Exploring m-Platform for Local Application Performance: Sound Source Localization

Jie Liu
Microsoft Research
Jie.Liu@microsoft.com

Slobodan Matic
UC Berkeley
matic@eecs.berkeley.edu

1 Introduction

In this report we present results of a performance study of the *m-platform*, a modular stack-based architecture for sensor nodes. An m-platform node consists of several boards or other hardware modules that have different performance properties, but share the same communication interfaces. The main motivation for a stack-based architecture is design flexibility and reconfiguration, and not necessarily least possible power or fastest system response. Nevertheless, here we investigate local processing and communication properties, i.e., the performance within a single node - a single stack of boards. We think this is an important analysis, especially considering the high power cost of radio communication between different nodes.

The report focuses on power minimization, and in particular, on two aspects related to the power consumption of the m-platform. First, each processing element allows for independent dynamic power management. Namely, each processor 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. However, the largest power reduction is achieved in *sleep* modes where the processor, some, or all peripherals are not clocked. Second, an m-platform stack is a heterogenous architecture, i.e., it consists of the processors of different speed and power properties. Typically, several low-power processors collect sensor data which is sent to and processed on a more power consuming processor. The energy consumed on these two types of processors can vary several orders of magnitude. In this research we study a high-level procedure that may provide answers to the following questions. Can low-power processors also process data in real-time, and thus eliminate the need for more powerful processors? Given application requirements how many processors are necessary? Can pure parallelization, i.e., with homogenous processors reduce total power? What are the optimal operation modes for each processor?

We present an integer linear programming (ILP) formulation of the problem. The problem addresses the tradeoff between system power and latency. Typically, the objective is to minimize power consumption while satisfying the period and/or deadline constraints of the application. In addition, we assume application is specified as an acyclic task graph where an edge represents a data dependency between two tasks, and an edge weight the size of the message communicated between the two tasks. The resource specification consists of a power model for each processor (and communication element, i.e., bus) of the node. The solution of the problem encodes both the optimal task-to-processor allocation, processor-to-mode mapping and task execution schedule. So, the objective function minimizes power while not violating the resource, precedence and timing constraints. Note that a similar problem of latency minimization (throughput maximization) under power constraints is also tractable in the ILP framework with some modifications.

The proposed procedure is aimed for periodic data-flow task graphs that are common in signal processing applications. As such it 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 the on-line scheduler in [17]). Also, the adopted power model does not contain energy or time costs for transition of operation modes. This is both because mode-transition part of consumed power largly depends on the dynamic portion of the application, and because for the audio applications considered in this study, that part is negligible.

The paper is structured as follows. In Sec. 2 we give the details of the proposed ILP formalism. The second part of the report presents two case studies in exploring the performance of the m-platform. Sec. 3 describes the procedure on a simplified application with Fast Fourier Transform computations and gives some conclusions about power minimization for a typical m-platform stack. Sec. 4 reports on the current m-platform implementation of the sound source localization algorithm. We also analyze power/latency performance using the values from the actual measurements.

Related Work. The details of the m-platform design and objectives can be found in [3]. There exists extensive research on sytem-level low power optimization. A good survey is given in [9]. 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 [16], [23], and [26] the objective is to maximize, respectively, the throughput, the minimal task slack, and task extensibility. In [19] 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 [25, 20].

2 Integer Linear Programming Formalism

We assume that the system specification, i.e., the *input* of the problem consists of:

