Run-time Energy Management for Mobiles

Lodewijk T. Smit, Gerard J.M. Smit and Paul J.M. Havinga

Department of Computer Science,
University of Twente, Enschede, the Netherlands
email:smitl@cs.utwente.nl

Abstract— **Due to limited energy resources, mobile computing requires an energy-efficient architecture. The dynamic nature of a mobile environment demands an architecture that allows adapting to (quickly) changing conditions. The mobile has to adapt dynamically to new circumstances in the best suitable manner. The hardware and software architecture should be able to support such adaptability and minimize the energy consumption by making resource allocation decisions at run-time. To make these decisions effective, a tradeoff has to be made between computation, communication and initialization costs (both time and energy). This paper describes our approach to construct a model that supports taking such decisions.**

Keywords— **low power, mobile computing, quality of service, run-time energy management.**

I. Introduction

MOBILE computing is becoming big business; more and more people make use of handheld devices that includes laptops, personal digital agents (PDAs) and telephones.

Two typical characteristics required for a *mobile* are the ability (1) to adapt to changing conditions and (2) to deal efficiently with a limited amount of energy resources.

As people move around and carry mobiles with them, these movements cause a quickly changing environment for the mobile. Moreover, wireless network connections vary in quality, for instance due to interference with other users, a passing vehicle, a tunnel, a building or the weather. So, the environment of the mobile may change drastically - in short term as well as in long term - in available resources as well as in available services. Optimizing a mobile for a specific situation is not enough. A good set of parameters for one situation may be bad for another situation. One of the key differences between mobile- and fixed systems is that the former has to be able to adapt to changes in Quality of Service (QoS) resulting from mobility, rather than trying to provide hard guarantees [4].

A growing number of applications are integrated on a single mobile device. Some of these applications will result in a more frequent use of the handheld. Another observation is that more powerful applications, generate a continuous demand for more computing power. The more frequent and intensive use of the handheld raises a problem: The capacity of the battery is limited and the battery is quickly depleted. The battery is small and cannot be enlarged because of the restricted size and weight of the handheld. Battery research improves the capacity of the battery of only a few percents per year [5]. Therefore mobiles have to become more energy efficient. Low-power hardware is a first requirement for minimizing energy consumption. A lot of research has been done in this area. However, low-power hardware alone is not enough. The software has to exploit the capabilities of the low power hardware; software support is needed.

Our goal is to construct a model for this software support that allows us (1) to adapt the mobile to the changing environment and (2) to minimize energy consumption of a mobile at runtime, given a certain QoS. This model is called the Chameleon model, adapting the mobile to different situations like a Chameleon. The Chameleon model deals with the software level; optimizations on the hardware level will not be part of this model. The Chameleon model takes care for the long-term adaptations in particular. Individual sub systems should cover the short-term adaptations. This paper focuses on our approach to construct the Chameleon model.

The structure of remainder of the article will be as follows. The second section will describe the most important aspects we take into account in the Chameleon model with regard to adaptability and minimizing the energy consumption. The next section will give an outline of our approach and starting points for developing the mentioned model. The fourth section will discuss one of the case studies, which will be used to develop and to validate the model.

II. Policies

To adapt the mobile to the current environment and to minimize the energy consumption, a lot of decisions in different areas can be made, each having several trade-offs. This section addresses some of the areas that the Chameleon model should cover, and describes some of the trade-offs.

A. Partitioning and allocation

An algorithm can be implemented in software or in hardware. Choosing a specific software or hardware implementation has several consequences, because each implementation has different characteristics. In general, a hardware implementation is more energy efficient and has a better performance than a software implementation. A software implementation is more flexible; small changes and bug fixes can be implemented quickly and easily. On the other hand, small changes in a hardware implementation of an algorithm may require a new design, which can take a long time. A hybrid solution with a combination of hardware and software can offer flexibility, has a good performance and is energy efficient. This range between efficient hardware and flexible software is illustrated in fig 1. In most applications or algorithms, there are only a few kernels that are computational intensive. For example, mp3 decoding has a computational intensive discrete cosine transform (DCT) kernel, which is only a small amount of the total code. This DCT kernel may be executed in a dedicated piece of hardware. Others parts of the algorithm that are not so computational intensive can be executed in software on a general-purpose processor.

