Energy-Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems

Michel Goraczko
Microsoft Research
One Microsoft Way
Redmond, WA 98052
michelg@microsoft.com

Jie Liu
Microsoft Research
One Microsoft Way
Redmond, WA 98052
liuj@microsoft.com

Dimitrios Lymberopoulos
Electrical Engineering
Yale University
New Haven, CT, 06511
dimitrios.lymberopoulos@yale.edu

Slobodan Matic
Dept. of EECS
Univ. of California
Berkeley, CA, 94720
matic@eecs.berkeley.edu

Bodhi Priyantha
Microsoft Research
One Microsoft Way
Redmond, WA 98052
bodhip@microsoft.com

Feng Zhao
Microsoft Research
One Microsoft Way
Redmond, WA 98052
zhao@microsoft.com

ABSTRACT

Embedded systems with heterogeneous processors extend the energy/timing trade-off flexibility and provide the opportunity to fine tune resource utilization for particular applications. In this paper, we present a resource model that considers the time and energy costs of run-time mode switching, which considerably improves the accuracy of existing models. Given an application, the software partitioning problem then becomes an optimization over energy cost given deadline constraints, which can be formulate as an integer linear programming (ILP) problem. We apply the resource modeling and software partitioning techniques to a multi-module embedded sensing device, the $mPlatform$, and present a case study of configuring the platform for a real-time sound source localization application on a stack of MSP430 and ARM7 processor based sensing and processing boards.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

General Terms

Design

Keywords

Multi-processor scheduling, energy-aware, real-time systems

1. INTRODUCTION

Heterogeneous multi-processors (HMP) are used in embedded systems for several reasons. For example, DSPs, GPUs, or network processors are used to improve system performance by speeding up particular instructions. In other cases, IP providers package an entire system functionality, such as an 802.11b or GSM radio protocol, into a module that is then integrated into a final system. The module usually contains an embedded microcontroller and application-specific circuits. Recently, platforms with multiple heterogeneous embedded processors are exploited for the purposes of saving energy consumption and extending system lifetime. For this reason, some newly developed wireless embedded sensor platforms, such as the Pasta sensor node [13], the LEAP system [11], and the mPlatform [9], all incorporate heterogeneous multi-processors.

Power savings when using heterogeneous platforms can be significant. There are large families of embedded processors with different power and speed characteristics, ranging from 8-bit microcontrollers that consume several milliwatts to 32-bit microprocessors that consume several watts. Their sleep current and wake up transition energy are also very different. These differences can be exploited in embedded sensing applications. Take patient monitoring as an example. When there is no interesting event happening, one wants to use minimum power in low duty cycle sensing and processing for event detection so that the system life time is long. However, when there is an interesting event, one may need orders of magnitude more processing power to quickly analyze and react to it. Dynamic voltage or frequency scaling (DVS/DFS) may not be sufficient to cover the range. Having low cost, power efficient microcontroller reserved for the "quiet" time can lead to big energy saving.

In HMP platforms, the *energy-optimal software partitioning* problem asks how to assign parts of an application to each processors to achieve maximum system lifetime without sacrificing application performance. Due to simplicity in design and implementation, software partitioning approaches are more practical than dynamical scheduling approaches where each task is allocated to one of the processors at run time. Here we focus on the partition problem.

Energy saving in computing can be achieved by two means: dynamic power management(DPM) and dynamic voltage/frequency scaling (DVFS). Noting the non-trivial cost of waking up a processor from a standby (a.k.a. sleep) mode, with duty cycled workload, when the processor has no active job, one needs to decide whether to simply keep the processor idle or pay the wake up cost and put the processor into the standby mode. For single processors, when only active, idle, and standby modes are considered, a simple notion of *break-even time* is sufficient to optimize system sleep time [4]. With DVFS, one can select the best processor frequency and supply voltage to meet application performance

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

requirements with minimum energy consumption [8, 6]. Multiprocessor energy-aware scheduling starts to gain more attention as embedded systems use more processor cores. However, multiprocessor energy aware scheduling is considerably more complicated than single processor cases, even for homogeneous multi-processors and independent tasks. Several heuristics has been proposed for task allocation problems for rate monotonic scheduling [3] and with variable execution time [15]. Recent works such as [17] use efficient algorithms to assign processor speed even for complex models where worst-case execution cycle count of a task depends on CPU frequency.

