Battery-aware Code Partitioning for a Text to Speech System

Anirban Lahiri, Anupam Basu, Monojit Choudhury, Srobona Mitra

Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur, INDIA 721302
alahiri, anupam, monojit @cse.iitkgp.ernet.in

Abstract

The advent of multi-core embedded processors has brought along new challenges for embedded system design. This paper presents an efficient, battery aware, code partitioning technique for a text to speech system, which is executed on a multi-core embedded processor. The system achieves significant performance improvements both in terms of execution time as well as battery lifetimes. The mentioned technique provides a new paradigm for battery aware embedded system design which can be easily extended to other applications.

1. Introduction

Applications, which were traditionally run on desktop computers, now need to be ported on low power handheld devices. The porting of such an application to a handheld embedded device is primarily hinged on two major factors. First is the execution speed of the application. Almost all applications on handheld devices need to satisfy certain hard timing constraints and if these are not met then the system is assumed to have failed. A tight upper bound on the response time of an application is one such important timing constraint. In order to meet the performance requirements dictated by these applications, modern embedded processors incorporate a number of application specific processor cores within a single package like the OMAP from Texas Instruments [12] and Nomadik from ST Microelectronics [11]. These processors enable the application code to be executed in parallel on multiple cores, and thereby producing significant performance gains. However, such an approach must take into consideration the various problems associated with the scheduling and synchronization of the various tasks of the application.

The second important factor of an application ported on a mobile embedded system is its power consumption. It is desirable that the application maintains a low power profile when executed on the embedded device. However, such mobile handheld systems typically depend on rechargeable batteries for their power. This implies that the usefulness of the device is limited by the system battery life. Recent work suggests that a low discharge profile alone does not produce an optimal battery life [4, 7]. Thus the battery life of the system provides a more accurate figure of merit than the peak value of the corresponding power profile in this case. In order to arrive at an optimal battery lifetime the various parameters of the system battery must be taken into consideration. In order to facilitate this, a number of battery models have been developed in the recent past. Such a battery model tries to emulate the working of real-life batteries E.g. [2, 9]. Many approaches found in literature have been used to accurately estimate the system battery life using a battery model [10].

Previous work has shown the advantages of functional partitioning of embedded systems for achieving both higher performances as well as reducing the system power consumption [3, 13]. However, none of these consider the effect of functional partitioning on the battery life of the system which, as mentioned previously, is becoming increasingly important. This paper describes the design flow for an Indian language text-to-speech system which first uses a functional partitioning approach and then a battery aware scheduling technique in an iterative way to arrive at an optimal design. The system serves as a paradigmatic example of a new design scheme that considers the battery lifetime as an important parameter for partitioning.

2. Motivation

It was empirically found that when a concatenative text-to-speech (TTS) engine like Shruti [8] was executed only on an embedded ARM processor, it produced unacceptable response time, especially for long sentences. Profiling the application using the Valgrind toolset [14] revealed that the TTS engine consisted of two broad

