Resource Modeling and Scheduling for Extensible Embedded Platforms

Slobodan Matic*, Michel Goraczko†, Jie Liu†, Dimitrios Lymberopoulos‡, Bodhi Priyantha†, Feng Zhao†
*Dept. of EECS, UC Berkeley, Berkeley, CA 94720
matic@eecs.berkeley.edu
†Microsoft Research, One Microsoft Way, Redmond, WA 09852
{michelg,liuj,bodhip,zhao}@microsoft.com
‡Electrical Engineering, Yale University, New Haven, CT, 06511
dimitrios.lymberopoulos@yale.edu

***Abstract*— Modern embedded processors have the flexibility of dynamic switching between power operation modes, such as using voltage and frequency scaling. Platforms with heterogeneous processors and reconfigurable buses further extend the energy/timing trade-off flexibility and provide the opportunity to fine tune resource usage for particular applications. This paper gives a resource model for heterogeneous multi-processor embedded platforms and formulates power-aware real-time resource scheduling problems as integer linear programming problems. In particular, we take the time and energy costs of mode switching into account, which considerably improves the accuracy of the model. We apply the resource model to a stackable multi-processor embedded platform, called $mPlatform$, and present a case study of scheduling a sound source localization application in a stack of four MSP430-based sensing boards and one ARM7-based processing board.**

I. Introduction

Computing platforms that feature multiple processors interconnected via high-speed buses have a number of advantages: flexibility in scheduling and executing concurrent applications to meet deadlines and reconfigurability to mitigate local failures or respond to changes in processing or communication resources. Examples include the multi-core/many-core systems, systems-on-a-chip (SoC), as well as extensible embedded platforms that comprise multiple stackable boards[19], [8], [12]. Resources here refer to the available processing or communication components which in turn consume power in order to operate. In many settings, especially in the embedded or real-time applications, it is important to minimize the power consumption for a longer battery life or minimize the latency in carrying out time-sensitive tasks.

Power savings when using heterogeneous platforms can be significant. There are large families of embedded processors with very different power and speed characteristics, ranging from 8-bit microcontrollers that consume several milliwatts to 32-bit microprocessors that consume several watts. As embedded applications become more complex, their application requirements may vary dramatically from time to time. Take embedded sensing, such as patient monitoring, as an example. When there is no interesting event happening, one wants to use minimum power in 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. Having low-end, power efficient microcontroller dedicated for the "quiet" time can lead to significant power saving.

Processors such as the ARM or MSP430 can be programmed to operate in one of the several power modes of operation, by software-controlled voltage (DVS), or operating frequency (DPM), or both. The flexibility to choose which processor and which operation mode to use opens up a possibility of a fine-grained resource management in heterogeneous multi-processor systems by trading power with speeds for the various operating components. However, beside timing constraints, a scheduler has to take into a consideration properties of different power modes. Analytical optimization is often not an option due to complex or unknown power consumption models.

The problem becomes more challenging when the cost of mode switching is taken into account. In the most aggressive power saving mode, called standby mode ($STBY$), almost all components inside the CPU are turned off, including the internal oscillator. As a consequence, when waking up from a $STBY$ mode by an external interrupt, the processor must wait until the internal oscillator stabilizes before it can start computing. Moreover, the time and energy cost for the transition depends significantly on the operating mode the processor enters next. And this cost can be substantial. For example, in an ARM7-based CPU waking up from a deep sleep mode to the most active mode takes 24.5 ms of time and 1.5 mJ of energy [13], which is enough for the processor to perform two 1024-sample FFT procedures. The $IDLE$ mode, on the other hand, consumes a little more power, but has little overhead to wake up from.

This paper addresses the resource modeling and power-aware task scheduling problem for extensible multi-processor systems. It formulates the scheduling problem as an integer linear programming (ILP) problem. In our formalism, tasks may have data dependencies, and each task can run at a distinct operation mode on one of the processors. There may be multiple communication media (e.g. buses) that connect the processors. Each of them may be shared by multiple processors and operate at different speed mode. A processor

may either idle or enter a $STBY$ mode when there is no task to execute. However, if it goes standby, it must pay the extra cost when waking up from the sleeping mode.

Our formalism covers both power and timing properties. In one setting, the objective is to minimize power consumption while satisfying the period and/or deadline constraints of an application. We assume an application is specified as an acyclic task graph where an edge represents a data dependency between two tasks. This is common in signal processing applications where the problem is represented by a periodic data-flow task graph. The resource specification consists of a power model for each processor and communication element (i.e., bus). The solution to the problem gives both the optimal task-to-processor allocation, task-to-mode mapping, and computation and communication schedule, all within the resource, precedence and timing constraints. Note that a similar latency minimization (i.e., throughput maximization) under power constraints is also tractable in our framework with some modifications.

We used our resource model and scheduling algorithm for sound source localization (SSL) on an extensible multiprocessor platform, called $mPlatform$, being developed at Microsoft Research (Figure 1). SSL uses an array of microphones to estimate the direction of the sound source. 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.

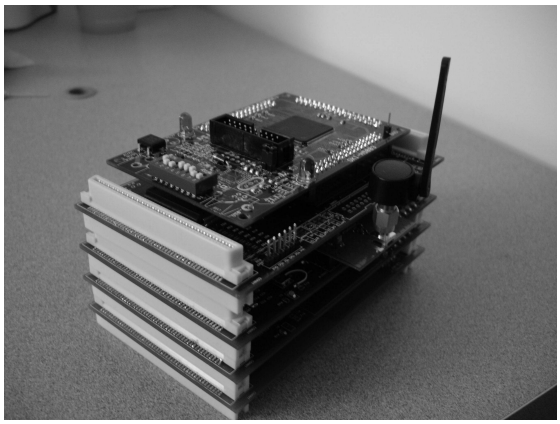


Fig. 1. A prototyping $mPlatform$ stack.

The paper is structured as follows. Section II introduces resource and application models and gives a simple motivating example for such models. Section III presents the power-aware scheduling problem as an ILP formulation. Section IV applies the resource model and scheduling algorithm to an extensible multiprocessor embedded platform and explores application parameter space for a sound source localization application.