Heterogeneous processors, duty cycled workload, and tasks dependencies are the main challenges to achieve optimal software task partitioning. For example, is it more energy efficient to use a low speed processor all the time, or use a high speed processor and pay the wake up cost? To meet end-to-end application deadline, shall one increase the clock frequency of an existing processor or add another processor to execute in parallel?

In this paper, we propose a comprehensive energy, performance model and an optimization framework that computes optimal task partitions for heterogeneous multiprocessors with duty-cycled interdependent real-time tasks. Note that the execution time of a task cannot be scaled easily across heterogeneous processors. In fact, they cannot be linearly scaled across different operation frequencies for a single processor due to CPU/memory/IO speed mismatch [14]. In addition, the wake up time and energy cost also depend on which destination DVFS setting a processor is waked up into [10]. These factors motivate us to use discrete operation modes to model different DVFS settings and sleeping states. This can be viewed as an extension of the power state machine model in [4].

Software tasks are modeled as acyclic dataflow graphs that captures task dependencies. This is common in sensing and signal processing applications. The dataflow graph can be further nested with state machines to achieve further expressiveness. Each task (actor) in the dataflow graph can be assigned to a processor under a DVFS mode. We show that the optimal software partitioning problem can be formulated as an integer linear programming (ILP) problem. The solution gives both the optimal number of processors needed, task-to-processor assignments, task-to-processor-mode assignments, and computation and communication schedule, all within the resource, precedence and timing constraints. Note that a similar latency minimization (i.e., throughput maximization) under energy constraints is also tractable in our framework with some modifications. For clarity reasons, we ignore the energy cost and latency of communication between processors. These costs and constraints can be easily incorporated in our model. A full-scale ILP formulation and solution is given in [7].

We apply our resource model and scheduling algorithm to a sound source localization (SSL) application on an extensible multi-processor platform, called $mPlatform$ [9], provided by Microsoft Research. SSL uses an array of microphones to estimate the direction of sound sources. It is a computational and memory intensive application that involves FFT and massive hypothesis testing. The $mPlatform$ we used in this study consists of one $ARM7$-based board and four $MSP430$-based boards each with a microphone attached. The SSL application stresses the platform in almost all dimensions: memory, execution time, and power consumption. Thus, it is an ideal candidate for verifying our resource model and scheduling algorithm.

The paper is structured as follows. Section 2 introduces the design flow for HMP software partitioning and resource and application models. Section 3 presents the ILP problem formulation and solution. Section 4 applies the resource model and scheduling algorithm to an extensible multiprocessor embedded platform and explores resource optimization for a sound source localization application.

2. APPLICATION AND RESOURCE MODELING

This section describes the software partitioning process and gives a formal specification of the application and resource models that are used in the configuration optimization presented in the next section. Throughout the paper, the lower-case, upper-case, and upper-case Gothic denote constants, variables, and sets of models respectively.

2.1 Design Flow

We solve the software partitioning problem in two steps, as shown in Figure 1.

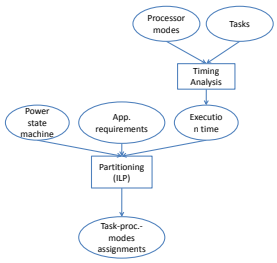

Figure 1: The design flow for energy optimal software partitioning.

The first step performs application and sleep schedule independent timing analysis for individual tasks. In this step, the execution time for each task on each feasible processor mode is obtained through either application module profiling or timing simulation such as using instruction set simulators.

Depending on application requirements, the execution time can either be worst-case execution time (WCET) for hard real-time applications, or median execution time for soft real-time applications. For example, table 2 shows the execution time for individual tasks in the sound source localization application.

The second step takes the results of step one and takes into account the relationship between processor power modes, dependency among tasks, and application performance constraints. These relationships are converted into ILP constraints as describe further in section 3. In the rest of this section, we focus on the models that feed into the second step partitioning problem.