3-9810801-0-6/DATE06 © 2006 EDAA

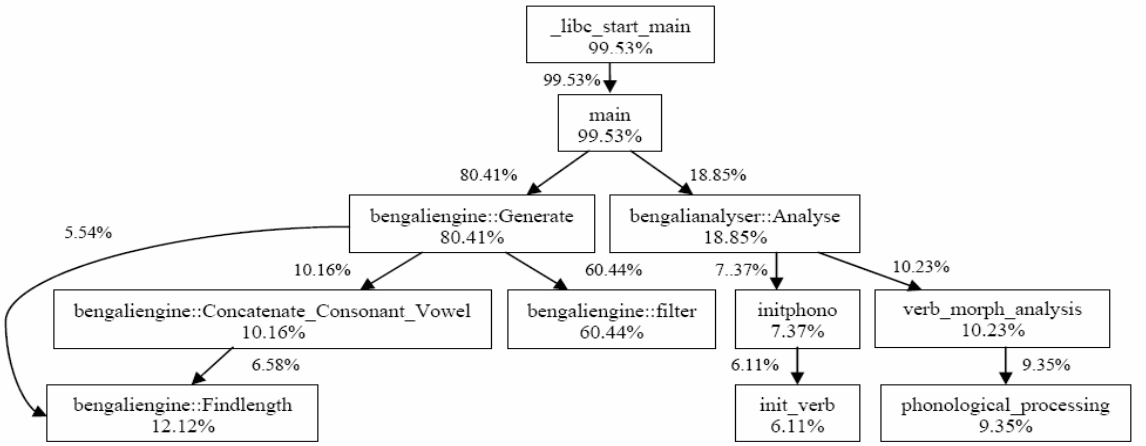

Figure 1: Call graph for the text to speech engine

categories of tasks, namely NLP (natural language processing) and DSP (digital Signal Processing) tasks, as shown in figure 1. The call graph of the application along with the percentage CPU time consumed by each task is illustrated in figure 1. It can also be observed from figure 1 that the DSP tasks taken together consume more than half of the CPU time. In order to address this issue a dual core embedded processor – TI OMAP was selected for the purpose. The architecture of the said processor and its relevance will be discussed in the following sections.

However, it was noted that a parallelized implementation of the speech application may severely compromise the battery life of the system. This indicated the need for a battery aware code partitioning scheme. The mentioned code partitioning technique has been explained in detail in subsequent sections of the paper. Another important aspect of the mentioned TTS is the memory usage of the application. The memory usage is also closely linked to the execution speed and the system battery life, hence a trade off must be reached in terms of these parameters. In order to achieve the mentioned objectives, functional partitioning, battery-aware scheduling and system performance evaluation is performed iteratively as shown in figure 2 until an optimal design is reached.

3. Partitioning for Speedup

3.1 Processor Overview

The design and development of the TTS is based on the TI OMAP5910 / OMAP1510. Both these processors are code compatible and have identical architectures. The OMAP5910 / 1510 has two processor cores – an ARM9 [1] core and a TMS320C55x DSP core. The two cores run independently while using mailboxes and shared memory for interprocessor communication and synchronization respectively.

3.2 Functional Code Partitioning

A variant of a low power functional partitioning technique [3] has been used for the purpose of code partitioning. The output of the functional partitioning is subjected to a battery aware scheduler [4] before the performance of the system is evaluated as shown in figure 2. A heuristic algorithm has been used for functional partitioning in this case. The cost function chosen for the algorithm includes the battery costs of the system using the expression suggested by [4]. For a large number of tasks a genetic algorithm is preferable as suggested by [3] for obtaining the functional partitioning. The result of the functional partitioning for the TTS system has been shown in figure 3. The DSP tasks are scheduled on the TMS320C55x core while the NLP tasks are executed on the ARM9 processor.

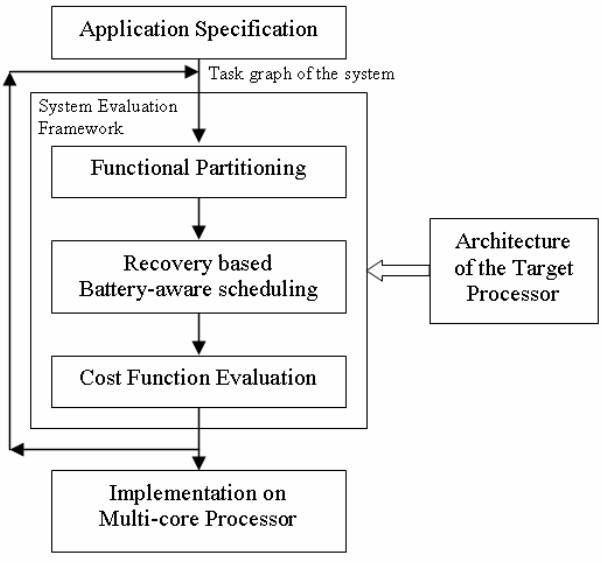

Figure 2: The system design flow

3.2 Interprocessor Communication

Primarily two interprocessor communication techniques have been employed in the design of the code partitioned TTS. A set of special purpose mailbox registers are used to pass data from one processor to another. When a processor writes some data to these mailboxes an interrupt to the other processor is automatically generated. This facilitates synchronization between the two processors. However, this mechanism is not suitable for transferring large amounts of data between the processors. A shared memory has been used to transfer bulk data between the processors. A block of data is transferred from one processor to another by first writing the data into the shared memory and then passing its address (in the shared memory), to the other processor, through the mailbox registers.