II. Task and Resource Modeling

We use a multi-mode, event-driven task model for extensible multi-processor systems. In this model, an application is structured as a set modes, called *configurations*. In each configuration, the application is a set of asynchronous components interacting with messages. We call these components *tasks*. The tasks are event triggered, that is, they respond to input events, process them, and may generate output events. So, a configuration is a directed graph of tasks linked by data precedence dependencies. We call this graph a *task graph*. In the following discussion, we restrict us to acyclic task graphs.

The benefit of using asynchronous message passing is that the tasks may be mapped to different processors transparent to users. This gives a system the flexibility to adjust operation conditions according to application requirements. Configuration changes, though, are explicitly specified by programmers. When changing configurations, existing tasks can be terminated; new tasks can be created; and the states of running tasks can be reconfigured. But within a configuration, the structure of a task graph keeps unchanged.

Each configuration may associate with user specified constraints, such as total energy budget, end-to-end real time requirements, and some specific task mapping. A task scheduling algorithm is used to determine the setting of processors and bus, such as voltage scaling and clock frequency, the complete allocation of tasks to processors, and the release time and deadline for each task. In this paper, we only consider the task scheduling problem within a configuration. This is not a significant limitation, since once schedules are computed for every configuration, changes of configuration can reload new schedules.

At run time, each task is mapped to a processor. 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 first formally specify the task and resource model that is used in the ILP formalism presented in the next section. Throughout the paper the following notation convention is used. The constant parameters, the variables, and the sets of the model are written in lower-case, upper-case, and upper-case Gothic letters respectively.

A. Resource Model

We assume three specifications are given: a platform specification of the hardware configuration and resources, an application specification of dependency and timing requirements, and a mapping specification that includes worst case execution time of each task on each processor and worst case communication time of each message on each bus.

Platform Specification

- A set of processors $\mathcal{P}$ communicating through a set of buses $\mathcal{B}$. We assume that each bus is either shared using a TDMA-based protocol or is dedicated to a single processor. More general processor communication models are possible within the formalism, but are not included in this paper for simplicity of the presentation.

- A power model for each component $c \in \mathcal{P} \cup \mathcal{B}$, i.e., for each processor or bus c a set of active operating modes $\mathcal{M}$ specified with power $\mathrm{p}_{c,m}$ consumed in each mode $m \in \mathcal{M}$. Almost all micro-controllers support frequency scaling, so, for instance, each mode in $\mathcal{M}$ may be related to a particular processor operating frequency.
  In addition to active power modes $\mathcal{M}$, there are typically two sleep modes $\mathcal{S} = \{\mathbf{I}, \mathbf{S}\}$, with $IDLE$ mode $\mathbf{I}$ and $STBY$ mode $\mathbf{S}$. In the $IDLE$ mode the internal clock is not stopped, but most other internal components are. For $c \in \mathcal{P} \cup \mathcal{B}$ $IDLE$ mode is specified with $\mathrm{p}_{c,\mathbf{I}}$, the power consumed on component c in $IDLE$ mode $\mathbf{I}$.
  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 $c \in \mathcal{P}$ $STBY$ mode is specified with: $\mathrm{p}_{c,\mathbf{S}}$ - the power consumed on component in $STBY$ mode $\mathbf{S}$, $\mathrm{p}'_{c,m}$ - the power consumed during waking up from $STBY$ to mode $m \in \mathcal{M}$, and $t'_{c,m}$ the wake-up time to mode $m \in \mathcal{M}$. The costs of waking up from the $IDLE$ are often considerably smaller than the the same costs for the $STBY$ mode [13], and thus will be ignored in the model.
  We also assume that a bus can only operate at one active mode within one configuration to avoid the complexity of dynamically synchronizing the TDMA protocol.
- For each component $c \in \mathcal{P} \cup \mathcal{B}$ an upper bound on allowed component utilization u_p.

Application Specification

- 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.
- A period π of the execution of the task graph. Here we present the procedure for a single-rate applications. 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.
- 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 $\tau \in Dst(\mathcal{G})$. A source (resp. sink) node is each node of $\mathcal{G}$ with input (resp. output) degree equal to 0. We assume $r_\tau \geq 0$ for each source, and $d_\tau \leq \pi$ for each sink node τ.

Mapping Specification

- For each task $\tau \in \mathcal{T}$, processor $p \in \mathcal{P}$ and mode $m \in \mathcal{M}$, the worst-case task execution time $t_{\tau,p,m}$. This value can be measured or estimated by computing worst-case task number of cycles.
- For each task $\tau \in \mathcal{T}$, bus $b \in \mathcal{B}$ and mode $m \in \mathcal{M}$, the worst-case communication time $t_{\tau,b,m}$ of the message that contains task output. This value can be measured or estimated by determining the largest size of the output of task τ.
- 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. For instance, a data sampling task may execute only on certain processor boards connected with a particular sensor. Similarly, a subset of tasks may have preassigned modes of operation.

The scheduling algorithm can also take into account energy per sensor reading or energy per memory read or write operation. Although this does not make the corresponding ILP more complex, we do not present it here to keep the presentation simpler.

B. Motivating Example

Beside solving the task allocation and scheduling problem for a given application, the objective of the formalism presented in Sec. III is to determine the active and sleep modes of operation that are optimal with respect to power. In this section we try to motivate the optimization procedure by showing that the optimal mode selection is not obvious and depends on power costs and other parameters even for the simplest applications.

We consider an application with a single periodic task τ executing on a processor p with period π. So, no communication is involved in the example. The power model for p consists of three active modes $\mathcal{M} = \{1, \frac{1}{4}, \frac{1}{32}\}$, where 1 denotes the mode with the largest and $\frac{1}{32}$ with the smallest (32 times smaller) operating frequency. In addition, there are two sleep modes, $IDLE$ and $STBY$, $\mathcal{S} = \{\mathbf{I}, \mathbf{S}\}$. We assume that after executing task τ in a certain mode m the processor enters one of the sleep modes s. Let a given parameter u be equal to the processor utilization in the slowest frequency mode. Thus, the execution time of the task in mode $m \in \mathcal{M}$ is

$$t_{\tau,p,m} = \frac{\pi \cdot u}{32 \cdot m}$$

The simple power model presented above provides the following expression for the energy spent in every period

$$J(m,s) = \mathrm{p}_{p,m} \cdot t_{\tau,p,m} + \mathrm{p}_{p,s} \cdot (\pi - t_{\tau,p,m} - t'_{p,m,s}) + \mathrm{p}'_{p,m,s} \cdot t'_{p,m,s}$$

The three elements of the sum denote energies spent in mode m, in sleep mode s and in waking-up from mode s to mode m, respectively from the first sum element.