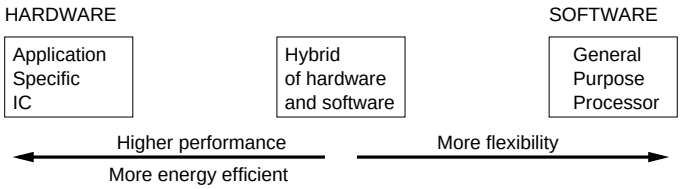

Fig. 1. Characteristics of hardware/software

Partitioning a program between hardware and software is generally known as hardware/software co-design. The difference between hardware/software co-design and our approach is the time at which the decisions are made. In hardware/software co-design a partitioning is made at design time and is static. Making partitioning decisions at design time may not always be optimal or may even be impossible, due to the dynamic nature of the environment of a mobile. Therefore, in a mobile environment a more dynamic approach is preferred. In our approach we investigate whether it is possible to make these partitioning decisions at run-time, adapting to the current circumstances. Making these decisions at run-time instead of at design time implies that there will be a limited decision time available, because making a decision also takes time and energy. Therefore a simple algorithm is required to make these partitioning decisions; extensive simulation for instance is not possible. So, making a decision at design time will likely give a better partitioning than at run-time due to the fact that a more sophisticated algorithm can be used. On the other hand, the run-time partitioning is better adapted to the current situation in contrast to a static partitioning at design time. So, a run-time partitioning may be less efficient but more effective than a partitioning made at design time.

The goal of hardware/software co-design partitioning is often to improve the performance of a system. Henkel, [7] on the other hand, presents a hardware/software partitioning approach that minimizes the energy consumption of the system. He reports power consumption reductions between 35% and 94% at the costs of relatively small additional hardware overhead. The performance is maintained or even slightly increased compared to the initial design.

Partitioning is not restricted to the handheld. Partitioning a program over the mobile and a server in the network can be attractive. Remote execution of a process gives about zero computation costs for the mobile. However, we do have energy costs for communication with the remote process. The total amount of energy needed for transmitting input data and receiving output data should be less than the amount of energy needed to execute the process on the mobile. This approach is especially suitable for computationally intensive processes with relatively small amount of data. Rudenko et al. [12] describe experiments with remote process execution to save battery energy of the mobile. They reach a power consumption reduction of up to 50% for the remote execution of relative large tasks.

Why is it not possible to have always the same partitioning? First, decisions to make a partitioning may depend on the environment of the mobile. This environment has a dynamic nature, so the trade-offs for a partitioning decision can be different. Secondly, there may be a lack of shared resources, especially if the mobile has a high load. This can for instance be the case when several algorithms use a shared resource, e.g. a

FPGA, which is configured for a specific process implementation. Finally, remote process execution may not always be possible. It's clear that a server can be used only if the server supports partitioning and it is possible to reach it through a (wireless) network connection.

B. Algorithm selection

For a specific service, the most suitable algorithm has to be selected. Example: A mobile that communicates wirelessly with a base station, may apply for different infrastructures. For example, a WaveLAN, a GSM or UMTS connection may be used. The decision space can be restricted, e.g. a connection is not available if there is no coverage. Choosing a specific infrastructure implies using the matching protocol and algorithms. Parameters for this decision are among others energy consumption, availability, costs and bandwidth.

C. Algorithm parameters

Algorithms often have specific parameters to influence their exact behavior. The choice of these parameters can have consequences for the power consumption. For example, consider the amount of redundancy used in a protocol to carry out the forward error correction. More redundancy implies transmitting more bits and transmitting bits costs energy. However, if the amount of redundancy is too small, the receiver cannot correct the errors and the whole packet has to be sent again. This will also consume a lot of energy. Therefore, depending on the current channel quality, a trade-off has to be made between the amount of energy consumed for redundancy and the amount of energy used for retransmission.

D. Policies for dynamic power management

Hardware components designed for low power have often different operation modes, e.g. off, sleep, idle and active. In different modes, the power consumption of a device varies. Hardware components that are inactive, can be set in a low power mode (e.g. sleep or off). Transitions from one mode to another cost energy and time, see figure 2. If the sleep time if too short, switching between modes can even increase the energy consumption and introduce an annoying delay. There is a trade off between the amount of energy to save in another mode and the energy costs and the delay introduced by the needed transitions. The two main questions are: (1) when to shutdown, and (2) when to wake-up [6]. The set of rules that describes the conditions that determine when a transition occurs between different modes is called a policy. Different components require different policies. An overview is given in [9].