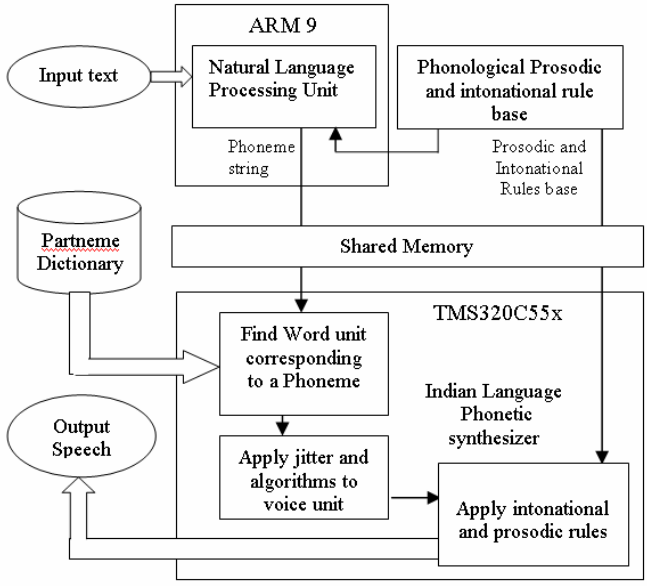

Figure 3: Functional block diagram of the TTS showing code-partitioning

3.3 Memory Access Conflicts

Memory access conflicts between the two processors have been greatly reduced by exploiting certain features of the OMAP architecture. The resources used by NLP tasks of the TTS (Eg. exception lists, lexicon), which are executed on the ARM9 core, are stored in the internal SRAM shown by the shaded rectangle in figure 4. On the other hand the relatively large (3.5MB) speech dictionary, which is accessed by the DSP processor, is stored in the external memory and is accessed through the external memory interface labeled as EMIFF in figure 4. The architecture of the memory interface traffic controller allows both these memory accesses to take place simultaneously without any conflicts, thus resulting in a greater speedup.

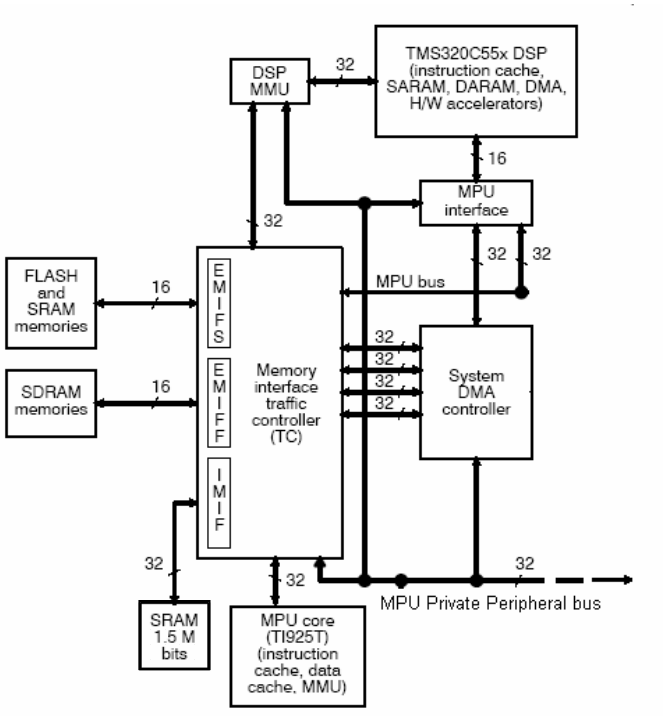

Figure 4: Relevant portions of the OMAP architecture (courtesy Texas Instruments)

4. Battery Life Optimization

4.1 Battery Basics

Rechargeable batteries are the primary source of power for mobile embedded devices. These batteries can be of various types Eg. Nickel Cadmium, Nickel Metal-Hydride, Lithium Ion, Reusable Alkaline, Lithium Polymer. However, all these batteries have some common characteristics. These include:

(a) rate capacity effect – a higher rate of discharge leads to lower battery efficiency

(b) relaxation / recovery effect – given idle periods of time the battery voltage may recover to some extent of its original voltage

These effects have been taken into consideration by number of battery models which try to emulate behavior of practical batteries [2, 9]. Such battery models can be used to accurately estimate the battery life under specific load conditions. For the purpose of the current work, a simulation framework was developed. This framework uses a modified Kinetic Battery Model [9] in order to perform battery life estimation in a cycle accurate manner. Hence it useful for evaluating various schedules obtained from the battery-aware scheduler.