In previous research we measured the parameters of the equation for an ARM-based processor and, together with other data, they are presented in the Sec. IV. We used them to compute the optimal modes m and s that result in minimal energy $J(m,s)$. The computation was performed for the range of period $\pi = [0, 100] ms$ and utilization $u = [0, 1]$ values. The results are shown in Fig. 2. It follows that, for different values of parameters, all active and sleep power modes can in some combination be optimal. In general, for large period π the optimal active mode is the largest frequency mode and optimal sleep mode is the standby mode. However, if u is large, even for medium values of π the optimal combination may be slowest frequency mode and idle sleep mode. It is interesting

to note that for the linear execution time model as used here, if the waking-up costs are all zero, the optimal modes are the largest frequency and standby modes, irrespective of the values for π and u.

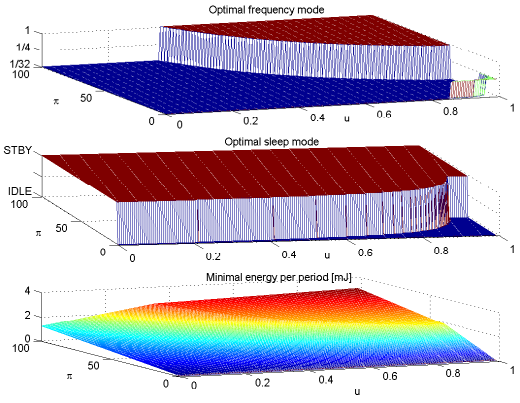


Fig. 2. Optimal active and sleep power modes for minimal energy costs per period in a simple periodic application

III. Integer Linear Programming Formalism

A. ILP Variables

The complete solution of the problem informally described in the introduction 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, buses and 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. For the CPLEX solver [9] that was used in this study, all constraints have to be of the form $(\sum_i a_i \cdot X_i)\ \rho\ b_i$, where ρ is an element of the set $\{\leq, =, \geq\}$, coefficients a_i and b_i are real-valued constants and X_i are program variables that can be of either binary (0 or 1) or integer type.

We first present the variables of the ILP problem that form the output of the entire procedure. The set of *core* variables consists of:

- Binary task-to-component allocation variable A. For each task $\tau \in \mathcal{T}$ and each processor $p \in \mathcal{P}$ let $A_{\tau,p}$ be 1 if and only if task τ is allocated to processor p. Also, for each task $\tau \in \mathcal{T}$ and each bus $b \in \mathcal{B}$ let $A_{\tau,b}$ be 1 if and only if the output of task τ is communicated over bus b.
- Binary task-to-mode and bus-to-mode variable M. For each task $\tau \in \mathcal{T}$ and each mode $m \in \mathcal{M}$ let $M_{\tau,m}$ be 1 if and only if task τ is to execute in mode m. Also, for each bus $b \in \mathcal{B}$ and each mode $m \in \mathcal{M}$ let $M_{b,m}$ be 1 if and only if bus b is to operate in mode m.
- Binary task transition variable X. For each task $\tau \in \mathcal{T}$ let X_τ be 1 if and only if, 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. For each task $\tau \in \mathcal{T}$ 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.

In general, if the ILP problem variables are bounded, as in our case, the problem is NP-hard. However, problems with thousands of variables and constraints can efficiently be solved with modern ILP tools. We tried to keep the number of core variables as small as possible because this number mostly determines the actual computational complexity.

Since some constraints cannot be represented as linear expressions of core program variables, additional variables are needed for the linear form of the program. Typically, such variables are determined once the values for the core variables are set. In the ILP problem constraints we use the following variables *derived* from the core variables described above:

- $U_{\tau,p,m}$, $U_{\tau,b,m}$ and $K_{\tau,p,m}$. For each $\tau \in \mathcal{T}$, $p \in \mathcal{P}$ and $m \in \mathcal{M}$, let binary variable
  - $U_{\tau,p,m}$ be 1 if and only if task τ is allocated to processor p and executes in mode m.
  - $K_{\tau,p,m}$ be 1 if and only if task τ, in addition to being allocated to processor p and executing in mode m, starts after a wake-up from standby mode **S**.

  For each $\tau \in \mathcal{T}$, $b \in \mathcal{B}$ and $m \in \mathcal{M}$, let $U_{\tau,b,m}$ be 1 if and only if the output of task τ is communicated over the bus b which operates in mode m.
- $V_{\tau,\tau',p}$, $N_{\tau,\tau',p}$, $B_{\tau,\tau',p}$, $R_{\tau,\tau',p}$, and $H_{\tau,\tau',p}$. For each pair of tasks $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$, let binary variable
  - $V_{\tau,\tau',p}$ be 1 if and only if τ and τ' are both allocated to processor p,
  - $N_{\tau,\tau',p}$ be 1 if and only if, in addition to τ and τ' being allocated to processor p, τ' immediately follows τ, not necessarily within the the same period iteration,
  - $B_{\tau,\tau',p}$ be 1 if and only if, in addition to τ and τ' being allocated to processor p, τ' immediately follows τ across period iterations, i.e., if and only if task τ' is the first and τ the last task executing on p,
  - $R_{\tau,\tau',p}$ be 1 if and only if, in addition to τ and τ' being allocated to processor p and τ' immediately following τ, 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.

For instance, the consumed power of a processor directly depends on the time spent in the standby mode. In fact, representing this time as a linear combination of core and derived variables makes the constraints of the ILP problem more difficult.

