Vol.2011-SLDM-149 No.3
Vol.2011-EMB-20 No.3
2011/3/18

An Energy Optimization Framework for Embedded Applications

Hideki Takase,[†1,†2] Gang Zeng,[†1]
Hirotaka Kawashima,[†1] Noritoshi Atsumi,[†1]
Tomohiro Tatematsu,[†1] Lovic Gauthier,[†3]
Tohru Ishihara,[†3] Yoshitake Kobayashi,[†4]
Shunitsu Kohara,[†4] Takenori Koshiro,[†4]
Hiroyuki Tomiyama[†5] and Hiroaki Takada [†1]

This paper presents a framework for the purpose of energy optimization of the embedded systems. Our framework is synthetic, that is, multiple techniques optimize the target application simultaneously. The main technique of our approach is to utilize the trade-off between energy and performance of the embedded processor configuration. Additionally, the optimization technique about the memory allocation is employed in our framework. Our framework is also gradual, that is, the target application is optimized in a step-by-step manner. The characteristic and behavior of target application are optimized in both intra-task and inter-task level at the static time. Based on the results of the static optimization, the energy-optimal processor configuration is dynamically changed according to the behavior of the application. Moreover, we implemented the presented framework as a toolchain and a real-time operating system. The energy minimization in the average case can be achieved with keeping the real-time performance.

1. Introduction

Our life is surrounded by a large number of equipments into which computers are embedded. If the energy reduction per one equipment even a little can be achieved, it is possible to give the meaningful contribution to the whole society. Also, the amount of energy consumption might decide commodity value of the product. The energy saving has become one of the primary goals in the embedded systems. So far, a number of techniques have been proposed for the energy optimization of embedded applications. However, the man-hour for the energy optimization cannot be thrown in the development process of embedded applications due to its growing scale and complexity.

This paper presents a synthesizing and gradual framework for energy optimization of the embedded applications. The term 'synthesizing' means that our framework includes multiple energy optimization techniques. It is advisable to apply as many techniques as possible for the energy reduction. Besides, combining individual elemental techniques will achieve a synergistic effect on the energy consumption. The term 'gradual' means that the whole optimization process is split into three phases, *i.e.*, the intra-task, the inter-task, and the runtime optimization phases. The static optimization obtains the characteristic and behavior of target application at both the intra-task and inter-task level. Then, the runtime optimization performs with the results of the static optimization. The main idea of our approach is to utilize the trade-off between energy and performance of the embedded processor. It aims the energy minimization in the average case while guaranteeing task deadline constraints. It should be noted that though our approach employs application-dependent optimization, it is widely applicable in many embedded applications.

An optimization toolchain at static phase and a real-time operating system (RTOS) for the runtime optimization are implemented in this work. In the embedded domains, there is a commonly-held view that the characteristic and runtime behavior of the application are known at the design phase. These are enough obtained by our analytic and optimization toolchain at static phase. Then, the RTOS utilizes the acquired analysis results for the purpose of runtime optimization. Our toolchain will contribute the reduction of incidence in the man-hour to the embedded application's developers since the energy optimization process is designed to be automatically performed.

The rest of this paper is organized as follows. Section 2 presents our objective and overview of the framework. From Section 3 to Section 5, the respective phases of the energy optimization framework are described in the order of processing.

†1 Nagoya University
†2 The Japan Society for the Promotion of Science
†3 Kyushu University
†4 Toshiba Corporation
†5 Ritsumeikan University