2.2 Application Model

We use a dependent task model for application specification. In this model, an application is a set of asynchronous components interacting with messages. We call these components *tasks*. The benefit of using asynchronous message passing is that the tasks may be mapped to different processors transparent to users. The tasks respond to input events, process them, and may generate output events. So, an application is a directed graph of tasks linked by data precedence dependencies. We call this graph a *task graph*. In the following discussion, we restrict ourselves to acyclic task graphs executed periodically.

Tasks are our basic granularity of configuration mapping. Each task is mapped to a processor running in a particular power mode. The communication between two tasks are either local, if the two tasks are on the same processor; or across the communication bus,

if the two tasks are on different processors. We assume tasks are atomic, and communication only happens between the execution of tasks.

Formally,

- An application is a directed acyclic task graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ with a set of tasks $\mathcal{T}$ and $\mathcal{E} \subseteq \mathcal{T}^2$. Let $\tau \in \mathcal{T}$ denote a task, and let a pair $(\tau_i, \tau_j) \in \mathcal{E}$ denote data dependency, i.e., precedence between two tasks τ_i and τ_j.
- Let π be a period of the execution of the task graph. Here we present the procedure for a single-rate application. In a multi-rate case, different task subgraphs may have different periods, the constraints are written for multiple instances of subgraphs, and π is defined as the least common multiple of all subgraph periods.
- The model also consists of a release time r_τ for each source node $\tau \in Src(\mathcal{G})$, and a deadline time d_τ for each sink node of $\tau \in Dst(\mathcal{G})$. We assume $r_\tau \geq 0$ for each source, and $d_\tau \leq \pi$ for each sink node τ.

An application is associated with a set of user specified constraints, such as total energy budget, end-to-end real time requirements, and some task-specific mapping. In the paper, we assume that the structure of a task graph is static. For more complex applications where the structure changes over time, tasks may be migrated at run time. In those cases, multiple optimal assignments may be computed offline and applied dynamically online.

2.3 Power State Machine

We formulate a realistic power model for each processor as a state machine, where each state corresponds to a DVFS setting or a standby mode. Energy and timing cost of mode switching are labeled on the transitions.

We consider a set of processors $\mathcal{P}$ communicating through a set of buses. We ignore bus contention issue by assuming a TDMA-based bus protocol or dedicated buses between every pair of processors. More general processor communication models are possible within the formalism, but are not included in this paper for simplicity of the presentation. In fact, we completely removed communication cost in this paper.

The power model for processor $p \in \mathcal{P}$, is a finite state machine, called *power state machine* (PSM), consisting of

- A set of active power modes $\mathcal{M}$. The power consumption under mode $m \in \mathcal{M}$ is $\mathrm{p}_{p,m}$. A mode is captured by all parameters that define the power consumption of the node, such as voltage and frequency scaling, as well as memory and peripheral configurations.
- In addition to active power modes $\mathcal{M}$, there are typically two sleep modes $\mathcal{S} = \{\mathbf{I}, \mathbf{S}\}$, with $IDLE$ mode **I** and $STBY$ mode **S**. In the $IDLE$ mode (sometimes also called a halt mode), the CPU simply runs NOP with no memory or peripheral access. The internal clock is not stopped, but most other internal components are. For $p \in \mathcal{P}$, $IDLE$ mode is specified with $\mathrm{p}_{p,\mathbf{I}}$, the power consumed on component p in $IDLE$ mode **I**.

  In the $STBY$ mode the internal oscillator is completely stopped but it can be maintained outside the chip, e.g., through a real-time clock. For $p \in \mathcal{P}$, $STBY$ mode is specified with: $\mathrm{p}_{p,\mathbf{S}}$ - the power consumed on processor p in $STBY$ mode **S**.
- Mode transition costs are captures on the mode transitions in the power state machine. We use $\mathrm{p}'_{p,m}$ to denote the power consumed during waking up from $STBY$ to mode $m \in \mathcal{M}$, and $t'_{p,m}$ the wake-up time to mode $m \in \mathcal{M}$. The costs of waking up from the $IDLE$ and the transition between active modes are often considerably smaller than the the same costs for the $STBY$ mode [10], and thus will be ignored in the model.