B. ILP Constraints

The ILP problem is defined with the following set of constraints:

- **System assumptions.** A task is allocated to a single processor. For all tasks $\tau \in \mathcal{T}$

$$\sum_{p \in \mathcal{P}} A_{\tau,p} = 1$$

A task executes in a single mode. For all tasks $\tau \in \mathcal{T}$

$$\sum_{m \in \mathcal{M}} M_{\tau,m} = 1$$

A shared bus operates in single mode (other than idle mode **I**). For all buses $b \in \mathcal{B}$

$$\sum_{m \in \mathcal{M}} M_{b,m} = 1$$

- **Execution and communication time.** By definition of a derived variable $U_{\tau,p,m}$, we have $U_{\tau,p,m} = 1$ if and only if $A_{\tau,p} = 1$ and $M_{\tau,m} = 1$. We first note that arbitrary binary variables X, Y and Z satisfy expression X AND Y = Z if and only if they satisfy linear inequality $0 \leq X+Y-2Z \leq 1$. Thus, for all $\tau \in \mathcal{T}$, $p \in \mathcal{P}$ and $m \in \mathcal{M}$

$$0 \leq A_{\tau,p} + M_{\tau,m} - 2 \cdot U_{\tau,p,m} \leq 1$$

Similarly, for all $\tau \in \mathcal{T}$, $b \in \mathcal{B}$ and $m \in \mathcal{M}$

$$0 \leq A_{\tau,b} + M_{b,m} - 2 \cdot U_{\tau,b,m} \leq 1$$

Note that the solution execution time E_τ of task τ, and communication time of its output C_τ, can be represented as the following linear expressions, that will be used as a shorthand in other constraints

$$E_\tau \triangleq \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} t_{\tau,p,m} \cdot U_{\tau,p,m}, C_\tau \triangleq \sum_{b \in \mathcal{B}} \sum_{m \in \mathcal{M}} t_{\tau,b,m} \cdot U_{\tau,b,m}$$

- **Wake-up time.** By definition of a derived variable $K_{\tau,p,m}$, we have $K_{\tau,p,m} = 1$ if and only if $X_\tau = 1$ and $U_{\tau,p,m} = 1$. Thus, for all $\tau \in \mathcal{T}$, $p \in \mathcal{P}$ and $m \in \mathcal{M}$

$$0 \leq X_\tau + U_{\tau,p,m} - 2 \cdot K_{\tau,p,m} \leq 1$$

The wake-up time W_τ of task τ can be represented as

$$W_\tau \triangleq \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} t'_{p,m} \cdot K_{\tau,p,m}$$

- **Release, deadline and utilization.** Each source task $\tau \in Src(\mathcal{G})$ cannot start execution before its release time instant

$$r_\tau \leq S^e_\tau$$

Similarly, each sink task $\tau \in Dst(\mathcal{G})$ has to complete execution before its deadline time instant

$$S^e_\tau + E_\tau \leq d_\tau$$

Each processor or bus $c \in \mathcal{P} \cup \mathcal{B}$ cannot be utilized above its maximum allowed utilization u_c

$$\sum_{\tau \in \mathcal{T}} \sum_{m \in \mathcal{M}} t_{\tau,c,m} \cdot U_{\tau,c,m} \leq \pi \cdot u_c$$

- **Ordering.** By definition of a derived variable $V_{\tau,\tau',p}$, we have $V_{\tau,\tau',p} = 1$ if and only if $A_{\tau,p} = 1$ and $A_{\tau',p} = 1$. Thus, for all $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$

$$0 \leq A_{\tau,p} + A_{\tau',p} - 2 \cdot V_{\tau,\tau',p} \leq 1$$

Binary variable $N_{\tau,\tau',p}$ is 1 if and only if on processor p task τ' executes immediately after task τ. The following three expressions put constraints on $N_{\tau,\tau',p}$. For all $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$

$$N_{\tau,\tau',p} \leq V_{\tau,\tau',p}$$

For all $\tau \in \mathcal{T}$ and $p \in \mathcal{P}$

$$\sum_{\tau' \in \mathcal{T}} N_{\tau,\tau',p} \leq A_{\tau,p}$$

For all $p \in \mathcal{P}$

$$\sum_{\tau \in \mathcal{T}} \sum_{\tau' \in \mathcal{T}} N_{\tau,\tau',p} = \sum_{\tau \in \mathcal{T}} A_{\tau,p}$$

Binary variable $B_{\tau,\tau',p}$ is 1 if and only if τ' is the first and τ the last task executing on p. Thus, we have for all $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$

$$B_{\tau,\tau',p} \leq N_{\tau,\tau',p}$$

For all $p \in \mathcal{P}$

$$\sum_{\tau \in \mathcal{T}} \sum_{\tau' \in \mathcal{T}} B_{\tau,\tau',p} = 1$$

We will use the following short notation

$$V_{\tau,\tau'} \triangleq \sum_{p \in \mathcal{P}} V_{\tau,\tau',p}, N_{\tau,\tau'} \triangleq \sum_{p \in \mathcal{P}} N_{\tau,\tau',p}, B_{\tau,\tau'} \triangleq \sum_{p \in \mathcal{P}} B_{\tau,\tau',p}$$

For instance, $V_{\tau,\tau'} = 1$, if there exists a processor such that both τ and τ' are allocated to it. If, for a given $\tau \in \mathcal{T}$ there is no $\tau' \in \mathcal{T}$ such that $(\tau, \tau') \in \mathcal{E}$ and the two tasks are allocated to different processors, than the output of τ should not be sent over any bus. Thus, for each task $\tau \in \mathcal{T}$

$$\sum_{b \in \mathcal{B}} A_{\tau,b} \leq \sum_{(\tau,\tau') \in \mathcal{E}} (1 - V_{\tau,\tau'})$$

As a consequence, for a task whose output is not sent over any bus we have $C_\tau = 0$.

- **Precedence.** A task may be scheduled for execution only after all its predecessor tasks complete. For each dependent task pair $(\tau, \tau') \in \mathcal{E}$

$$S^e_\tau + E_\tau \leq S^e_{\tau'}$$

Also, the output of a task may be communicated only after the task completes. For each $\tau \in \mathcal{T}$

$$S^e_\tau + E_\tau \leq S^c_\tau$$

If the two tasks in a dependent task pair $(\tau, \tau') \in \mathcal{E}$ are assigned to different processors, then the start time instant of τ' is constrained by the completion of the communication of the output of τ. In the following

constraint, the number z is a positive constant with a large value. If the two tasks are assigned to the same processor the rightmost element takes a large value. The given constraint still holds and the constraint is automatically satisfied, so the communication time is ignored. However, if the two tasks are not assigned to the same processor the rightmost element is zero and the communication time is taken into account. For each dependent task pair $(\tau, \tau') \in \mathcal{E}$