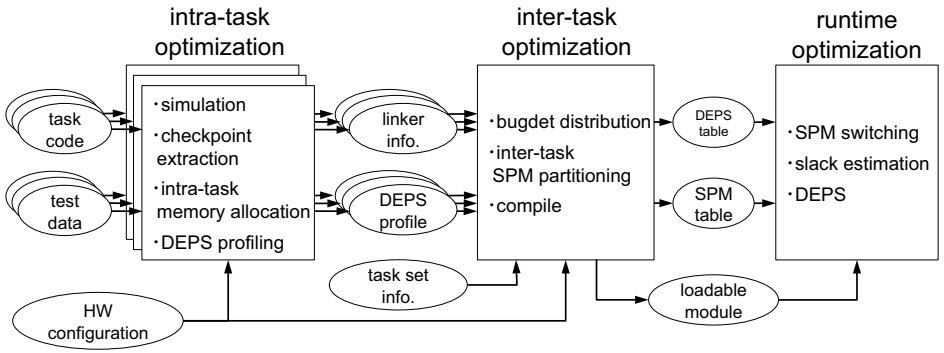

Fig. 1 The overview of the energy optimization framework.

Section 6 presents the implementation of the prototype of our framework. Finally, Section 7 concludes this paper.

2. Overview of the Energy Optimization Framework

2.1 Objective and Target Systems

Our objective is to minimize the energy consumption in the average case of the embedded application. The framework presented in this paper targets the hard real-time embedded systems where the high responsiveness is indispensable to execute its processing exactly. In the real-time systems, one of the most important things is that all tasks must be completed within their deadline. Therefore, our framework takes a policy for guaranteeing the task deadline constraint in the optimization.

In the target systems, a set of independent periodic tasks or sporadic tasks (aperiodic tasks with minimal inter-arrival separation) is assumed to constitute the application. Also, we assumed the environments where tasks are scheduled according to the priority based preemptive scheduling with static priority assignment to ensure the real-time performance.

2.2 Workflow

Fig. 1 indicates the workflow of our framework. It consists of three phases. The first is an intra-task optimization phase. Characteristic of each task is analyzed with execution traces obtained by the instruction-set simulation. The second is an inter-task optimization phase. Based on the results obtained at the former phase, an application level characteristic and behavior are analyzed and optimized. each task is assigned to a processor and execution time budget is distributed to a task. The last is a runtime optimization phase. The runtime optimization achieves maximal energy savings with two management tables generated at previous phase. This runtime mechanism is designed as the function of a RTOS.

The framework treats source code of task and input data set for the task with weight as pieces of input information. So that the average case energy consumption becomes minimum, a frequently executed input data set has to have a large weight. It should be noted that the characteristics of target application are enough obtained at static time in the embedded domain. We use an analytic approach by using a number of execution traces obtained by the simulation. The reason why we use the trace analysis is that execution traces includes the information about the characteristics and realistic behavior of target application. Moreover, execution traces are easily acquired since the worst case execution time analysis or the functional test is generally performed by a set of representative test data at development time. These obtained results are fully exploited at runtime phase for the purpose of energy optimization. Since the optimization is performed based on the execution traces of the target application, our approach is widely applicable to many embedded applications.

2.3 Elemental Techniques

As a key technique of our framework, dynamic energy and performance scaling (DEPS) technique has been proposed in 1), 2). The DEPS is a generalized technique of commonly used DVFS (Dynamic Voltage and Frequency Scaling). The rationale behind DEPS is the existing trade-off between performance and energy by selecting different processor configurations. In addition to the voltage and frequency of the processor, any reconfigurable hardware mechanisms that can trade-off performance for energy savings are considered in the DEPS.

In general, the performance requirements of application are different from phase to phase. The DEPS tries to set the processor configuration at runtime to the lowest energy consumption subject to the performance constraint. A main issue of DEPS is to determine when and what processor configuration should be

used. To cope with it, we developed several software tools for fully exploiting the performance/energy characteristics of givens applications at design time, and an energy-aware RTOS for processor reconfiguration at run time. To evaluate and validate the DEPS, we have developed a multi-performance processor (MPP)[3),4)]. The MPP has two processing elements with different voltage and frequency, and they can be dynamically switched between them rapidly (within 1 us). Moreover, instruction cache of MPP is resizable by changing its associativity.