- A set of processors $\mathcal{P}$ communicating through a shared bus b using a TDMA protocol. Let $\mathcal{P}_b$ denote set $\mathcal{P} \cup \{b\}$.
- For each processor or bus $p \in \mathcal{P}_b$ a power model, i.e., a set of operating modes $\mathcal{M}_p$ specified with power $P_{p,m}$ consumed in each mode $m \in \mathcal{M}_p$. Almost all microcontrollers support frequency scaling, so, for instance, each mode in $\mathcal{M}_p$ may be related to a particular processor operating frequency. In addition, and to simplify the notation, for each $p \in \mathcal{P}_b$ the sleep mode which is entered whenever p becomes idle is treated separately from $\mathcal{M}_p$. Let s denote this mode and $P_{p,s}$ the (relatively small) amount of power consumed in it.
- For each $p \in \mathcal{P}$ an uper bound on processor utilization u_p.
- A directed acyclic task graph $G = (\mathcal{T}, E)$ with a set of tasks $\mathcal{T}$ and $E \subseteq \mathcal{T}^2$. Let $\tau \in \mathcal{T}$ denote a task, and let a pair $(\tau_i, \tau_j) \in E$ denote data dependency, i.e. precedence, between two tasks τ_i and τ_j.
- A period T 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 T is defined as the least common multiple of all subgraph periods.
- A release time r_i for each source node of G and a deadline time d_i for each sink node of G. A source (resp. sink) node is each node of G with input (resp. output) degree equal to 0. We assume $d_i \leq T$ for each sink node.
- For each task $\tau \in \mathcal{T}$, processor $p \in \mathcal{P}$ and mode $m \in \mathcal{M}_p$, a worst-case task execution time $e_{\tau,p,m}$. This value can be measured or estimated by computing worst-case task number of cycles.
- For each task $\tau \in \mathcal{T}$ and mode $m \in \mathcal{M}_b$ of a shared bus, a worst-case task output communication time $c_{\tau,m}$. This value can be measured or estimated by determining the largest size of the output of task τ.

The procedure 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.

2.1 ILP Variables

The solution of the problem informally described in Sec. 1 consists of allocation (task-to-processor) and operation mode (processor-to-mode) mappings, but also of a static time schedule for the tasks. Since the number of processors 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 the right optimization formalism is integer linear programming formalism. 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 [15] that was used in this study, all constraints have to be of the form $\sum_i a_i \cdot X_i \leq b_i$ or $\sum_i a_i \cdot X_i \geq b_i$, where coefficients a_i and b_i are real constants and X_i are program variables that can be of either binary or integer type.

The set of *core* variables of the ILP program, which is also the *output* of the procedure, consists of:

- Binary task-to-processor allocation matrix X. For each task $\tau \in \mathcal{T}$ and each processor $p \in \mathcal{P}$ let $X_{\tau,p}$ be 1 if task τ is allocated to p, or 0 otherwise.

- Binary processor-to-mode mapping matrix M. For each processor or bus $p \in \mathcal{P}_b$ and each mode $m \in \mathcal{M}_p$ let $M_{p,m}$ be 1 if processor or bus p is to operate in mode m, or 0 otherwise.

- Integer task execution and communication start time instant lists S and C. For each task $\tau \in \mathcal{T}$ let S_τ denote the time instant when τ is ready to start executing, and let C_τ denote the time instant when τ is ready to start communicating its output.

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. Let the pre-allocation function $\bar{X}$ be defined on $\bar{\mathcal{T}}$ and given as an additional input to the problem. Similarly, a subset $\bar{\mathcal{P}}_b$ of $\mathcal{P}_b$ may have pre-assigned modes of operation. Let the pre-mapping function $\bar{M}$ be defined on $\bar{\mathcal{P}}_b$ and given as an additional input to the problem.

Since some constraints cannot be represented as linear expressions of core program variables, additional variables are needed for the linear form of the program. For that purpose the following binary variables are *derived* from the core variables described above:

- $W_{\tau,p,m}$. For each $\tau \in \mathcal{T}$, $p \in \mathcal{P}$, and $m \in \mathcal{M}_p$ let $W_{\tau,p,m}$ be 1 if task τ is allocated to processor p and processor p is mapped to mode m, and 0 otherwise. For $p = b$ let $W_{\tau,m}$ be 1 if the output of the task τ is communicated over the bus and if bus is mapped to mode m, and 0 otherwise.

- $V_{\tau_i,\tau_j,p}$. For each pair of tasks τ_i and τ_j of $\mathcal{T}$, let $V_{\tau_i,\tau_j,p}$ be 1 if τ_i and τ_j are both scheduled for execution on the processor p, and 0 otherwise.