$$S^c_\tau + C_\tau \leq S^e_{\tau'} + z \cdot V_{\tau,\tau'}$$

- **Overlap.** A task can begin its execution anytime but its execution cannot overlap with the execution of other tasks. Recalling large constant z as in the previous constraint, the following constraint is not automatically satisfied only if $N_{\tau,\tau'} = 1$, i.e., only if on a processor the execution of τ' immediately follows the execution of τ. In essence, assuming $B_{\tau,\tau'} = 0$, the following constraint requires $S^e_{\tau'}$ to be larger than S^e_τ for the execution time of task τ and wake-up time of task τ' (if different than 0). Binary variable $B_{\tau,\tau',p}$ is 1 if and only if τ' is the first and τ the last task executing on p. So, the term $-\pi \cdot B_{\tau,\tau'}$ accounts if the execution of τ extends over the period π bound. For all $\tau, \tau' \in \mathcal{T}$

$$S^e_\tau + E_\tau + W_{\tau'} - \pi \cdot B_{\tau,\tau'} \leq S^e_{\tau'} + z \cdot (1 - N_{\tau,\tau'})$$

  Since a bus is shared through a TDMA protocol additional communication constraint is that two transmissions from the same processor board cannot overlap.

$$S^c_\tau + C_\tau - \pi \cdot B_{\tau,\tau'} \leq S^c_{\tau'} + z \cdot (1 - N_{\tau,\tau'})$$

- **Standby time.** Derived binary variable $R_{\tau,\tau',p}$ is 1 if and only if $N_{\tau,\tau',p} = 1$ and processor p starts executing τ' after waking up from standby mode **S** ($K_{\tau'} = 1$). Thus, for all $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$

$$0 \leq N_{\tau,\tau',p} + K_{\tau'} - 2 \cdot R_{\tau,\tau',p} \leq 1$$

  If $R_{\tau,\tau',p} = 1$ then derived variable $H_{\tau,\tau',p}$ is the time spent in standby mode **S** after completing τ, else $H_{\tau,\tau',p} = 0$. If $R_{\tau,\tau',p} = 1$ then the both sides of the following inequality reduce to zero making $H_{\tau,\tau',p}$ equal to the the standby time. For all $\tau, \tau' \in \mathcal{T}$ and $p \in \mathcal{P}$

$$0 \leq S^e_\tau + E_\tau + H_{\tau,\tau',p} + W_{\tau'} - S^e_{\tau'} - \pi \cdot B_{\tau,\tau',p} \leq z \cdot (1 - R_{\tau,\tau',p})$$

$$0 \leq H_{\tau,\tau',p} \leq z \cdot R_{\tau,\tau',p}$$

- **Predetermined variables.** The preallocation of tasks $\bar{\mathcal{T}}$ specified with the mapping a generate the following constraint. For all $\tau \in \bar{\mathcal{T}}$

$$A_{\tau,c} = a_{\tau,c}$$

  Similar constraints can be written for predetermined mode variables.

C. Objective function

The optimization objective defines the objective function and specifies the optimization direction, min or max. In this paper we minimize the system power while satisfying timing and dependency constraints described above. We assume that the total system power consists of power consumed by computation and communication elements, i.e., by processors in $\mathcal{P}$ and buses in $\mathcal{B}$. Recall that $\mathrm{p}_{c,m}$ denotes the power consumed on component $c \in \mathcal{P} \cup \mathcal{B}$ in mode $m \in \mathcal{M} \cup \mathcal{S}$, and $\mathrm{p}'_{c,m}$ denotes the power consumed on c during a wake-up from the standby mode $\mathbf{S} \in \mathcal{S}$ to mode $m \in \mathcal{M}$. Let $T_{c,m}$ be the total time in a single period spent on component $c \in \mathcal{P} \cup \mathcal{B}$ in mode $m \in \mathcal{M} \cup \mathcal{S}$, and $T'_{c,m}$ 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

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

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:

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

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

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

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

In the scope of this project we have built a small tool that automatically generates input to the CPLEX ILP solver, i.e., the constraints and objective function, from a high-level application and resource specification.

IV. Model Evaluation Case Study

A. Sound Source Localization

Sound source localization (SSL) is 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[18], human-computer interactions, and target tracking[6]. 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 [22]. 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.

In SRP-PHAT (and similar algorithms) the location is determined by computing delay between times of audio signal arrival to different microphones. This delay could, in principle, be estimated from the signal cross-correlation function. With an array of microphones, the sum of correlation functions over all pairs of microphones has to be considered and maximized. If the number of used microphones is N_m such a sum would

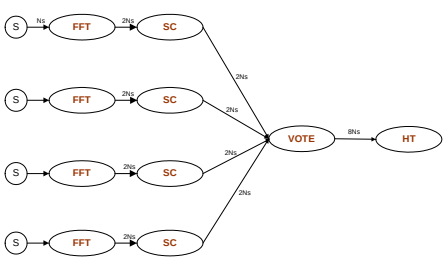


Fig. 3. Task graph of an SSL application. Sound signals are sampled from 4 microphones synchronously; Fourier transform is applied to each signal sequence to extract the frequency components; the SC task classifies if the source sequence comes from human speech or background noise; and the HT task performs the location hypothesis testing in the frequency domain.

naturally require $O(N_m^2)$ computational complexity. However, by assuming a certain suitable weighting function to take into account the noise, the complexity can be reduced to $O(N_m)$ [22].

In practice, the maximization of the correlation function is achieved through hypothesis testing. Namely, multiple source location hypotheses are tested and the one that results in the largest correlation is declared as the source location. For teleconference applications the location is commonly represented in spherical coordinates, so each hypothesis corresponds to a spherical segment at a certain distance from the center of the scene. In this study we considered simpler implementation in which each hypothesis is related to a planar angle. The selection of the number of hypotheses N_h is also important and directly affects computational complexity (see [16] for improvements).

The signal processing algorithms like SRP-PHAT are usually performed in the frequency domain because of more efficient processing and noise filtering. The algorithm is performed for a window of frequencies, i.e., for a window of N_w discrete frequencies. For audio applications this window is usually within 0.2-4KHz. Moreover, since hypothesis testing is used in the SRP-PHAT algorithm, the frequency domain allows for a table with a phase shift for each hypothesis to be computed off-line, thus reducing the number of operations performed on-line.