Additional technique to be synthesized to the framework is the optimization of the memory allocation. We have proposed the memory allocation techniques for the scratch-pad memory (SPM)[5)–8)]. SPM is a fast, tiny, and energy-efficient on-chip SRAM compared with the cache memory. The basic idea of these techniques is that concentrated allocation of a frequently accessed code and data to the SPM will bring energy reduction of the embedded systems. The technique of 5) can arrange the code and static data placement at the single task level. The technique of 6) is able to apply to the stack data. A dedicated sequence of instructions is inserted to the source code to control the value of stack pointer for allocating a part of stack frame to SPM. Moreover, efficient SPM allocation techniques for multi task environments were proposed in 7), 8). The space of SPM is spatially or temporarily partitioned to the tasks for the purpose of energy minimization.

3. Intra-task Optimization Phase

Fig. 2 indicates the workflow of the intra-task optimization phase. The pieces of output information in this phase are the modified task code, the information about intra-task level memory allocation, and a DEPS profile. This section describes the processing of the intra-task phase in the order of step.

3.1 Simulation

At first step of the intra-task phase, a number of execution traces are obtained by the simulation of target application with various kinds of input data set. It is advisable that the number of obtained execution traces is expected to be larger. Additionally, each input data set is weighted by its appearance frequency. The different execution paths are generated by using the different input data set. These can improve the effectiveness of optimization for the average case energy consumption since an importance of execution path is related to the appearance

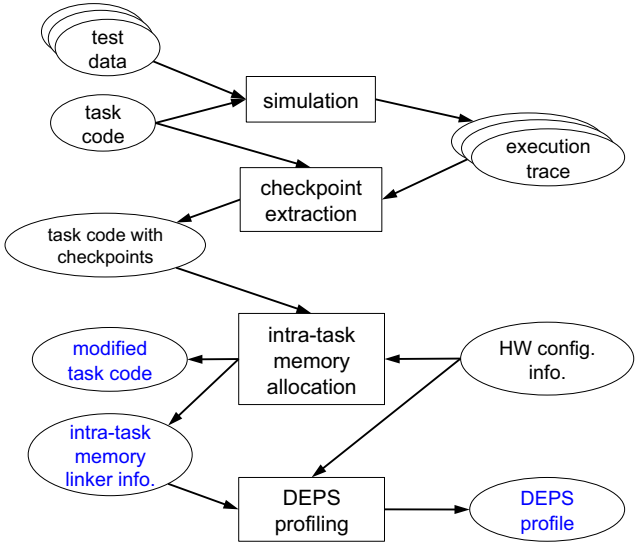

Fig. 2 The workflow of the intra-task phase.

frequency of input data set.

3.2 Checkpoints Extraction

The purpose of this step is to insert checkpoints appropriately into the source code of task. We define a checkpoint as the location in a program where the appropriate processor configuration may be changed. Checkpoints are inserted into the program as a sequence of instructions.

To achieve this purpose, we have proposed an execution trace mining in 9), which is an analytical technique for deriving the characteristics of the task automatically from the set of execution traces. This technique is applied to extract the most effective checkpoints with the large amount of execution traces obtained by the previous step.

It is important where to insert checkpoints to enhance the effectiveness of the DEPS. Checkpoints should be inserted at which the calculation of the remaining worst case execution time can be greatly changed or at which characteristic of the task can be greatly changed. These are extracted as checkpoints where the energy-efficient processor configuration may change at runtime. Note that too

many checkpoints raise the significant overhead on both execution time and energy consumption at runtime. Therefore, only effective checkpoints are selectively inserted in the source code.

3.3 Intra-task Memory Allocation

Next optimization is the memory allocation about each task. The allocations of program code and data are decided to whether SPM or main memory based on the memory access history obtained from execution traces.