4.2 Code Partitioning and Battery-aware Scheduling

A number of scheduling techniques exist for exploiting the above mentioned battery characteristics so as to maximize the battery life. It may be mentioned that these scheduling techniques are different from power aware scheduling techniques like [6]. Power aware scheduling techniques are primarily concerned with decreasing the peak power consumption of the system. Conversely, for battery aware schedulers the primary goal is to increase the system battery life. This is done by considering the various battery characteristics.

Scheduling algorithms considering recovery effect suggest that the system battery lifetime depends not only on the peak power consumption but also on the power discharge profile [4, 7]. A recovery based static scheduling approach specified in [4] has been used for scheduling the TTS tasks. This leads to a significant increase in the battery life of mobile handheld devices using the TTS.

It is apparent from figure 3, that the DSP tasks are data dependent on the NLP tasks. As previously mentioned, the system has severe limitations on memory usage and response time. Due to these reasons the TTS needs to be executed on the dual core processor in a pipelined fashion. This means that the entire input string of words is not processed as a single unit, rather it is broken up into sets of words. These clusters of words are then treated as units for processing. Thus a sequence of tasks is generated from the input character string. This stream or sequence of tasks then forms the input to the battery aware scheduler. The pipelined execution of the TTS is effective in reducing the response time or delay of the system since the speech corresponding to the first cluster of words can be sent to the audio output device while the remaining word clusters are still being processed.

The mentioned sequence of tasks are partitioned and scheduled on the two processors using a functional partitioning approach as mentioned previously. Thus the DSP tasks are scheduled on the DSP processor while the NLP tasks are scheduled on the more generic ARM processor. It is imperative that the special hardware accelerators on the DSP processor help in speeding up the DSP tasks, hence improving the system efficiency. Splitting the input character string into groups of words and then processing them also lowers the memory requirement of the system. The size of the word groups greatly influences the memory requirements and also the battery lifetime of the system.

Investigations have shown that sudden transitions in the battery discharge profile are detrimental to the battery lifetime [4]. Such unwanted transitions in a power discharge profile have been shown by thick lines in figure 5(a) given below.

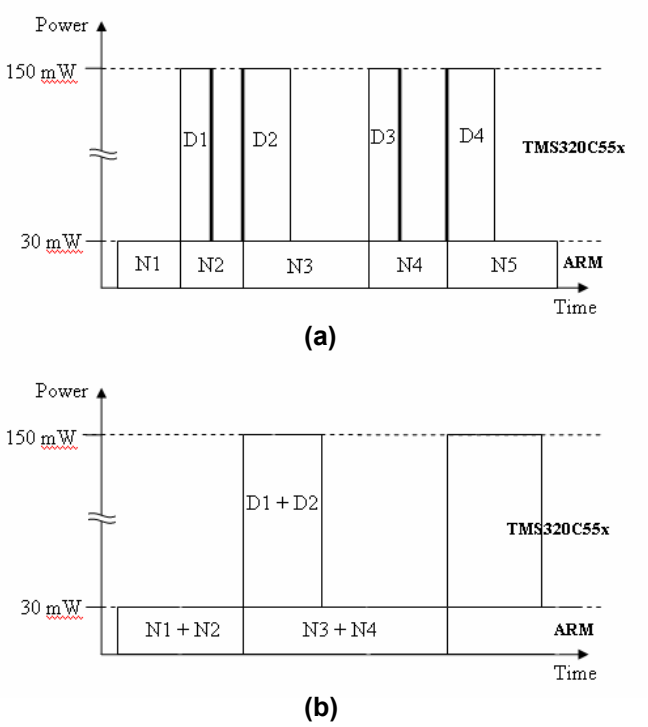

Figure 5: Power aware Gantt charts showing schedule for (a) five word (b) ten word clusters