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. In the simplest variants of the algorithm the noise level is used to classify the currently processing block of samples, i.e., to decide whether the currently processed sound is noise or voice. If more than two channels decide that their blocks contain voice samples, the HT task is executed to determine source location through correlation maximization. In more complex algorithm variants the noise level itself is used in the expression for correlation. If more than 3 SC tasks vote that the samples come from a sound source rather than noise, the HT task is triggered. HT task performs hypothesis testing to find the most likely angle of sound source. For this discussion, we ignore the cost of the VOTE task.

Parameter	ARM7	MSP430
Active power at full speed (mW)	186	10.8
Active power at $1/4$ of full speed (mW)	76.4	2.7
Active power at lowest speed (mW)	42.5@1.9MHz	1.4@0.75MHz
Idle power (mW)	42	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 I

PROCESSOR POWER CONSUMPTION AT DIFFERENT OPERATING MODES.

B. Hardware Design

We evaluated SSL using $mPlatform$, a modular and extensible hardware platform developed at Microsoft Research. $mPlatform$ consists of a collection of circuit boards; a number of these boards are stacked together to implement a device with specific features. Some of these boards are general purpose processing boards while others are special purpose boards such as radio boards for wireless communication, sensor boards for sensing physical phenomena, and power boards for supplying power to a stack of boards. Each special purpose board, except for power boards, also has a local processor which enables efficient real-time event handling.

All $mPlatform$ boards implement a uniform hardware interface. This uniform interface makes it possible to stack together any combination of boards to implement a device that meets specific application needs.

Each $mPlatform$ board connects to multiple buses for inter-processor communication. There is 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. A set of switchable serial buses enable dynamic pair-wise communication between processors using standard serial protocols such as RS232 and SPI. There is also a multi-master I2C bus shared by all the local processors. For SSL implementation, we used a stack of 6 boards. The stack consisted of a processing board with an ARM processor, 4 sensor boards - each with an omni-directional microphone attached to an $MSP430$ processor, and a power board. We used the 24-bit CPLD bus for inter-processor communication.

Platform specification. We used the OKI ML675003 microcontroller with 512K of Flash ROM, and 32K RAM for the ARM processor [2]. The processor in our system runs at a maximum clock frequency of 60MHz. The clock can be scaled down by 2,4,8,16, or 32, resulting in 6 different modes corresponding to different operating frequencies. In our previous research [13] we presented results of extensive power measurements. Some of the measured data relevant for this study is given in Table IV-B.

The TI MSP430F1611 microcontroller used in the $mPlatform$ sensor boards operates at 4 different frequencies, the highest being 6Mhz [1]. The required power data for this microcontroller is taken from [17]. The parallel bus has maximum clock rate of 16MHz.

Parameter	RingCam	*mPlatform*
sampling frequency f_s	16KHz	8KHz
sample block size N_{FFT}	640	512
number of hypotheses N_h	90	12
number of microphones N_m	8	4
window size N_w	240	240

TABLE II
THE BASELINE PARAMETERS OF THE SSL APPLICATION.

Similar to the processors, the bus can also be slowed down resulting in five different possible clock rates making it possible to vary the $CPLD$ power consumption. The required power data can be computed from the curves given in [3].

C. Performance Model

The parameter space of the SSL application is large, which enables tuning the performance even for embedded implementations such as $mPlatform$. Table II shows the baseline parameters we implemented with $mPlatform$, in comparison to the similar algorithm implemented in RingCam project [7] using a dual CPU (Pentium 4) 2.2GHz PC. The table gives an idea of the performance level that can be expected from the embedded solution such as $mPlatform$.

Beside the basic signal processing parameters such as sampling frequency f_s, the sample block size N_{FFT}, and the window size N_w, there are several application-level parameters such as the number of microphones N_m, the number of hypotheses N_h (determining the sensing accuracy), and the classification threshold (determining the sensitivity). The effect of each of these parameters on the application time and memory complexity can be tremendous. For instance, the size of the constant look-up table that stores phase-shift values for all location hypotheses is $O(N_h \cdot N_m \cdot N_w)$. Since each value is a complex number, even for 4 microphones, 12 hypothesis and window size 240, the requirement easily sums up to 200KB. The RAM requirements are $O(N_{FFT} \cdot N_m)$ which may also be critical since even the ARM board has only 32KB of RAM.

The time complexity analysis of the SSL algorithm is important if we want to have timing guarantees for the application. The tasks in basic variant of the algorithm perform the following order of operations (usually multiply operations): $O(N_{FFT} \cdot N_m)$ for FFT, $O(N_h \cdot N_m)$ for SC, and $O(N_h \cdot N_m \cdot N_w)$ for HT. So, the dominant part of the time is required for the HT task, which becomes even worse if noise correction algorithms are implemented ($O(N_m^2)$). The processor boards in the current $mPlatform$ do not have a DSP or floating-point co-processor so all signal processing algorithms are implemented using software floating-point emulation. However, the code for all the tasks typically consists of nested loops of arithmetic operations, so the execution times are highly deterministic and almost data independent.

Mapping Specification. We conclude this subsection by presenting some of the execution times of the SSL tasks directly measured on different processors of our prototype implementation. The basic application parameters for all experiments are the parameters shown in Table II. We measured the task execution times for the fastest mode, and verified that the execution times under other frequencies are scaled linearly from these numbers.

The execution time of the FFT task, measured on both ARM and MSP board, is shown in Table III for different sample block sizes N_{FFT}. Table IV shows the measured worst case execution times of the HT and SC tasks on the ARM board that in most scenarios has to execute these tasks. Table (a) gives execution times for different number of hypotheses N_h, and Table (b) gives execution times for different window sizes N_w.

samples	MSP430 (ms)	ARM7 (ms)
16	1.62	0.219
32	3.8	0.364
64	8.82	0.686
128	20.1	1.4
256	45.2	2.95
512	99.2	6.32
1024	218	13.6

TABLE III
THE EXECUTION TIME OF FFT OF VARIOUS SIZE.