At first of this step, stack data allocation proposed in 6) is performed. A frequently accessed part in the stack frame is decided to be allocated to the space of SPM. The dedicated sequences (warp/unwarp instructions[6]) are inserted into the source code to control the value of stack pointer. The completely modified source code is generated in this optimization. No modification to the source code of target application is made after this optimization. Then, the method proposed in 5) optimizes the allocation of program code and static data. It should be noted that this optimization decides only the amount of SPM used by a task.

3.4 DEPS Profiling

The last step of the intra-task optimization phase is a DEPS profiling proposed in 10). By running the cycle-level simulation with modified task code and determined memory allocation, the worst case execution time and the average energy consumption of each configuration at each checkpoint are obtained. The DEPS profiling uses these information to calculate the worst case execution time and the average energy consumption of the possible combinations of the processor configuration. A challenge is that there are too many combinations of processor configuration to check them. Our approach is to only reserve the Pareto-optimal configuration combinations that have higher performance or less energy consumption than any other ones, and other combinations are pruned at this analysis. We call its results a DEPS profile[10], that consists of configuration combination of checkpoints, and their worst case execution time and average energy consumption.

4. Inter-task Optimization Phase

Fig. 3 shows the workflow of the second phase of our framework. The application level behavior is optimized with the information of a task set, *i.e.*, the activate interval, the deadline and priority of each task within the application. Additionally, the inter-task optimization uses the results of the prior phase as shown in Fig. 3. There are two management tables as outputs of this phase.

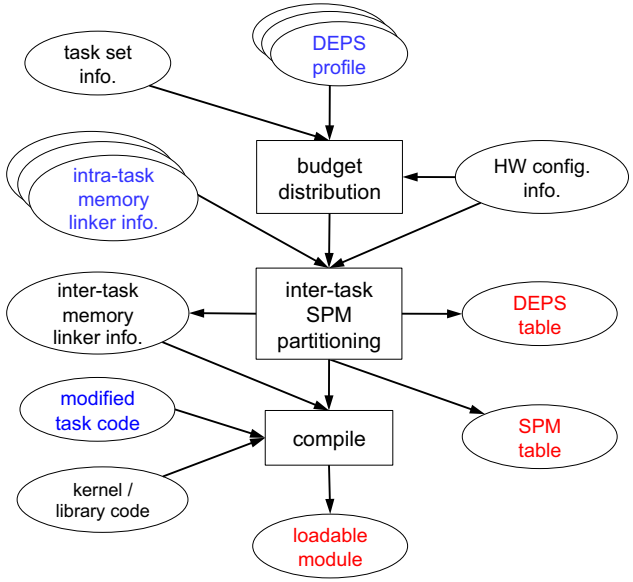

Fig. 3 The workflow of the inter-task phase.

4.1 Budget Distribution

In this step, tasks are assigned to cores, and execution time budgets are distributed to each task in such a way that the total system energy is minimized and all deadlines are met. To achieve the above objective, an integer linear problem is constructed, and it is solved by using the task set information and DEPS profiling of each task[2].

The output of this step is a DEPS management table. As shown in **Fig. 4**, this table consists of all selectable processor configurations for each checkpoint and corresponding remaining worst case execution time. Note that these configurations are sorted as increased remaining worst case execution time and decreased energy consumption to simplify their use at the runtime phase.

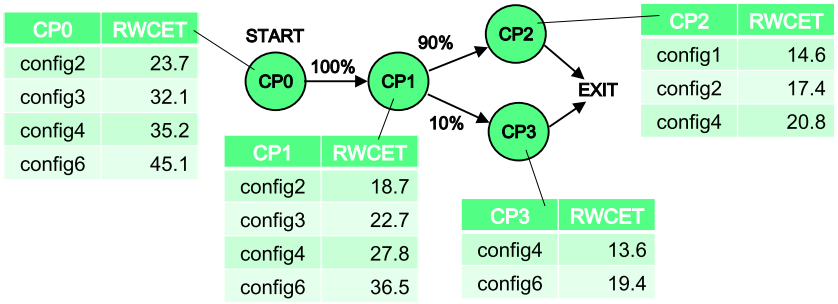

Fig. 4 An example of the DEPS management table.