Figure 2 shows the PSM for an ARM7 based-processor OKI ML675003 with power performance measured in [10].

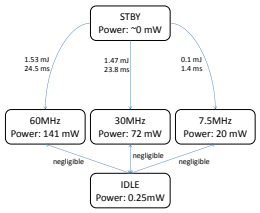

Figure 2: Power state machine for ARM7.

2.4 Performance model

Through timing analysis, we obtain the performance for each task such as execution time on each processor mode. In addition, some tasks may be pre-mapped to certain processors due to memory, IO, and other hardware constraints. Formally,

- For each task τ, processor p and mode m, the task execution time is $t_{\tau,p,m}$. This value can be measured or estimated by computing the number of cycles a task execute. Note that we can use worst-case execution time for hard real-time application, or use average (or percentile) execution time for soft real-time applications.
- An optional allocation mapping a of tasks in $\bar{\mathcal{T}} \subseteq \mathcal{T}$ to processors: $a_{\tau,p} = 1$ if task $\tau \in \bar{\mathcal{T}}$ is preallocated to processor $p \in \mathcal{P}$, otherwise $a_{\tau,p} = 0$. Depending on a problem instance, for a subset $\bar{\mathcal{T}}$ of tasks $\mathcal{T}$ allocation may be determined directly by the problem specification.

3. INTEGER LINEAR PROGRAMMING FORMALISM

With the above models, we formulate the energy-optimal software problem as a (mixed) integer linear programming (ILP) problem. The complete solution of the problem consists of allocation (task-to-processor, task-to-bus) and operation mode (task-to-mode, bus-to-mode) mappings, but also of a static time schedule for the tasks. Since the number of processors and power modes is finite and relatively small, the mappings could be encoded with binary variables. However, this is generally not true for the schedule part of the solution and, therefore, one approach for the problem is (mixed) integer linear programming (ILP). In principle, a correct ILP solver will always find an optimal solution, whenever there exists a feasible schedule that satisfies all constraints.

3.1 ILP Variables

We first present the variables of the ILP problem that form the output of the entire procedure. The set of *core* variables consists of: for each task $\tau \in \mathcal{T}$, each processor $p \in \mathcal{P}$ and each mode $m \in \mathcal{M}$:

- Binary task-to-component allocation variable A. $A_{\tau,p} = 1$ iff task τ is allocated to processor p.

- Binary task-to-mode and bus-to-mode variable M. $M_{\tau,m} = 1$ iff task τ is to execute in mode m.
- Binary task transition variable X. $X_\tau = 1$ iff, on a processor to which τ is allocated, the execution of task τ starts after a wake-up from standby mode **S**.
- Integer task execution and communication start time-instant variables S^e and S^c. Let S^e_τ denote the time instant when τ starts executing, and let S^c_τ denote the time instant when τ starts communicating its output.

Since some constraints cannot be represented as linear expressions of core program variables, additional variables are needed for the linear form of the formalism. Typically, such variables are determined once the values for the core variables are set. We use the following binary *derived* variables to model task dependencies:

- Task allocation variables $U_{\tau,p,m} = 1$ iff task τ is allocated to processor p and executes in mode m.
- Task transition variables $K_{\tau,p,m} = 1$ iff $U_{\tau,p,m} = 1$ and τ starts after a wake-up from standby mode **S**.
- $V_{\tau,\tau',p} = 1$ iff τ and τ' are both allocated to p.
- $N_{\tau,\tau',p} = 1$ iff $V_{\tau,\tau',p} = 1$ and τ' immediately follows τ, not necessarily within the the same period iteration.
- $B_{\tau,\tau',p} = 1$ iff $V_{\tau,\tau',p} = 1$ and τ' immediately follows τ across period boundaries, i.e., τ' is the first and τ the last executing on p in an iteration.
- $R_{\tau,\tau',p} = 1$ iff $N_{\tau,\tau',p} = 1$ and between the two tasks the processor p is in the standby mode **S**. Let $H_{\tau,\tau',p}$ represent the time spent in the standby mode between τ and τ'.