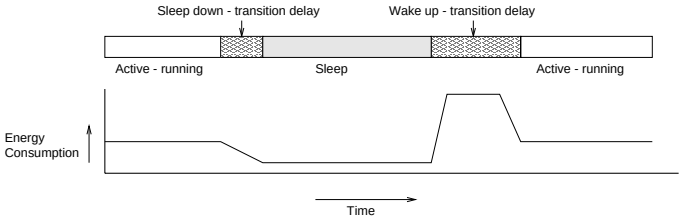

Fig. 2. Transitions between different states

The most common approaches are predictive techniques and stochastic control. Predictive techniques are based on heuristics, like a time-out. No strong optimality has been found [11] for heuristic dynamic power management (DPM) techniques. According to [2], it is more appropriate to assume a stochastic model for the system. However, stochastic techniques assume a priori knowledge of the system and its workload statistics [2]. This may be a problem for a mobile, due to the dynamic nature of the environment. For both predictive and stochastic techniques, static and adaptive approaches can be used. Static approaches are independent of the current workload, in contrast to adaptive approaches.

III. Approach

This section describes the approach we intend to use for developing the Chameleon model.

A. Properties of the decision trade-offs

The decisions mentioned in the previous section, which have to be made by the Chameleon model, require a careful trade-off between computation, communication and initialization costs. Initialization costs are not always neglectible. For example, reconfigurable hardware has to be (re)configured before it can be used. If the process time is relative short compared to the initialization time, it may not be worth to execute the process in reconfigurable hardware. Instead it may be better to do the execution in software.

Due to the dynamic nature of a mobile environment, it is not possible to determine a static solution on beforehand. The decisions have to be made at runtime. The mobile has to adapt dynamically to its changing environment.

A complicating factor is that the decisions are related to each other. Changing one parameter may

affect another. For example consider data compression before transmission over a wireless link. Despite the extra energy needed to perform the compression, energy may be saved because less data has to be sent. However, error correction for compressed data is harder. So, given a certain signal to noise ratio, there is a higher probability that a compressed package needs to be retransmitted. This may result in more frequent retransmissions, which nullifies the gain reached by the compression, or requires even more energy.

The decisions are directly influenced by the environment, including the user of the mobile. The quality of the wireless channel, the number and type of applications in use and the availability of a base station are all aspects that influence the decisions directly.

The current status of the system also influences the decisions to be made. There may exist a better total solution that is currently not feasible, because there are existing processes that cannot be stopped or migrated.

There are constraints, like performance, that have to be met. A user demands a minimum quality of service that has to be guaranteed in the current situation, if possible at all. The end result of the decisions taken, have to meet these requirements.

B. Hierarchical structure

We believe that it is important to cover the relations and interactions of different sub parts of a mobile system in a model to achieve a global optimization of the mobile's architecture. If independent optimization models are used for different parts, optimizations of the parts are the result instead of a global optimization of the system. Optimizing the sub parts may result in a non-optimal system and optimizing the global system may lead to non-optimal sub systems. If one model has to cover all the details of the sub systems, it will become too complex. Therefore, a hierarchical model is used, in which the models of the sub systems will only pass the relevant control parameters to a higher level. This gives an abstraction to handle the complexity and also takes care for local adjustable sub parts of the mobile system avoiding needless overhead.

C. Functional program description

After partitioning, parts of a program will run in hardware and parts in software. Since the partitioning is made at run-time, it is on beforehand unknown which implementation of a certain function will run. Therefore, an abstraction has to be made from the exact implementation to describe the functional behavior of the program.

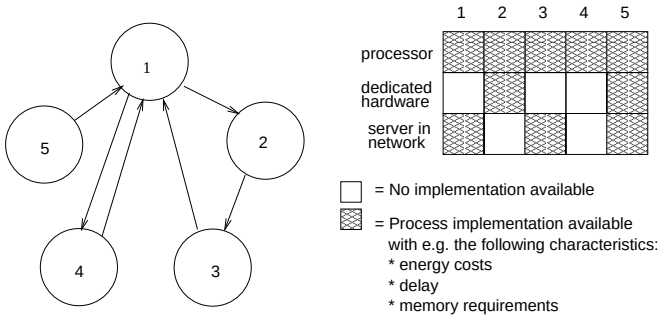

Fig. 3. Processes interconnected with channels. Different implementations of the processes are available.

To describe a program, we make use of communicating processes, depicted in the left part of figure 3. There are no dependencies between processes (like shared variables) except the order of execution (if any). A process can have multiple implementations, in hardware, in software or outside the mobile. Each process has to have at least one implementation. Each implementation has different characteristics, such as performance, delay and energy consumption. At runtime a trade-off will be made that is dependent on the current QoS parameters (e.g. quality of wireless link), the characteristics of the different implementations (e.g. energy consumption, performance), and the optimization requirements (e.g. minimize energy consumption). The target function for the optimization may differ, which may give another partitioning. For example, optimizing for performance may result in another partitioning as optimizing for minimum energy consumption. For each process the most suitable implementation for the given situation will be selected from a library, see the right of figure 3.