- N_{τ_i,τ_j}. This variable is taken into account only when the two tasks τ_i and τ_j are allocated to the same processor. Let N_{τ_i,τ_j} be 1 if task τ_i executes before τ_j, and 0 otherwise.

- B_{τ_i,τ_j}. For each pair of tasks τ_i and τ_j of $\mathcal{T}$, let B_{τ_i,τ_j} be 1 if τ_i and τ_j are allocated to the same processor and both tasks have to send their outputs using the bus, and 0 otherwise.

2.2 ILP Constraints

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

- A task can be allocated to only one processor. For all tasks $\tau \in \mathcal{T}$

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

- A processor or a bus can operate in only one mode (at a time, while not in a sleep mode). For all $p \in \mathcal{P}_b$

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

- By definition of a derived variable $W_{\tau,p,m}$, we have $W_{\tau,p,m} = 1$ if and only if $X_{\tau,p} = 1$ and $M_{p,m} = 1$. Since the variables are binary, this constraint can be expressed as

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

- Each source task $\tau_i \in \mathcal{T}$ cannot start execution before the release time instant

$$r_i \leq S_{\tau_i}$$

  Similarly, each sink task $\tau_i \in \mathcal{T}$ has to complete execution before the deadline time instant

$$S_{\tau_i} + \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} e_{\tau_i,p,m} \cdot W_{\tau_i,p,m} \leq d_i$$

- Each processor p cannot be utilized above its maximum alowed utilization u_p. For each $p \in \mathcal{P}$

$$\sum_{\tau \in \mathcal{T}} \sum_{m \in \mathcal{M}_p} \frac{e_{\tau,p,m}}{T} \cdot W_{\tau,p,m} \leq u_p$$

- A task may be scheduled for execution only after all of its predecessor tasks complete. For each dependent task pair $(\tau_i, \tau_j) \in E$

$$S_{\tau_i} - S_{\tau_j} + \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} e_{\tau_i,p,m} \cdot W_{\tau_i,p,m} \leq 0$$

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

$$S_{\tau_i} - C_{\tau_i} + \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} e_{\tau_i,p,m} \cdot W_{\tau_i,p,m} \leq 0$$

- By definition of a derived variable $V_{\tau_i,\tau_j,p}$, we have $V_{\tau_i,\tau_j,p} = 1$ if and only if $X_{\tau_i,p} = 1$ and $X_{\tau_i,p} = 1$. Since the variables are binary, this constraint can be expressed as

$$0 \leq X_{\tau_i,p} + X_{\tau_j,p} - 2 \cdot V_{\tau_i,\tau_j,p} \leq 1$$

- If the two tasks in a dependent task pair $(\tau_i, \tau_j) \in E$ are assigned to different processors, then the start time instant of τ_j is constrained by the completion of the communication of the output of τ_i. 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 leftmost element of the sum 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 leftmost element is zero and the communication time is taken into account. For each dependent task pair $(\tau_i, \tau_j) \in E$

$$C_{\tau_i} - S_{\tau_j} + \sum_{m \in \mathcal{M}_b} c_{\tau_i,m} \cdot W_{\tau_i,m} - Z \cdot \sum_{p \in \mathcal{P}} V_{\tau_i,\tau_j,p} \leq 0$$

- 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 the tasks τ_i and τ_j are allocated to the same processor ($V_{\tau_i,\tau_j,p} = 1$ for some p) and if τ_i executes before τ_j ($N_{\tau_i,\tau_j} = 1$). For each dependent task pair $(\tau_i, \tau_j) \in E$

$$S_{\tau_i} - S_{\tau_j} + \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} e_{\tau_i,p,m} \cdot W_{\tau_i,p,m} + \sum_{p \in \mathcal{P}} Z \cdot V_{\tau_i,\tau_j,p} + Z \cdot N_{\tau_i,\tau_j} \leq 2 \cdot Z$$