Figure 5 shows the power aware Gantt charts corresponding to the discharge profiles for different word cluster lengths. It may be considered that in the second schedule the NLP and DSP tasks of adjacent five word clusters have been paired together. Each rectangle represents the maximum power consumption for a particular task when executed on a certain processor core. The other elements of system power consumption like those for memory, buses, etc. are assumed to be proportional to the power consumption of the processor which is using them. When not in use these system components are assumed to consume some constant power. Thus the corresponding Gantt chart will either be shifted in the y-axis or it will be suitably scaled. The exact power consumption of these components therefore need not be considered for the purpose of scheduling. It has been observed that although in the second schedule (Fig. 5(b)) the peak power consumption remains the same as that in the first one (Fig 5(a)) it achieves a significantly longer battery lifetime. It has been empirically found that in this case doubling the word group size can increase the battery lifetime by up to 6%. This is attributed to certain characteristics of battery recovery effect [4, 9] and is suitably exploited by battery-aware scheduling techniques which remove unwanted level transitions in the power discharge profile mentioned previously. The resulting schedule in this case also provides large gaps in the DSP

activity. This is allows for using dynamic voltage scaling (DVS) and dynamic power management (DPM) techniques on the OMAP processor for further improving battery life [12].

However, the length of the word clusters cannot be increased arbitrarily. Since each word cluster is treated as a single unit, tasks manipulating them would require more memory space. Also the speech data generated from the word cluster needs to be stored in the memory before it is played on the speaker. Additional buffering requirements also add to the system memory overhead. Thus the length of the word clusters is limited by memory availability of the system.

5. Results

A number of experiments were carried out with the TTS design. The primary tools used for the purpose include the Code Composer Studio and the Innovator Kit from Texas Instruments. A cycle accurate framework was also developed for the purpose of battery life estimation. The mentioned framework uses a modified kinetic battery model [9] for estimating the life of the battery when subjected to a particular discharge profile. For simplicity, battery is assumed to have a theoretical capacity of 2000mAh at 3.7V and the only the maximum discharge currents corresponding to the processors have been considered in the said experiments. A large number of simulations were carried out using this framework in order to verify the effectiveness of the recovery based scheduling algorithm [4] with respect to the current application. The findings of the mentioned experiments have been summarized in the following paragraphs.

The mentioned system achieves an average reduction of 39.36% in execution time over the corresponding non-partitioned implementation. Figure 6 shows the relative performances of the code-partitioned TTS, executed on the OMAP processor, versus the non-partitioned implementation executing on the ARM processor. It may be noted that a maximum speedup of up to 1.86 is achievable through this technique.

As previously mentioned a trade off must be reached between the memory usage of the application and the system battery life. The dependence of these two parameters on the word cluster size is illustrated by figures 7 and 8.

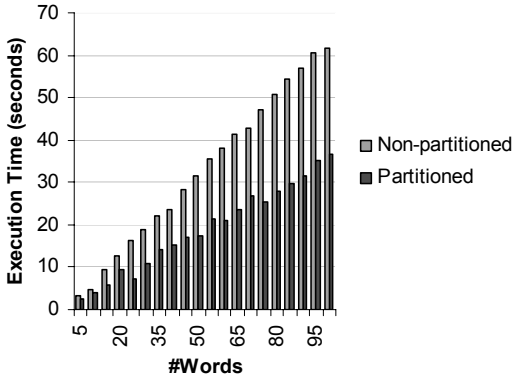

Figure 6: Comparison of non-partitioned and partitioned TTS execution times

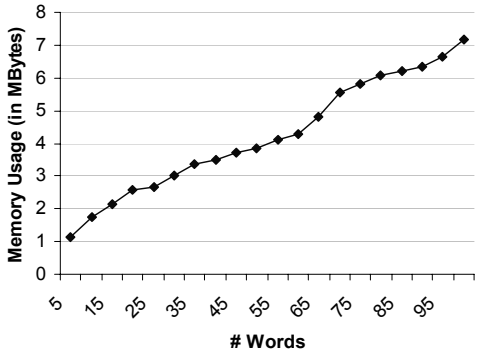

Figure 7: Graph showing the rise in memory usage with increase in word cluster lengths

The memory usage of the system shows almost a linear rise with the size of the word clusters, whereas the increase in the battery lifetime tends to saturate after a certain word cluster length. Further increase in word cluster length is not warranted as only minor improvements in battery lifetime are achieved at the cost of increased memory usage. Also it is nearly impossible to utilize the entire theoretical capacity of the battery. Thus performance of the TTS can be optimized by judiciously increasing the word cluster length. In case of highly intonated TTS engines all word cluster lengths may not be applicable; in that case, it may be necessary to consider phrases instead of just words.