D. Resource Scheduling and Performance Exploration

Assuming the sampling frequency f_s=8KHz and sample block size N_{FFT}=512 the block of samples is collected in time period of $T_f = \frac{N_{FFT}}{f_s}$=64ms. Consider first the case when all tasks execute on ARM board. When the total execution time $t_{tot} = t_{FFT} + t_{SC} + t_{HT}$ for the entire task graph is taken into account, we see that the ARM processor can process every sample in real-time only for the most conservative values of other application parameters. Namely, Fig. 4(a) and (b) show the ratio $\frac{t_{tot}}{T_f}$ when the parameters N_h and N_w are varied respectively. Ideally, this ratio should be less than 1. So, for instance, if N_h=12 (i.e., location resolution of 30 degrees) and N_w=240, we have $\frac{t_{tot}}{T_f}$=2.6, which means that only every third sample can be processed.

We used the ILP procedure presented in Sec. III to explore the optimal resource management assuming the application parameters from Table II and resource models presented in previous subsections. Motivated by the simple analysis from the previous paragraph we performed the procedure for different values of the application period from $200ms$ to $250ms$.

Generally speaking, the fundamental trade-off in this system comes with the fact that for any task, it takes MSP 15 times more time to execute, but uses $1/18$ the energy of the ARM processor. The idle mode on ARM is significantly more expensive than that on the MSP and waking up to an active mode costs time and energy. So, it makes sense to allocate tasks as much as possible to the MSP processor as long as the real time constraints are not violated. This gives ARM enough time to go into a deeper $STBY$ mode. The effect of more tasks executing on MSP becomes even more apparent when application parameters, e.g. the FFT block size or number of hypotheses, are reduced, making the task execution times smaller.

Figure 5 shows the task allocation for the following three cases:

N_h	2	3	4	6	12	18
t_{HT}	27.8	37.8	48.8	75.8	138.7	235.98
t_{SC}	1.15	1.62	1.92	3.8	4.8	9

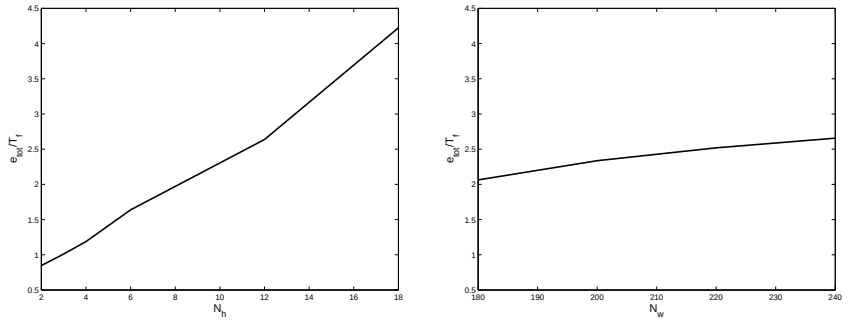
(a)

N_w	160	180	200	220	240
t_{HT}	92.67	103	120	130.4	139
t_{SC}	2.4	3.8	4.25	5.52	5.74

(b)

TABLE IV

MEASURED TASK EXECUTION TIMES VALUES FOR HT AND SC ON ARM.



(a) (b)

Fig. 4. Portion of samples that can be processed as a function of number of hypotheses (a) and frequency window size (b)

A. When voting is considered as part of HT and the period is $200ms$, the optimal allocation is to have MSP boards send all their samples to ARM board and have the FFT, SC, as well as HT running on ARM. All tasks run in the fastest processor mode. The ARM cannot switch to $STBY$ mode. The idle time on ARM is $42ms$, while the MSP boards spend most of the time in the $STBY$ mode. The total energy cost in one cycle is $34.7mJ$.

B. When voting is considered as part of HT and the period is $250ms$, it pays off to set ARM into the $STBY$ mode, and move both FFT and SC to the MSP board. All tasks run in the fastest processor mode. This results in a total energy cost for one cycle of $34.1mJ$. Now the ARM processor spends $87.5ms$ in the $STBY$ mode.

C. It is interesting to observe that when voting is considered as a task (as shown in Figure 3) with arbitrarily small execution time, it completely changes the mode that the ARM wakes up into. In this case, the ARM wakes up to the slowest mode with transition time $1.4ms$ and transition energy $0.1mJ$. Now, with the same MSP allocation as in B the total energy cost is $33.2mJ$. This verifies the observation made in [13] that when a ARM processor wakes up from a standby mode, it should always first wakes up to the slowest frequency mode.

It is clear that the HT task is the biggest time, memory, and power consumer. Its 138ms execution on ARM consumes more than 25 mJ (or 75%) of energy. Note that the above analysis is for worst case scenarios. So, if there is only background noise, the HT task does not have to be trigger, and the $ARM7$ may not need to be activated. This shows the advantage of heterogeneous multiprocessor platforms.

We also implemented the complete SSL application on $mPlatform$ with more functionalities such as adapting to noise level. By allocating FFT, SC, and noise-level updating on each MSP board and allocating VOTE, SSL, and Display tasks on ARM board, we can achieve an end-to-end delay of $235ms$ per 512 samples, that is, our final system processes roughly one forth of the source signals. A detailed break down measurement is shown in Table V.

task	board	execution time (ms)
ADC and DMA	MSP	64
512 point FFT	MSP	100
SC	MSP	72
Noise level update	MSP	48
Bus (4 channels)	CPLD	2
Voting	ARM	unmeasurable
HT	ARM	138
Overhead	MSP	14
Overhead	ARM	6

TABLE V

THE EXECUTION TIME OF SSL APPLICATION ON *mPlatform*.

V. RELATED WORK

There exists extensive research on system-level low power optimization. A good survey is given in [4]. Most of the techniques, especially analytical ones, study single processor systems. Our ILP formulation integrates multiprocessor allocation and schedule generation with operating mode selection. The ILP framework has also recently been used for optimization of multiprocessor systems, but with different optimization criteria and without taking into account power at all. So, in [10], [20], and [23] the objective is to maximize, respectively, the throughput, the minimal task slack, and task extensibility. In [14] authors use integer programming to solve the problems with more complicated power, but simpler timing models. In sensor networks research, ILP formalism was recently also used to address optimization of global communication between nodes [21], [15].

VI. CONCLUSION

We tackle the challenge of resource modeling and software scheduling in extensible multi-processor embedded systems.