Communication between processes only occurs by means of channels. The channel concept is derived from the plan9 operating system [10] from Bell Labs. A channel is a first in first out (FIFO) queue for fixed-size messages. Processes use explicit communication primitives like *send* to write a message into a channel and *receive* to read a message from a channel. A channel may contain a buffer. If a channel is unbuffered, a send operation blocks until the corresponding receive operation occurs and vice versa. Channels are one-way. Channels are point-to-point, the end of a channel is always connected to exactly one process; multicasts are not supported. This mechanism provides a uniform interface for communication. Processes do not

have to know whether the process on the other side of the channel is running in hardware, in software or on a server.

IV. Case study

To determine the most relevant parameters and to develop and validate the Chameleon model, a number of case studies will be used. One of these cases is the turbo-decoding algorithm.

Turbo coding is an error-correcting algorithm for transmitting data over a wireless link. The introduction of Turbo Codes by Berrou et al. [3] in 1993 opened up new perspectives for channel coding theory. Using turbo coding, a bit error rate near the Shannon limit [13] can be obtained. The wide range of potential applications created a large interest in this coding scheme. Turbo codes are part of the third generation telephony standard UMTS [1]. For a general introduction to turbo coding, see [14].

A. Turbo (de)coding principe

The turbo encoder of 3GPP-UMTS [1], consists of two parallel 8-state Recursive Systematic Conventional (RSC) encoders and an interleaver, see figure 4.

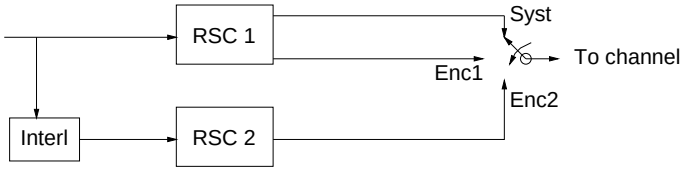

Fig. 4. Turbo encoder

An 8-state RSC encoder is a 3-bit shift register with XOR functions in the feedback loops. The output of the turbo encoder consists of the original message (systematic output *syst* from RSC 1) and the parity output *enc1* and *enc2* from both RSCs, resulting in a 1/3 code. The structure is also known under the name Parallel Concatenated Convolution Code (PCCC). Turbo codes are block codes. The block-length of a message is determined by the interleaver length. In 3GPP-UMTS this interleaving length is specified between 40 and 5114 bits. The total number of bits which are send over the channel is $3*m+12$, where m is the block-length of the message, and 12 is the number of termination bits to ensure that the encoders are returned in the initial state.

Figure 5 shows a (simplified) turbo decoder. The two most important modules are the (de)interleaver and the Soft-Input Soft-Output (SISO) module.

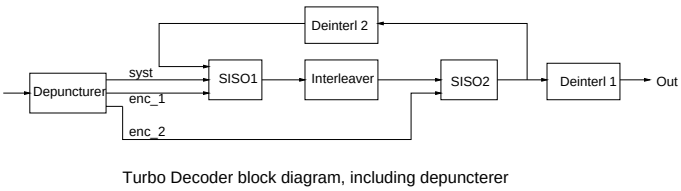

Fig. 5. Turbo decoder

B. Trade-offs

The turbo decoder highlights several important issues. Many trade-offs can be made with respect to energy consumption and adaptability. These aspects and trade-offs are described below, illustrating why the turbo decoder is an interesting case.

B.1 Partitioning

To get an idea about the energy consumption of turbo decoding, we performed measurements on the two most important modules of turbo decoding, the SISO and the interleaver. For a first coarse approximation we assume that the energy consumption is linear to the number of executed instructions. The two algorithms were compiled using a gcc 2.7.2.3 compiler. Reading the input data and initializations are excluded from the measurements. The following numbers of instructions were counted for execution of the modules with a block length of 5002 on a Pentium processor using floating-point numbers:

TABLE I
Number of executed instructions

	Number of instructions	
Optimalisation	SISO	Interleaver
full (-O3)	$2.818 * 10^3$	$80 * 10^3$
none	$9.755 * 10^3$	$402 * 10^3$

The computational complexity of the SISO module is high. For each decoded bit, about 550 instructions are executed. This module is a good candidate to implement in hardware. More research has to be done to find out whether it is better to implement the whole turbo decoder in hardware, or only the SISO module. Only implementing the SISO module in hardware will introduce additional communication overhead.