  Similarly, if τ_j executes before τ_i ($N_{\tau_i,\tau_j} = 0$) the constraint takes the form

$$S_{\tau_j} - S_{\tau_i} + \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} e_{\tau_j,p,m} \cdot W_{\tau_j,p,m} + \sum_{p \in \mathcal{P}} Z \cdot V_{\tau_i,\tau_j,p} - Z \cdot N_{\tau_i,\tau_j} \leq Z$$

- By definition of a derived variable B_{τ_i,τ_j}, we have $B_{\tau_i,\tau_j} = 1$ if and only if the tasks τ_i and τ_j are allocated to the same processor, if there exists $\tau_{i+1} \in \mathcal{T}$ such that $(\tau_i, \tau_{i+1}) \in E$ and the two tasks are not allocated to the same processor, and if respective relations hold for τ_j and τ_{j+1}. Since the variables are binary, this constraint can be expressed as

$$-2 \leq \sum_{p \in \mathcal{P}} V_{\tau_i,\tau_j,p} - \sum_{p \in \mathcal{P}} V_{\tau_i,\tau_{i+1},p} - \sum_{p \in \mathcal{P}} V_{\tau_j,\tau_{j+1},p} - 3 \cdot B_{\tau_i,\tau_j} \leq 0$$

- Since the bus is shared through a TDMA protocol additional communication constraint is that two transmissions from the same processor board cannot overlap. The following constraint is not automatically satisfied only in case $B_{\tau_i,\tau_j} = 1$ and N_{τ_i,τ_j}

$$C_{\tau_i} - C_{\tau_j} + \sum_{m \in \mathcal{M}_b} c_{\tau_i,m} \cdot W_{\tau_i,m} + Z \cdot B_{\tau_i,\tau_j} + Z \cdot N_{\tau_i,\tau_j} \leq 2 \cdot Z$$

Similarly, if τ_j executes before τ_i ($N_{\tau_i,\tau_j} = 0$) the constraint takes the form

$$C_{\tau_j} - C_{\tau_i} + \sum_{m \in \mathcal{M}_b} c_{\tau_j,m} \cdot W_{\tau_j,m} + Z \cdot B_{\tau_j,\tau_i} - Z \cdot N_{\tau_j,\tau_i} \leq Z$$

- The preallocated tasks generate the following constraint. For all $\tau \in \bar{\mathcal{T}}$

$$X_{\tau,p} = \bar{X}_{\tau,p}$$

- The premapped processors generate the following constraint. For all $p \in \bar{\mathcal{P}}_b$

$$M_{p,m} = \bar{M}_{p,m}$$

2.3 Objective function

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 of $\mathcal{P}_b$, i.e., by processors $\mathcal{P}$ and bus b. The simplest power model corresponds to the operation in which element of $\mathcal{P}_b$ remains in the same mode all the time, i.e., even when it is idle. In this case the total system power, written as a linear expression of program variables $\mathcal{M}_{p,m}$ and power coefficients $P_{p,m}$, is

$$P = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_p} P_{p,m} \cdot \mathcal{M}_{p,m}$$

However, to save energy, in a typical implementation, a processor or a bus goes into the sleep mode of operation, as soon as no task is ready for execution or communication. In this case, worst-case execution and communication times must be taken into account to compute duty cycle $D_{p,m}$ of a processor p in mode m and duty cycle $D_{b,m}$ of bus b in mode m. Recall that $P_{p,s}$ represents the power consumed in the sleep mode. The system power is given with the expression

$$P = \sum_{p \in \mathcal{P}_b} \sum_{m \in \mathcal{M}_p} (P_{p,m} \cdot D_{p,m} + P_{p,s} \cdot (1 - D_{p,m}))$$

where

$$D_{p,m} = \sum_{\tau \in \mathcal{T}} \frac{e_{\tau,p,m}}{T} \cdot W_{\tau,p,m} \quad \text{and} \quad D_{b,m} = \sum_{\tau \in \mathcal{T}} \frac{c_{\tau,m}}{T} \cdot W_{\tau,m}$$