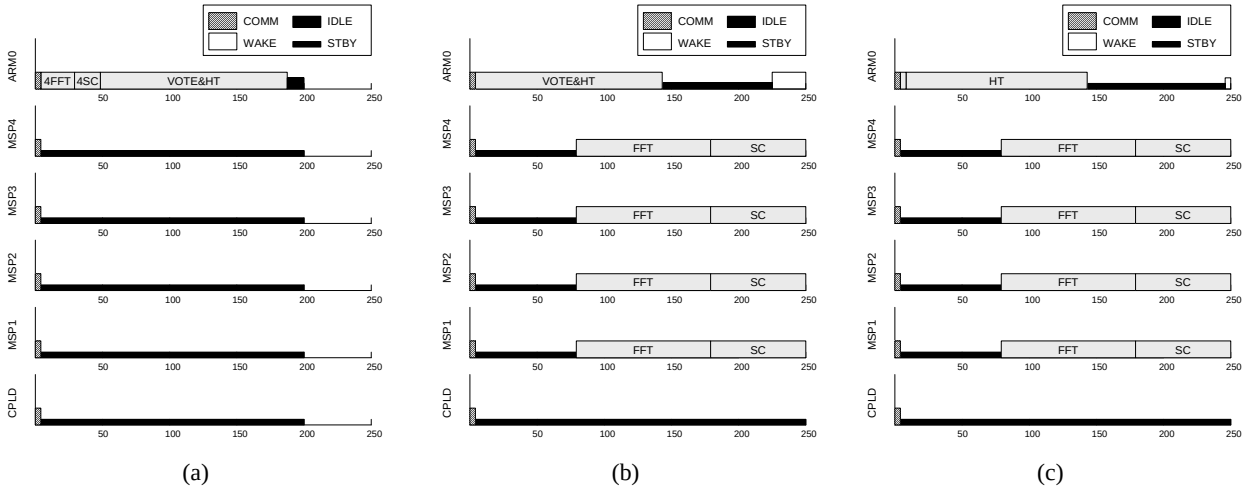


Fig. 5. Task scheduling results (a) Optimal solution, period=$200ms$ (b) Optimal solution, period=$250ms$ (c) Optimal solution when VOTE task runs at low speed, period=$250ms$

Our model takes into account multiple operation modes and the cost of mode switching. With an ILP formalism, we are able to solve the optimal task to specific processor mode assignment. Thus, with given end-to-end real time constraints, we can achieve minimum energy consumption. We have built $mPlatform$, a stackable multi-processor platform with heterogeneous microprocessors, including MSP430 and ARM7 class processors. Using a sound source localization application as an example, we show interesting resource trade off based on application quality requirements.

Our example has a periodic data-flow task graph that is common in signal processing applications. As such the scheduling is assumed to be performed off-line. However, its output, in the form of a static schedule, can be used as a basis for an on-line scheduler if the application contains also aperiodic or bursty task requests (for instance, see [11]). We plan to further develop on-line scheduler and task migration mechanisms for extensive multi-processor systems. These algorithms will most likely involve heuristics due to the complexity of seeking optimal task assignment. The ILP formalism gives a baseline and theoretical bound for other heuristics.

REFERENCES

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

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

[3] Xilinx Coolrunner series CPLDs. http://www.xilinx.com/products/silicon_solutions/cplds/coolrunner_series/index.htm.

[4] Luca Benini, Alessandro Bogliolo, and Giovanni De 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] J.C. Chen, L. Yip, J. Elson, H. Wang, D. Maniezzo, R.E. Hudson, K. Yao, and D. Estrin. Coherent acoustic array processing and localization on wireless sensor networks. *Proc. of the IEEE*, 91(8):1154–1162, 2003.

[7] Ross Cutler, Yong Rui, Anoop Gupta, Jonathan J. Cadiz, Ivan Tashev, Li wei He, Alex Colburn, Zhengyou Zhang, Zicheng Liu, and Steve Silverberg. Distributed meetings: a meeting capture and broadcasting system. In *ACM Multimedia*, pages 503–512. ACM Press, 2002.

[8] Nicholas Edmonds, Doug Stark, and Jesse Davis. Mass: modular architecture for sensor systems. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 53, Piscataway, NJ, USA, 2005. IEEE Press.

[9] Ilog, Inc. Solver CPLEX. http://www.ilog.com/products/cplex/.

[10] Yujia Jin, Nadathur Satish, Kaushik Ravindran, and Kurt Keutzer. An automated exploration framework for fpga-based soft multiprocessor systems. In *CODES+ISSS*, pages 273–278. ACM Press, 2005.

[11] Jiong Luo and Niraj K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *ICCAD*, pages 357–364. IEEE Press, 2000.

[12] Dimitrios Lymberopoulos, Bodhi Priyantha, and Feng Zhao. A flexible and efficient architecture for sharing data in stack-based sensor network platforms. Technical Report MSR-TR-2006-142, Microsoft Research, 2006.

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

[14] Saraju P. Mohanty, N. Ranganathan, and Sunil K. Chappidi. Ilp models for simultaneous energy and transient power minimization during behavioral synthesis. *ACM Trans. Design Autom. Electr. Syst.*, 11(1):186–212, 2006.

[15] Luca Negri and Lothar Thiele. Power management for bluetooth sensor networks. In *EWSN*, pages 196–211. Springer, 2006.

[16] J.M. Peterson and C. Kyriakakis. Hybrid algorithm for robust, real-time source localization in reverberant environments. In *ICASSP*, pages 1053–1056. IEEE Press, 2005.

[17] Joseph Polastre, Robert Szewczyk, and David E. Culler. Telos: enabling ultra-low power wireless research. In *IPSN*, pages 364–369. IEEE Press, 2005.

[18] Yong Rui, Anoop Gupta, Jonathan Grudin, and Liwei He. Automating lecture capture and broadcast: technology and videography. *ACM Multimedia Systems Journal*, 10(1):3–15, 2004.

[19] 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.

[20] T. Sivanthi and U. Killat. Global scheduling of periodic tasks in a decentralized real-time control system. In *IEEE IWFCS*. IEEE Press, 2004.

[21] Yang Yu and Viktor K. Prasanna. Energy-balanced task allocation for collaborative processing in wireless sensor networks. *MONET*, 10(1-2):115–131, 2005.

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

[23] Wei Zheng, Jike Chong, Claudio Pinello, Sri Kanajan, and Alberto L. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. In *ACSD*, pages 132–141. IEEE Computer Society, 2005.