficient dynamic scheduling algorithms. Figure 1 shows a conceptual example for the mixed execution of applications with and without real-time constraints.

2 Application Model

Now we describe the task model being handled by our energy-aware or quality-aware intra application real-time scheduling algorithm. The application is specified by means of a data-flow graph containing the internal (periodic) tasks that have to be executed. Tasks are representing abstract pieces of work and they are implemented by either one or

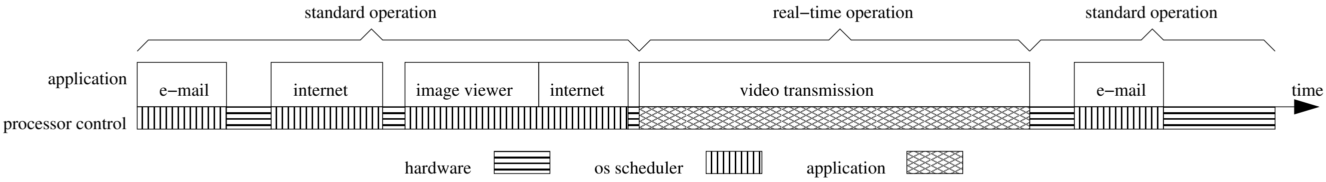

Figure 1. Processor control and operation modes

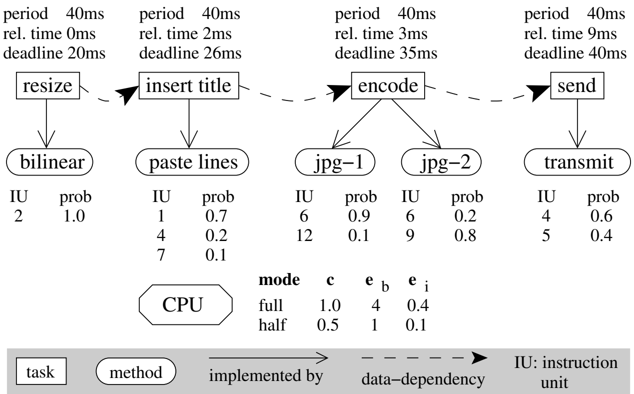

Figure 2. Live video transmission real-time application

multiple methods. Tasks can be data-dependent on other tasks and they can have release times and deadlines relative to their period beginning.

An example application is shown in figure 2. A cell phone user creates a video and sends it to a friend. The first task is to resize the captured frames to reduce their resolution. Then, text can be pasted into the frame by pressing a special button on the phone before the frame is compressed to JPG-format. There are two methods available for this task. The first one has a higher worst-case execution time but a lower expected execution time than the second one. The lower expected execution time leads to a smaller average energy consumption, and therefore it is preferable to schedule method *JPG-1* if enough time is available. At last, the encoded frame has to be sent. As we want to process 25 frames per second the deadline for the send task is 40 milliseconds. All methods except *bilinear* resizing have non-deterministic execution times, because it is not known in advance how much text the user inserts, how long compression takes (content dependent), or how much data has to be transmitted. Every time a method finishes before its worst-case execution time, it is leaving slack time to its successors that can be used to save energy by either reducing the processor's speed or by choosing a method with a lower expected energy consumption. The last part of the application model is the processor. Since the processor has multiple speed modes, we do not specify the execution times of the methods but the probability distribution of their numbers of instruction units. One instruction unit is the number of instructions that is processed at the highest speed mode within one millisecond. The example processor has two speed modes. The first speed mode processes one unit of instructions per millisecond whereas the second mode only processes half an unit of instructions. The energy consumption per millisecond is four times higher in the first mode. Therefore, fifty percent of the energy can be saved by scheduling a method at the second speed, because we need twice the time in the second mode.

3 Application Development

In practical application development there are many obstructions for using complex scheduling algorithms or for fulfilling their requirements and constraints. Some algorithms can not handle data-dependencies or slack time, or they are to difficult to configure. This section describes the application development scheme for our algorithm.

First the developer specifies the tasks building the application and then he adds data-dependencies and real-time constraints. Adding the methods being available for the tasks and the processor type completes the specification. If no execution time data for the methods is available he specifies sample data for all tasks being not data-dependent on other tasks to enable their (semi-)automatic profiling. Periodic tasks are instantiated within one hyperperiod and effective release times and deadlines are calculated. Then the offline optimizer calculates an energy optimal execution plan for the task instances and saves it as compilable code, e.g. if the application is written in C, a file containing a main() method is created. Finally, the file is (cross-)compiled and ready for execution on the target platform. Further profiling can take place during application execution to refine the execution time profiles of the methods. These Profiles can be used to build a custom tailored execution plan in a second turn around.