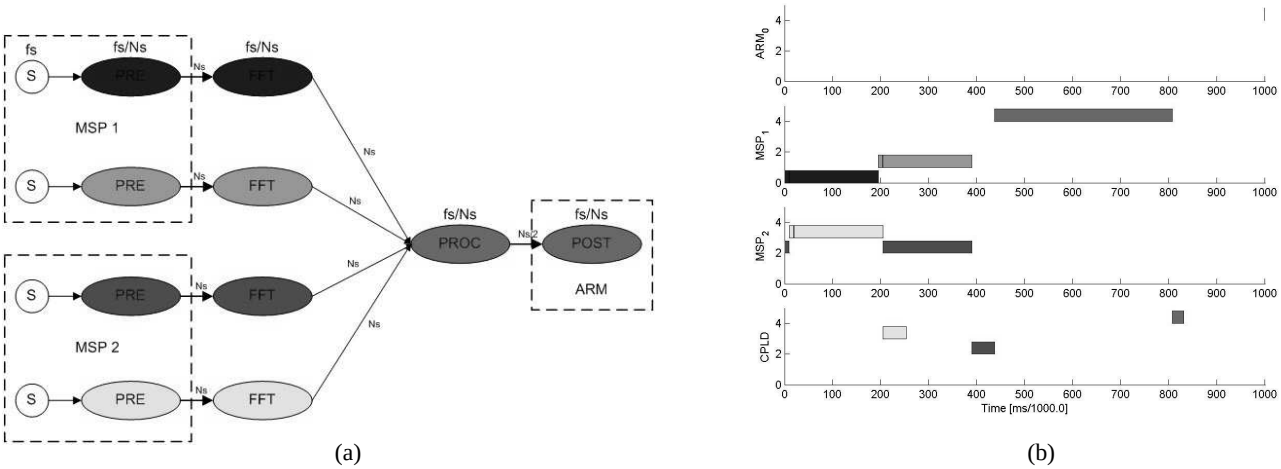
In the rest of the paper we assume the latter, more realistic, form of power model.

3 FFT Model

In this section we demonstrate the optimization procedure on a simple FFT-based application model. We first present model parameters, some of them measured in previous research, some only estimated to be as close as possible to the current m-platform implementation.

Two processor boards are currently developed for the m-platform: ARM - medium-power ARM7TDMI-based OKI ML67Q5003 board [7], and MSP - low-power MSP430-based TI MSP430F1611 board [6]. Note that the powers consumed by the two boards in their fastest active mode differ about 30 times (total current for ARM being about 70mA, and for MSP about 2mA). Future m-patform configurations will also have a high-power Intel PXA based board. Each board also contains a low-power CPLD [8] that implements a TDMA based shared bus. Let a stack of one ARM and two MSP boards be available for the application discussed in this section. So, according to the notation described in the previous section we may write $\mathcal{P} = \{ARM0, MSP1, MSP2\}$ and $b = CPLD$.

The maximum clock frequency of the ARM processor is 60MHz. A software controlled divider can slow it down 2,4,8,16, or 32 times [7]. Therefore, $\mathcal{M}_{ARM0}$ contains 6 modes defined by the operating frequency. For the values of $P_{ARM0,m}$ we use the results of extensive power measurements presented in [18]. Similarly, the MSP processor used in the current m-platform design can operate with 4 different frequencies, the fastest being 6Mhz [6]. The data for $P_{MSP1,m} = P_{MSP2,m}$ is



(a) (b)