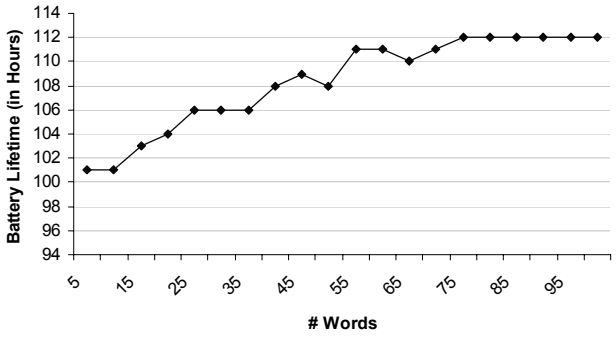

Figure 8: Graph showing increase in battery lifetime with increasing word cluster length

5. Conclusions and Future Work

A battery aware code partitioning scheme for an Indian language text-to-speech (TTS) system has been presented in this paper. The mentioned scheme achieves up to 40% reduction in the average execution time. It also extends the battery life of the system by up to 11%. The system can be further improved using DVS (Dynamic Voltage Scaling) and DPM (Dynamic Power Management) techniques. The paradigm provided by the text to speech system can be easily extended to other applications for low power, mobile embedded systems.

6. Acknowledgements

This work has been supported in part by Media Lab Asia. The authors owe special thanks to Prof. Bhargab B. Bhattacharya for providing immense help and motivation towards the work. The authors would also like to thank Mr. Arijit Mukhopadhyay, Mr. Soumyajit Dey and Tirthankar Dasgupta for helping with application profiling as well as their valuable suggestions.

7. References

[1] ARM Corp. http://www.arm.com

[2] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncio, R.Scarsi, “A Discrete-Time for High-Level Power Estimation”, Proc. Design automation and test in Europe, pp. 35-41, 2000

[3] Yunsi Fei and Niraj K. Jha, “Functional partitioning for low power distributed Systems of Systems-on-a-chip”, Proc. Intl. Conf. on VLSI Design, 2002

[4] Anirban Lahiri, Bhargab B. Bhattacharya, Anupam Basu, “Recovery-based Real-Time Static Scheduling for Battery Life Optimization”, communicated to the Intl. Conf. on VLSI Design, 2006

[5] Kanishka Lahiri, Anand Raghunathan, Sujit Dey, Debashish Panigrahi, “Battery-Driven System Design: A new frontier in low power design”, in Proc. Conf. Asia South Pacific DAC, 2002

[6] Jiong Luo and Niraj Jha, “Power-concious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems”, Proc. Intl. Conf. on Computer-Aided Desgin, pp. 357-364, 2000

[7] Jiong Luo and Niraj Jha, “Battery-Aware static scheduling for distributed real-time embedded systems”, in Proc. Design Automation Conference, pp. 444-449, June 18-22, 2001

[8] Mukhopadhyay A., Chakraborty S., Choudhury M., Lahiri A., Dey S. and Basu A.(2005). “Shruti - An Embedded Text-to-Speech System for Indian Languages”. To be published in IEE Proceedings on Software Engineering

[9] Venkat Rao, Gaurav Singhal, Anshul Kumar, Nicholas Navet, “Battery model for embedded systems”, Intl. Conf. VLSI Design, Jan, 2005

[10] Tajana Simunic, Luca Benini and Giovanni De Micheli, “Energy-Efficient Design of Battery-Powered Embedded Systems”, In Proc. ISPLED ’99

[11] ST Microelectronics. http://www.st.com

[12] Texas Instuments. http://www.ti.com

[13] Frank Vahid, Thuy Dm Le, Yu-chi Hsu, “A Comparison of Functional and Structural Partitioning”, Proc. Intl. Symposium on System Synthesis, 1996

[14] http://valgrind.org/