4.2 Inter-task SPM Partitioning

In a multi-task environment, SPM is to be shared among the tasks. The space of SPM is spatially or temporarily partitioned to the tasks in order to minimize the amount of SPM where multiple tasks share by using the technique proposed in 7), 8). Note that the allocation of each task's memory object has been decided in the intra-task optimization phase. The inter-task SPM partitioning step only determines the address in which each task uses.

Like prior step, this step outputs a table about the SPM optimization. We call this output information as a SPM management table as shown in **Fig. 5**. The SPM management table falls into two types; the task-level and the system-level management tables. These management tables are utilized at the runtime phase. The space of SPM where multiple tasks use is maintained so that a running task temporarily uses it.

5. Runtime Optimization Phase

The pieces of optimization processing in the runtime phase are as follows;

(1) Switch the contents of SPM at each task switching
(2) Calculate the amount of slack time at each checkpoint
(3) Switch the processor configuration at each checkpoint

The first is performed with the SPM management table described in Section 4.2. When a task is switched to running state, its code and static data are temporarily

task-level table

rgn	dram addr
0	0x600
2	0x630
3	0x670

system-level table

index	alloc	spm addr	mm size
0	2	0x00	0x30
1	2	0x50	0x10
2	3	0x60	0x40
3	1	0xa0	0x30

ROM area
RAM area

Fig. 5 An example of the SPM management table.

allocated to the space of SPM. The second is conducted at each checkpoint to estimate the runtime slack approximately at low cost. The third is performed based on the slack time calculated at runtime and the DEPS management table described in Section 4.1. The optimal processor configuration that can meet the deadline constraint and with the minimum energy consumption is selected.

In our framework, the runtime overhead is limited because most of the analysis and optimization are performed at the static phase, and their results are saved as tables for runtime use.

6. Implementation

To evaluate the effectiveness of this work, we implemented the prototype of the presented framework as a toolchain.

As the target processor, the Toshiba MeP[11] was employed in our implementation. We developed the RTOS which has the runtime mechanism presented in Section 5 as the extended version of TOPPERS/FMP kernel[12]. TOPPERS/FMP kernel is the open source RTOS for the multi-processor systems. For calculating energy consumption, we have developed an energy estimation tool in the work of 13). The amount of energy consumption is estimated by the execution trace and the characterized information of the chip as shown in **Fig. 6**.

Our prototype is designed to be executed on the multi-performance processor (MPP)[3),4)]. The MPP has two processing elements and resizable 4-way instruction cache. Our RTOS treats these eight combinations of configuration as the selectable processor configuration by the DEPS function.

As the target application, we deal the brief version of the video-conference application shown in Fig. 6. It consists of video encoding/decoding tasks, audio

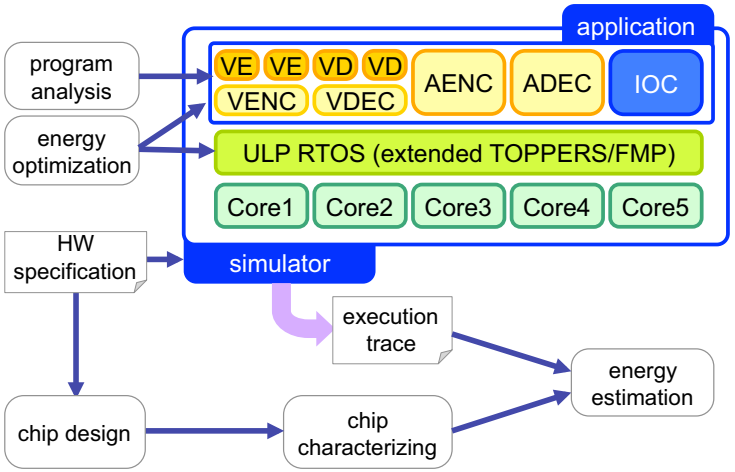

Fig. 6 The constitution of the video-conference system and the workflow of the energy estimation.

encoding/decoding tasks, and the I/O control task. Moreover, each video codec is further divided to multiple tasks. Each task is executed concurrently on the multi processor environment. We are planning to evaluate the effectiveness of implemented toolchain with its application.