Figure 1. (a) Task graph for the Sec. 3 example (b) Optimal solution, $N_s = 8$

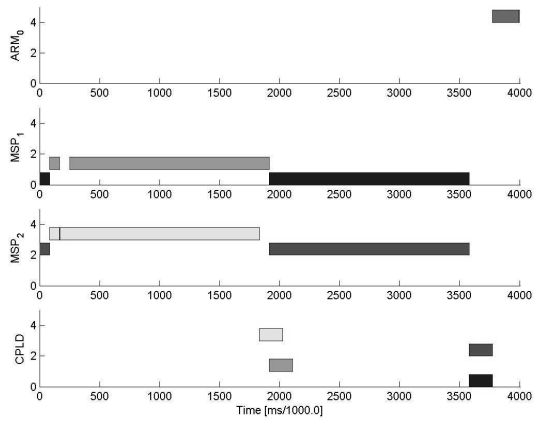
taken from [22]. Finally, the $CPLD$ bus is nominally clocked at 16MHz, but can also be slowed down to operate in one of the five different frequencies. This enables adjustments of the $CPLD$ power consumption according to the processor which it is connected to. Power data $P_{CPLD,m}$ can be computed from the curves given in [8]. In this example we assume that the utilization bound for each processor p is $u_p = 1$, i.e., we assume full processor utilization.

As shown in Fig. 1(a) the application task graph consists of 10 tasks, where PRE denotes preprocessing, FFT Fast Fourier Transform, PROC processing, and POST postprocessing tasks. The graph was constructed as a simplified version of a beamforming application task graphs used in [10] or [24]. The four audio channels are sampled (denoted with S) at the same time. The symbol f_s denotes the frequency with which data is sampled at each channel. For audio applications the value for f_s is typically taken from the range 4-8KHz. The symbol N_s denotes the size of the processing block of samples. For FFT applications it is equal to a power of 2, typically larger than 128. Under the assumption that every sample is going to be processed, the period of the task graph is equal to $T = \frac{N_s}{f_s}$. Even though it is not necessary, here we assume that the release time of all PRE tasks is 0, and that the deadline for each graph path, i.e., of the POST task is equal to T. Finally, the values for the task execution and communication times are determined from the values reported in research papers [14] and [13], and application notes [1] and [5].

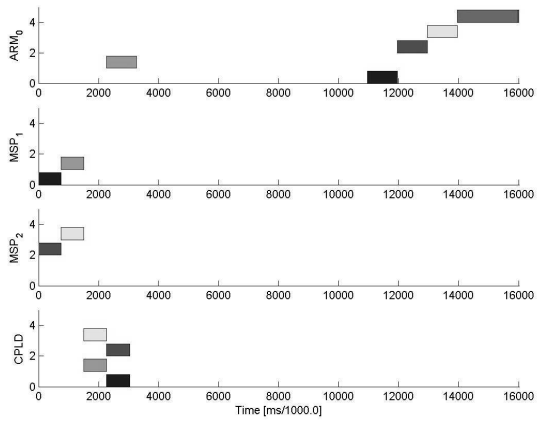
In this example we also assume that the m-platform implementation demanded preallocation of some tasks. Namely, as shown with dashed line in Fig. 1(a) the two PRE tasks are required to execute on the $MSP1$, the two other PRE tasks on the $MSP2$, and the POST task on the ARM processor. Using the notation described in Sec. 2 we may, for instance, write $\bar{X}_{POST,ARM} = 1$.

After all these application parameters are determined, the input of the problem is specified, and the constraints and objective are fully defined. We used the CPLEX ILP environment to solve for the optimal variable values, and then interpreted the core variables $X_{\tau,p}$, $M_{p,m}$, S_τ, and C_τ, for the allocation, mode mappings and the static schedule. We experimented with different parameter values and also with other task graphs containing up to 30 tasks. In most cases the optimization procedure takes 0-20 seconds on a 2GHz server running Windows XP. 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.

Fig. 1(b), Fig. 2(c), and Fig. 2(d) graphically show the optimal schedule for the cases when $N_s = 8$, $N_s = 32$, and $N_s = 128$, respectively. The graphs show how power-optimal task allocation (i.e., optimal value for $X_{\tau,p}$) changes by changing the sample block size N_s. The direct consequence of increasing N_s is increase in execution and communication



(c)



(d)

Figure 2. Sec. 3 example (c) Optimal solution, $N_s = 32$ (d) Optimal solution, $N_s = 256$