B.2 Iterations

The turbo decoder can improve the bit error rate of the message by increasing the number of iterations. The iterations can be stopped when all errors are corrected or when a predefined maximum of iterations is

reached. Eventual residual errors must be corrected in higher protocol layers, or the message has to be resent. So, a bad channel requires more iterations to correct all the errors. In most turbo decoders, the number of iterations is equal to the number of iterations needed in the worst case. This means that frequently, more iterations are made then strictly necessary. This is a waste of energy. Minimizing the number of iterations will minimize the amount of energy needed for computation. Therefore, an adaptive number of iterations is desirable. In [8] it is empirically shown that for a specific signal to noise ratio, there is an expected number of iterations for the turbo decoder.

B.3 Puncturing

For each original bit, three encoded bits are transmitted over the channel. In case of a good channel quality, this is overdone. In this situation puncturing can be applied. After transmitting a systematic bit, an even bit of the first encoder or an odd bit of the second encoder is transmitted in an alternating fashion. This results in an 1/2 rate encoder. Using an 1/2 rate instead of an 1/3 rate encoder saves energy because less bits are sent. However, possible errors are more difficult to correct.

C. Case discussion

Turbo decoding seems to be a promising case. The turbo decoding process contains a computational intensive kernel and other parts, which are less computational intensive, making a partitioning useful. Turbo decoding also has several different parameters, which are directly related to the external environment. They should be adapted - by the Chameleon model - to changes in the environment. Making a better adaptation results in lower energy consumption.

V. Summary and conclusion

In this paper we presented the first idea's to develop a model for energy-aware adaptive mobile computing. The different decision areas have been identified and described, including examples. Optimizing the whole system, instead of gluing together optimized sub systems is a key issue. Abstraction from all the details and avoiding overhead leads to a hierarchical model.

Based on the formulated approach, we will continue to develop and evaluate the model. The turbo decoding case will be analyzed further.

Acknowledgements

This research is conducted within the Chameleon project (TES.5004) supported by the PROGram for Research on Embedded Systems & Software (PROGRESS) of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the technology foundation STW. We thank Paul Heysters for commenting on draft versions of this paper.

References

[1] 3GPP TSG RAN WG1: "TS 25.212 Multiplexing and Channel Coding (FDD) V3.1.1 (1999-12)", see `http://www.3GPP.org`, 1999

[2] Benini, L. and De Micheli, G.: "System-level power optimization: techniques and tools", in *ACM Transactions on design automation of electronic systems*, volume 5, issue 2, pages 115-192, 2000

[3] Berrou C., Glavieux A. and Thitimajshima P.: "Near Shannon limit error-correcting coding and decoding: Turbo codes", in *IEEE Proceedings of ICC'93*, pages 1064-1070, May 1993.

[4] Chalmers, D. and Sloman M.: "A survey of quality of service in mobile computing environments", *IEEE Communications Surveys*, Second Quarter 1999.

[5] Cox, D.C. "Wireless personal communications: What is it?", in *IEEE personal communications*, april 1995.

[6] Havinga, P.J.M. and Smit, G.J.M.: "A review of energy-efficient wireless networking for multimedia applications", *To appear*, 2000.

[7] Henkel, J.: "A low power hardware/software partitioning for core-base embedded systems", *Proceedings of the* 36^{th} *Design Automation Conference*, 1999.

[8] Leung, O.Y., Yue C., Tsui C. and Chen, R.S.K.: "Reducing power consumption of turbo code decoder using adaptive iteration with variable supply voltage", in *Proceedings of the International Symposium on Low Power Electronics and Design*, august 1999.

[9] Lorch, J.R. and Smith, A.J.: "Software strategies for portable computer energy management", in *IEEE Personal Communications*, June 1998.

[10] Lucent Technologies, see: `www.plan9.bell-labs.com /magic/man2html/2/thread`, 2000

[11] Paleologo G.A., Benini L., Bogliolo A. and De Michili G.: "Policy optimization for dynamic power management", in *Proceedings of the* 35^{th} *Design Automation Conference*, 1998

[12] Rudenko, A., Reiher P., Popek, G.J. and Kuenning G.H.: "Saving portable computer battery power through remote process execution", in *Mobile computing and Communication Review*, Vol. 2, Number 1.

[13] Shannon, C.E.: "A mathematical theory of communications", in *Bell Systems Technical Journal*, 27((3,4)):379-423, 623-656, 1948

[14] Valenti, M.C.: "Turbo codes and iterative processing", in *Proceedings of the IEEE New Zealand Wireless Commun. Symp. '98*, Auckland New Zealand, November 1998.