7. Conclusion

This paper presented a synthesizing and gradual framework for the energy minimization of the embedded applications. Our approach mainly utilizes the trade-off between energy and performance on the processor configuration. The characteristic and runtime behavior of target application are enough analyzed at static phase of our framework. Then, energy optimization is performed at runtime based on the information of two management tables generated at static phase. We implemented the optimization framework as the toolchain and the RTOS. In future, we are planning to evaluate the effectiveness of presented framework via the practical applications.

Acknowledgments This work is supported by Core Research for Evolutional Science and Technology (CREST) of Japan Science and Technology Agency.

References

1) Zeng,G., et al.: A Generalized Framework for Energy Savings in Hard Real-Time Embedded Systems, *IPSJ Transactions on System LSI Design Methodology*, Vol.2, pp.180–188 (2009).
2) Zeng,G., et al.: A Generalized Framework for Energy Savings in Real-Time Multiprocessor Systems, *Proc. Intl SoC Design Conference (ISOCC)*, pp.44–49, (2008).
3) Ishihara,T. et al.: AMPLE: An Adaptive Multi-Performance Processor for Low-Energy Embedded Applications, *Proc. IEEE Sympo. on Application Specific Processors (SASP)*, pp.83–88 (2008).
4) Ishihara,T.: A Multi-Performance Processor for Reducing the Energy Consumption of Real-Time Embedded Systems, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E93-A, No.12, pp.2533–2541 (2010).
5) Ishitobi, Y., et al.: Code and Data Placement for Embedded Processors with Scratchpad and Cache Memories, *Journal of Signal Processing Systems*, Vol. 60, No.2, pp.211-224 (2008).
6) Gauthier,L. and Ishihara,T.: Optimal Stack Frame Placement and Transfer for Energy Reduction Targeting Embedded Processors with Scratch-pad Memories, *Proc. IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, Grenoble, France, pp.116–125 (2009).
7) Takase,H., et al.: Partitioning and Allocation of Scratch-Pad Memory in Priority-Based Multi-Task Systems, *IPSJ Transactions on System LSI Design Methodology*, Vol.2, pp.180–188 (2009).
8) Takase,H. et al.: Partitioning and Allocation of Scratch-Pad Memory for Priority-Based Preemptive Multi-Task Systems, *Proc. Design Automation and Test in Europe (DATE)*, Dresden, Germany, pp.1124-1129 (2010).
9) Tatematsu,T., et al.: Checkpoints Extraction Using Execution Traces for Intra-Task DVFS in Embedded Systems, *Proc. Intl Sympo. on Electronic Design, Test and Applications (DELTA)*, Queenstown, New Zealand, pp.19–24 (2011).
10) Kawashima,H., et al.: Intra-task Analysis of Worst Case Execution Time and Average Energy Consumption on DEPS Framework, *IEICE Technical Report*, to appear.
11) MeP (Media embedded Processor). `http://www.semicon.toshiba.co.jp/product/micro/selection/mep/index.html` (accessed 2011-02-10).
12) TOPPERS project. `http://www.toppers.jp/en/` (accessed 2011-02-10).
13) Ishihara,T. and Goudarzi,M.: System-Level Techniques for Estimating and Reducing Energy Consumption in Real-Time Embedded Systems, *Proc. Intl SoC Design Conference (ISOCC)*, pp.67–72 (2007).