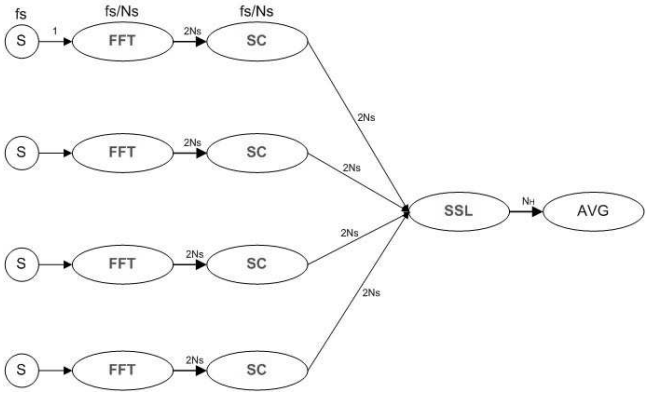
times of all tasks. We see that for small values of N_s, i.e., for the values of $e_{\tau,p}$ and $c_{\tau,p}$ small enough to be executed on an MSP processor, the optimal assignement tries to utilize the MSP's as much as possible. For larger values of N_s most of the tasks tend to be allocated on the ARM processor.

What the graphs do not show is the power-optimal processor-to-mode mapping, i.e., the optimal value for $M_{p,m}$. However, for this mapping the optimization procedure usually results in the fastest possible mode for all processors and the bus (of course, under the assumption that each processor goes to sleep as soon as it becomes idle). This is so because in this simplified application model that uses estimated or calculated power and timing parameters, the processor power in a mode is a linear function of mode frequency, and the execution times are inversely proportional to the frequency. This is true for the currently used m-platform processors, because they can only manage power through frequency scaling. However, this does not hold if a processor exhibits dynamic voltage scaling, as in the case of the Intel PXA processor. In such processors, DVS can reduce power consumption quadratically, while reducing execution speed only linearly. This also means that the same performance can be achieved with reduced power by the distribution of the computation load even using homogenous processors [24]. To conclude, if for the m-platform processors DVS is not an option, the power optimization has to leverage heterogenous character of the platform. This typically means to execute as many tasks as possible at low-power (MSP) boards.

4 SSL Implementation

In the final section of the report we describe the sound source localization algorithm, study its computational complexity, discuss its implementation, and analyze its performance on the m-platform ARM board. This is just an intermediate step in the project since the goal is to have an end-to-end local application that takes the advantage of both low and high power processors of the platform.

Sound source localization implementations are becoming common in teleconference, intelligent environment, and other applications with human-computer interactions. Several methods have been developed for source location estimation. The SSL algorithm studied here, the SRP-PHAT algorithm [11], uses an array of precisely positioned microphones to determine the location of an acoustical source. It is primarily designed for a single active sound source at a time. Compared to other algorithms, SRP-PHAT is simple and efficient, yet robust against noise and reverberation. For more complex algorithms see



(a)

Parameter	RingCam	m-platform
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

(b)

Figure 3. (a) Task graph for the SSL application (b) Parameter values in two SSL implementations

the discussion in [2].

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 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 $O(N_m)$ [2].