In general, if the ILP problem variables are bounded, as in our case, the decision problem is NP-hard. However, many problems with thousands of variables and constraints can efficiently be solved with modern ILP tools typically using branch and bound heuristics. The complexity of ILP solving is determined by the number of both core variables and derived variables. Let n_τ, n_p and n_m be the sizes of the sets $\mathcal{T}$, $\mathcal{P}$ and $\mathcal{M}$ respectivelly. The number of core binary variables is $O(n_\tau \cdot (n_p + n_m))$ and the number of core integer variables is $O(n_\tau)$. The number of derived binary variables is $O(n_\tau \cdot n_p \cdot (n_\tau + n_m))$.

3.2 ILP Constraints

The ILP problem is defined with the following set of constraints. To make this section concise, we list all constraints considerations but omit most formulas. A full formalism can be found in [7].

- **System assumptions.** A task is allocated to a single processor and a single mode.
- **Execution and communication time.** These constraints describe the relationship between core variables $A_{\tau,p}$ and $M_{\tau,m}$ and derived variables $U_{\tau,p,m}$.
- **Wake-up time.** These constraints describe the relationship between core variables X_τ and derived variables $K_{\tau,p,m}$.
- **Release, deadline and utilization.** Each source task $\tau \in Src(\mathcal{G})$ cannot start execution before its release time instant S^e_τ. Similarly, each sink task $\tau \in Dst(\mathcal{G})$ has to complete execution before its deadline time instant d_τ. Each processor $p \in \mathcal{P}$ cannot be utilized above its maximum allowed utilization u_p.
- **Ordering.** These constraints express the periodicity relations among the tasks. When task periods are not the same, extra constaints are necessary to capture task ordering across period boundaries.
- **Precedence.** A task may be scheduled for execution only after all its predecessor tasks complete.
- **Non-overlaping.** A task can begin its execution anytime but its execution cannot overlap with the execution of other tasks.
- **Standby time.** These constraints relate the standby time to the execution time.
- **Predetermined variables.** The preallocation of tasks $\bar{\mathcal{T}}$ are specified as equivalent constraints.

3.3 Objective function

We minimize the system power while satisfying timing and dependency constraints described above. Recall that $\mathrm{p}_{p,m}$ denotes the power consumed on processor $p \in \mathcal{P}$ in mode $m \in \mathcal{M} \cup \mathcal{S}$, and $\mathrm{p}'_{p,m}$ denotes the power consumed on p during a wake-up from the standby mode $\mathbf{S} \in \mathcal{S}$ to mode $m \in \mathcal{M}$. Let $T_{p,m}$ be the total time in a single period spent on component $p \in \mathcal{P}$ in mode $m \in \mathcal{M} \cup \mathcal{S}$, and $T'_{p,m}$ the time spent in waking up from standby mode to mode $m \in \mathcal{M}$. The system energy consumed in a period π is given with the linear expression

$$J = \sum_{p \in \mathcal{P}} \Big(\sum_{m \in \mathcal{M} \cup \mathcal{S}} \mathrm{p}_{p,m} \cdot T_{p,m} + \sum_{m \in \mathcal{M}} \mathrm{p}'_{p,m} \cdot T'_{p,m} \Big)$$

All power data is considered to be known, and all time variables can be represented through following linear expressions of the ILP problem variables defined previously:

$$T_{p,m} = \sum_{\tau \in \mathcal{T}} t_{\tau,p,m} \cdot U_{\tau,p,m} \quad (\text{for } m \in \mathcal{M})$$

$$T'_{p,m} = \sum_{\tau \in \mathcal{T}} t'_{p,m} \cdot K_{\tau,p,m} \quad (\text{for } m \in \mathcal{M})$$

$$T_{p,\mathbf{S}} = \sum_{\tau \in \mathcal{T}} \sum_{\tau' \in \mathcal{T}} H_{\tau,\tau',p}$$

$$T_{p,\mathbf{I}} = \pi - \sum_{m \in \mathcal{M}} (T_{p,m} + T'_{p,m}) - T_{p,\mathbf{S}}$$

4. SSL EXAMPLE ON MPLATFORM

We have built a software tool that automatically generates input to the CPLEX ILP solver, i.e., the constraints and objective function, from the high-level application and resource specification. We experimented with different models and task graphs containing up to 30 tasks. In most cases the optimization procedure takes 0-20 seconds on a 2GHz server. On larger graphs it may take up to a couple of minutes, but a user may specify cutoff time after which the solver outputs the best solution found. We next apply the optimization method to a sound source localization (SSL) application on mPlatform.