Figure 3 shows an energy-optimal execution plan for the live video application. The execution of the application starts by scheduling the task, method, and speed specified in the root node. As soon as a scheduled method finishes, the scheduler branches to the child node having the smallest

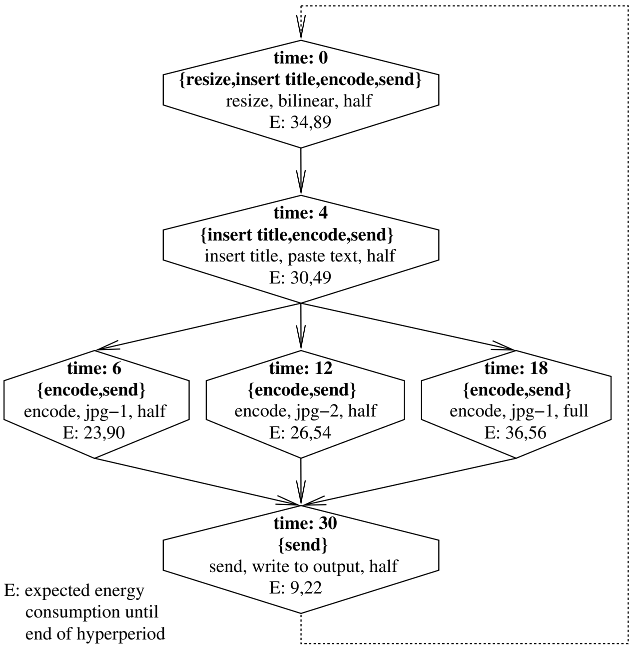

Figure 3. Energy-optimized internal application plan

time entry bigger than the time elapsed (modulo the hyperperiod).

4 Tool Support

Although the application development process is simple and straight forward there still has to be tool support for most of the steps to encourage developers to use our approach. We implemented a prototype of a tool that minimizes the effort the developer has to spend for application creation and - more important - if changes are made to the application. Figure 4 shows the tool aided application development. Depending on the target system and the profiling information already available the developer has to create three files at most. The first file contains the application model in XML format. If there are some methods that have not been profiled, test data has to be provided to initialize all methods demanding input not being produced by preceding tasks. The last file contains operating system specific code, e.g. code to allocate memory. If such a file is already available for the target operating system, there is no need to create a new one. From there on the whole application building and optimizing process can be automated. The application model is parsed (1) and initialization code and directory information (method storage) are created. Together with the test data and the operating system specific code a C file containing the application's main() method is created (2), compiled (3) and executed (4). During the first execution all methods included in the application get profiled in breadth first search sequence on the data-flow graph. Afterwards, the new profiling data is (after merging (5) with existing data) merged (6) with the application model and the energy optimization (7) algorithm is applied to it. The execution plan being created by the optimizer replaces the breadth first search sequence with a finite state automaton and is merged (8) with the operating system specific code and the initialization code to a new C file now containing an optimized main() method. This new main() method is implementing the actions that have to be taken at the internal power management and scheduling points. Compiling (9) the optimized main() method results in an optimized executable which can output additional profiling data (10). This data is used for further refinement of the main() method by repeating the optimization cycle.

All the input data and the information created by the tool is stored in files. This allows partial development and optimization of the application on a host system and the target system, because the tool can be split into parts running on either one of the systems. Complex optimization and cross compilation usually is done on workstations while the application execution and profiling takes place on the slow target device.

5 Case Study: Soccer Playing Robots

A prototype of the development tool has been implemented for soccer playing robots. The robots exclusively run the *playsoccer* application during a match while they are used as workstations otherwise. In the RoboCup [6] context our tool is used to maximize the skills (quality) of the soccer playing robot and to met deadlines, too. Since the robot's batteries last long enough for a soccer game saving energy is not important, here. Lots of sensor data has to be processed and decisions have to be drawn in real-time to accommodate to the dynamically changing environment. The robots have to do lots of computation intensive image processing and planning, and therefore thoroughly scheduling the internal tasks of the application is necessary. All methods are implemented in C/C++ and the application runs on RTAI Linux on a 400 MHz AMD K6-2 processor. The robot is interfaced by a serial connector taking commands every 100 milliseconds. To send commands as often as possible the applications deadline as well as the internal tasks' period durations have been set to 100 milliseconds.