Sound source localization (SSL) is a classical sensing application that uses a microphone array to detect the direction of a sound source. They are used in teleconference, intelligent lecture/class rooms[12], human-computer interactions, and target tracking. The basic principle is to use the time differences of arrival from the sound source to different microphones to triangulate the sound source location. There are many algorithms proposed for the application

[16]. In this paper, we use a SRP-PHAT algorithm [5] with four microphones placed at the four corners of a square. The length of the sides is 20cm.

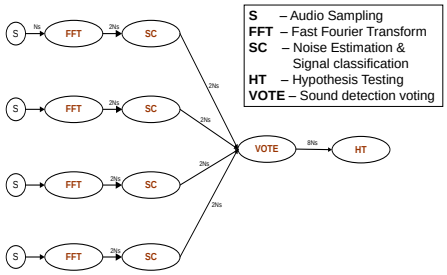

Figure 3: Task graph of an SSL application.

Application specification. Fig. 3 shows the task graph of the SRP-PHAT algorithm. The FFT task applies Fourier transform to the sampled sound signals. The SC task performs noise power estimation. If more than two channels decide that their blocks contain voice samples, the HT task is executed to determine source location through correlation maximization. HT task performs hypothesis testing to find the most likely angle of sound source. For this discussion, we ignore the cost of VOTE. We used the following algorithm parameters in our SSL implementation: sampling frequency $f_s = 8KHz$; FFT block size $N_{FFT} = 512$; number of hypotheses $N_h = 12$; sliding window size $N_w = 240$ samples; and the number of microphones $N_m = 4$.

Hardware Platform and PSM. We implemented SSL on a mPlatform with two sets of processors $ARM7$ processors (thereon referred as ARM) and $MSP430$ microcontroller (thereon referred as MSP). Each of the MSP can connect up to 2 omni-directional microphones. Given that we need 4 microphone channels, we consider a maximum of 1 ARM board and 4 MSP boards. We use a 24-bit wide parallel bus that connects to the local processor through a programmable bus, implemented using Complex Programmable Logic Device (CPLD). The CPLD bus is shared using a TDMA-like protocol. The CPLD bus is fast enough that the communication latency can be ignored in comparison with task execution time.

The $ARM7$ processor in our implementation is OKI ML675003 microcontroller with 512K of Flash ROM, and 32K RAM [2]. It runs at a maximum clock frequency of 60MHz. The clock can be scaled down by 2 and 8, resulting in 3 different modes corresponding to different operating frequencies. The TI MSP430F1611 microcontroller used in the mPlatform sensor boards operates at 4 different frequencies: 6MHz, 3MHz, 1.5MHz, and 0.75MHz [1]. None of the processors has voltage scaling features.

Table 1 shows the key parameters in the processor PSM, most of which were obtained by direct measurements on our experimental setup. We observe that for ARM, there is a non-trivial cost, in terms of both time and energy, to wake up from a standby mode.

Performance Model We prototyped each software module in SSL and measured their execution time under various processor modes, as shown in 2. Since HT requires more storage space that a MSP420 can provide, it can only be implemented on ARM7. This further becomes a preallocation constraint in the ILP.

It is interesting to observe that although execution time scales linearly on MSP, it is not so at the low frequency for ARM. For example, comparing ARM at 60MHz and 7.5MHz, the clock speed reduced by a factor of 8, but the execution time only increase by a factor of 5. Combining this with about 7 times power consumption reduction at the lowest speed, we find that 7.5MHz is a sweet spot for ARM. We believe this is due to the performance mismatch between the CPU and flash memory. This verifies our choice of using explicit power states in the modeling.