We successfully applied our optimization algorithm to the *playsoccer* application. It has been modeled with a graphical editor that is part of our scheduling framework PASCHA [5]. The methods were profiled on the target system by running the initial main() method with test data. As we measure the methods with test data, we can not be sure that the highest measured execution time of a method is its worst-case execution time, but the optimizer assumes that

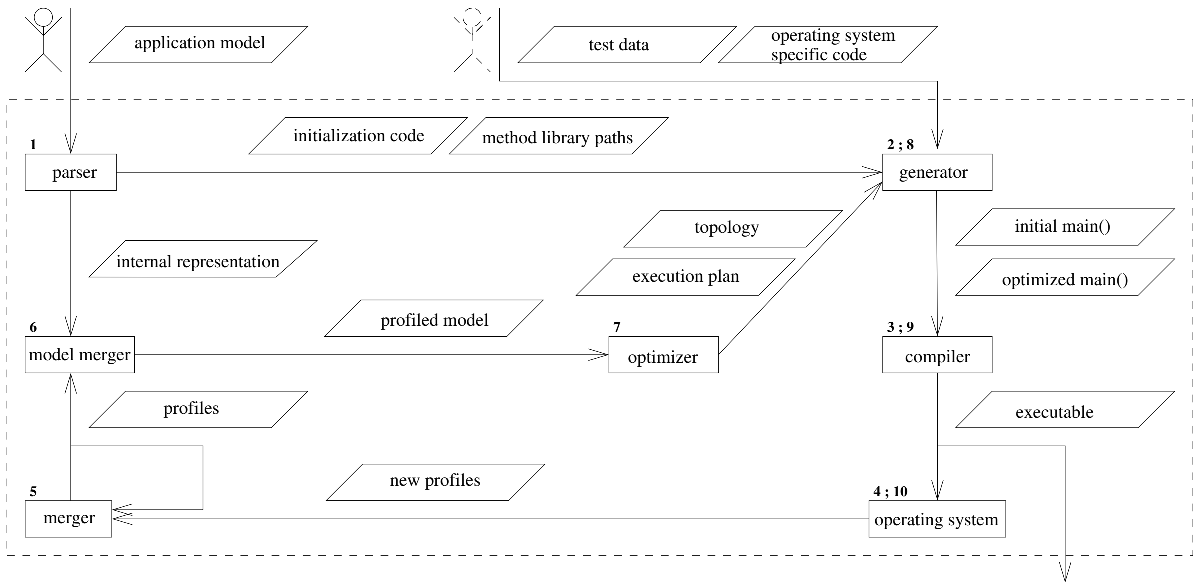

Figure 4. Tool aided application optimization

it actually is the worst-case time. In case of a longer execution time occurring during application execution, we implemented a fall back strategy in our application internal scheduler to keep the application running if no deadline has been missed. If the elapsed time is that high, that no matching node exists in the execution plan the scheduler chooses the most suitable node: the node with the highest time entry. Due to many methods not running to their specified worst-case execution time there is good chance for the application to catch up and allow scheduling to continue according to the execution plan. This is an admissible strategy for our application because no harmful damage happens to our robots if a deadline is missed. However, in other applications the developer may choose that the application should be aborted if there is the risk of a deadline being missed.

6 Conclusion and Outlook

A new approach complementing existing energy saving hardware strategies and operating system schedulers was presented. It allows for energy-aware or quality-aware execution of real-time tasks on mobile devices. Our approach directly compiles internal task scheduling and power management into the application. It exploits knowledge about the usual behavior and processing demands of the application's internal tasks during off-line optimization while it causes nearly no runtime overhead. The full flexibility of the existing operating system schedulers or hardware strategies is preserved while the ability for handling real-time constraints is added. Computational intensive real-time applications can get the full processing power provided by the device, while maximum energy savings are achieved for less intensive real-time applications. Tool supported energy-aware or quality-aware application development is proposed that minimizes the development effort and the effort for accommodating to changes. A description of a development tool was given and a case study for its prototype implementation was presented.

References

[1] Intel. Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor, White Paper, Order Number: 301170-001, March 2004, http://www.intel.com/design/intarch/papers/301174.htm.

[2] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Moss. *Power Management Points In Power-Aware Real-Time Systems; In Power Aware Computing, ed. by R. Graybill and R. Melhem, Plenum/Kluwer Publishers*. 2002.

[3] M. Ramsauer. Using Simulated Annealing for Hard Real-Time Design-to-Time Scheduling. In *ESA 2003: Proceedings of the 2003 International Conference on Embedded Systems and Applications, Las Vegas, 23. - 26. June*, 2003.

[4] M. Ramsauer. Simultaneously Exploiting Dynamic Voltage Scaling, Execution Time Variations and Multiple Methods in Energy-aware Hard Real-time Scheduling. In *Proceedings of the 17th International Conference on Architecture of Computing Systems (ARCS '04), Augsburg, Germany*, 2004.

[5] M. Ramsauer, T. Schwarzfischer, B. Sick, and W. Grass. Specification and Simulation of Scheduling Problems With the Multi-Purpose Framework PASCHA, Technical Report. University of Passau, Faculty of Mathematics and Informatics, 2002, http://lrs.fmi.uni-passau.de/ ˜ pascha/.

[6] RoboCup. http://www.robocup.org.

[7] D. Roychowdhury, I. Koren, C. Krishna, and Y.-H. Lee. A Voltage Scheduling Heuristic for Real-Time Task Graphs. In *Proceedings of the Performance and Dependability Symposium (IPDS)*, pages 741–750, June 2003.