Parameter	ARM7 @2.5V	MSP430 @3V
Active power at full speed (mW)	141 @60MHz	10.8 @6MHz
Active power at $1/2$ speed (mW)	72	5.4
Active power at $1/4$ speed (mW)	—	2.7
Active power at $1/8$ speed (mW)	20	1.4
Idle power (mW)	0.25	0.005
Standby power (mW)	negligible	negligible
Wakeup energy (to full speed) (mJ)	1.5	negligible
Wakeup energy (to lowest speed) (mJ)	0.1	negligible
Wakeup time (to full speed)	24.5ms	6μs
Wakeup time (to lowest speed)	1.4ms	$< 6\mu$s

Table 1: Processor power consumption at different operating modes.

Processor Mode	FFT (ms)	SC (ms)	HT (ms)
ARM7, 60MHz	7.8	4.4	111
ARM7, 30MHz	15.6	9.0	222
ARM7, 7.5MHz	39.6	23.3	567
MSP430, 6MHz	99.2	37.2	—
MSP430, 3MHz	196	76	—
MSP430, 1.5MHz	394	152	—
MSP430, 0.75MHz	792	300	—

Table 2: The execution time of tasks under different processor modes.

Software Partitioning Exploration We used the ILP procedure presented in Sec. 3 to explore the optimal resource management assuming the samples are processing with larger period: 130ms, 256ms, and 1s. Note that we only process the last 512 samples taken in a period, and treat the period boundaries as deadlines. Depending on the number of preallocated tasks and preassigned modes the number of variables and constraints varies, but typically the number of core variables is about one hundred and the number of constraints several thousands.

Figure 4 shows task allocation results obtained from the ILP solver:

A. When period $D = 130$ms, the optimal schedule is shown in Figure 4(a). Both ARM and 4 MSP have to run at full speed in order to meet the stringent timing constraints. The scheduling result clearly shows the pipelining among the tasks cross the period boundary. That is, FFT and SC executed on MSP provide data for HT in the next period. Even after ARM finishes its active task, it cannot go to the STBY mode since the next iteration is coming up. The total energy for one iteration is 21.7mJ, corresponding to 166.7mW average power consumption. There is a 300mW fixed power cost on ARM and 0.4mW on MSP boards.

B. When period $D = 256$ms, the optimal schedule is shown in Figure 4(b). Notice that only two of the four MSP boards are necessary. Although the schedule in [A] is still feasible, the wakeup energy cost prevents ARM to get into the standby mode. The optimal schedule assign ARM to 30MHz, with 34ms of idle mode. The total energy for one iteration is 22.1mJ, corresponding to 86.4mW average power consumption due to the longer period. Note that were we still use

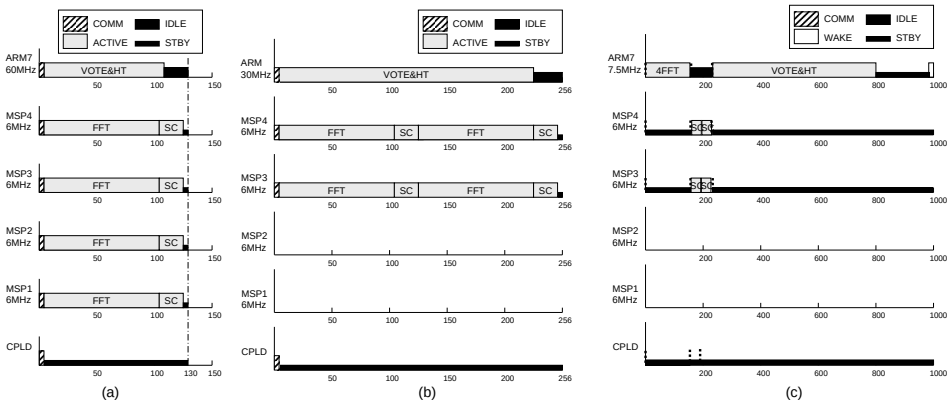

Figure 4: Optimal task scheduling results: (a)period=$130ms$ **(b) period=**$256ms$ **(c) period=**$1000ms$

schedule A, ARM7 would go to standby and the total dynamic energy use would be 22.7mJ.

C. When period $D = 1000$ms, the optimal schedule is shown in Figure 4(c). The very relaxed timing constraints, together with the negligible communication cost, leads to an assignment of tasks only by their corresponding processor energy efficiency. In fact, we need to pre-allocate Sampling tasks with 0 cost to the MSP boards to enforce the hardware constraints that each MSP can only support up to two microphones. This permits ARM to use the slowest mode of 7.5MHz, which is the most energy efficient. Since the wakeup energy is also low, ARM can get into standby mode between computations. The optimal schedule also assigns FFT to ARM, but leaves SC on MSP, since it is more energy efficient to run SC on MSP at 6MHz. The total energy consumption for one iteration is 16.2mJ, corresponding to 16.2mW on average.

From the resource optimization results, we can see that with increasing iteration periods, we can improve power performance and lower hardware cost by properly selecting the number of processors, their modes and task assignments. The cost to pay is an increase in the application response time. In comparison to the ILP results, we measured 575 mW average power consumption for the entire application running schedule A. Therefore, the scheduling experiments capture about 28% of the total power consumption. With about 400 mW consumed by voltage regulators, microphones, LEDs, memory, I/O, etc., there is a large space to exploit in terms of controlling peripherals.

5. CONCLUSION

We tackle the challenge of resource modeling and software partitioning in heterogeneous multiprocessor embedded systems. Our model does not assume linear execution time nor power consumption with respect to clock frequencies. It is based on a more generic notion of PSM with the cost of mode switching. With an ILP formalism, we solve the optimal hardware configuration, processor mode selection, and task-to-processor assignment to achieve minimum energy consumption given end-to-end time constraints. Using a sound source localization application as an example, we show fine-grained resource trade off based on application quality requirements. When the number of tasks in an application is huge, ILP solutions become challenging. Fast and suboptimal heuristics are necessary as a future work.

6. REFERENCES

[1] MSP430: Ultra-Low Power Microcontrollers. http://www.ti.com.

[2] OKI ML67Q5003: ARM7TDMI Processor. http://www.okisemi.com.

[3] T. A. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 213–223, Washington, DC, USA, 2005. IEEE Computer Society.

[4] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. Very Large Scale Integr. Syst.*, 8(3):299–316, 2000.

[5] M. Brandstein and H. Silverman. A robust method for speech signal time-delay estimation in reverberant rooms. In *ICASSP*, page 375. IEEE Computer Society, 1997.

[6] D. Li, P. H. Chou, and N. Bagherzadeh. Mode selection and mode-dependency modeling for power-aware embedded systems. In *vlsid, ASP-DAC/VLSI Design 2002*, pages 697–705, 2002.

[7] J. Liu and S. Matic. mplatform: A flexible and efficient architecture for sharing data in stack-based sensor network platforms. In *Microsoft Research, Technical Report MSR-TR-2006-142*, 2006.

[8] Y.-H. Lu, L. Benini, and G. D. Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(11):1284–1305, 2002.

[9] D. Lymberopoulos, B. Priyantha, and F. Zhao. mplatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *Information Processing in Sensor Networks (IPSN)*, 2007.

[10] D. Lymberopoulos and A. Savvides. Xyz: a motion-enabled, power aware sensor node platform for distributed sensor network applications. In *IPSN*, pages 449–454. IEEE Press, 2005.

[11] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. J. Kaiser. The low power energy aware processing (leap) embedded networked sensor system. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 449–457, New York, NY, USA, 2006. ACM.

[12] Y. Rui, A. Gupta, J. Grudin, and L. He. Automating lecture capture and broadcast: technology and videography. *ACM Multimedia Systems Journal*, 10(1):3–15, 2004.

[13] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho, and L. Wang. A modular power-aware microsensor with > 1000x dynamic power range. In *Information Processing in Sensor Networks (ISPN) 2005, SPOTS track, Los Angeles, CA*, April 2005.

[14] D. C. Snowdon, S. Ruocco, and G. Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, Sept. 2005.

[15] C. Xian, Y.-H. Lu, and Z. Li. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 664–669, New York, NY, USA, 2007. ACM.

[16] C. Zhang, Z. Zhang, and D. Florêncio. Maximum likelihood sound source localization for multiple directional microphones. In *ICASSP*, 2007.

[17] X. Zhong and C.-Z. Xu. Frequency-aware energy optimization for real-time periodic and aperiodic tasks. In *LCTES*, pages 21–30, 2007.