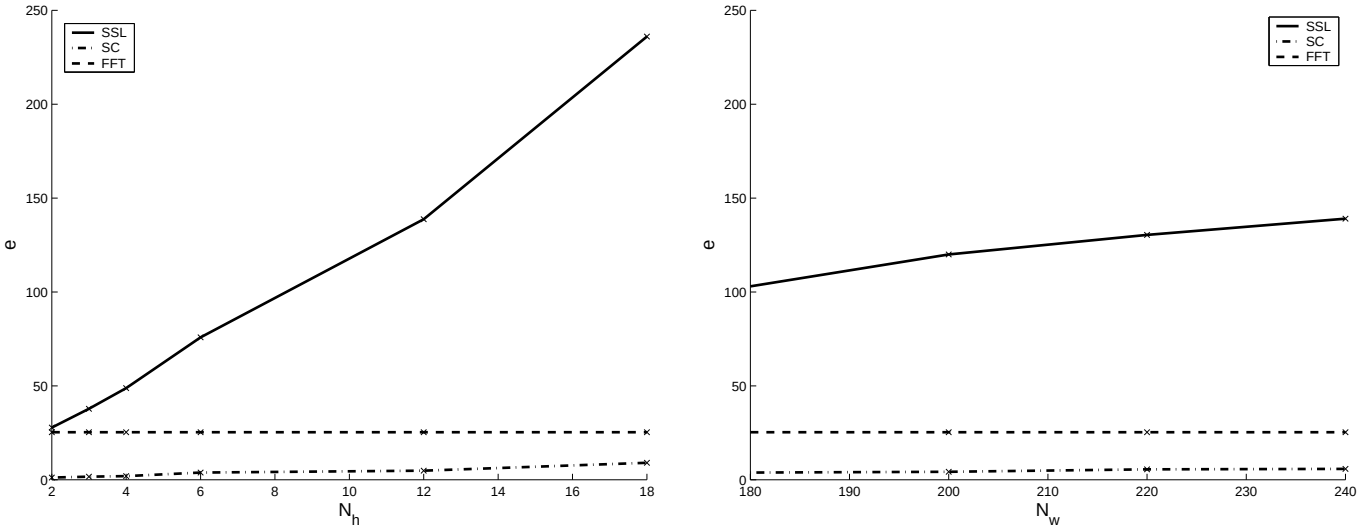
In practice, the maximization of the correlation function is achieved through hypothesis testing. Namely, multiple source location hypotheses are tested and the one that results in the largest correlation is declared as the source location. For teleconference applications the location is commonly represented in spherical coordinates, so each hypothesis corresponds to a spherical segment at a certain distance from the center of the scene. The selection of the number of hypotheses N_h is also important and directly affects computational complexity (see [21] 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. Fig. 5(a) shows the task graph of the SRP-PHAT algorithm. Shown for four audio channels, it resembles the task graph in Fig. 1(a). 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 the block is classified as a block of voice samples the SSL task is executed to determine source location through correlation maximization. In more complex algorithm variants the power level itself is used in the expression for correlation. The AVG task is executed only after a certain number of the SSL task executions. It perfroms the averaging of the location output of the SSL task.

We implemented the algorithm on the m-platform ARM board with the ARM7TDMI 60MHz processor, 512K of Flash ROM, and 32K RAM. The SRP-PHAT algorithm puts such an architecture close to its performance bounds. In comparison,

the similar algorithm was implemented in the scope of the RingCam project [12]. There, the system was also used for audio archiving, but the hardware consisited of two dual cpu (Pentium 4) 2.2GHz machines. To give a sense of the difference in performance of the two implementations we show the values of some SSL parameters in Fig. 5(b). One of the problems with the ARM solution is that the ARM7TDMI processor does not have DSP features [4]. The most serious drawback for signal processing is that the floating-point arithmetic is not supported in hardware. The application was implemented and tested using the ArmGnu suite of open tools. The new versions of the tool chain do not emulate floating-point operations, but instead have efficient software libraries. This difference alone can make the execution times of the SSL algorithm smaller more than ten times. Another drawback is that the DMA transfer available on the ARM board is not suitable for multiple-channel audio applications neither in burst nor in cycle-stealing mode [7]. Therefore, sampling had to be done through interrupts which, in our case, resulted in overhead of about 5%.

On the other hand, the parameter space of the SSL algorithm is large, which enables tuning the performance even for embedded implementations as in the m-platform case. 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 SSL-related parameters such as the number of microphones N_m, the number of hypotheses N_h, the classification threshold for the SSL execution, the period of the AVG task, and various noise related parameters for algorithm modifications. 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 the parameter values given in Fig. 5(b) the requirement easily sums up to 200KB. The RAM requirements are $O(N_{FFT} \cdot N_m)$ which may also be critical since the ARM board has only 32KB of RAM.



(a) (b)

Figure 4. Task execution times in μs as a function of number of hypotheses (a) and frequency window size (b)

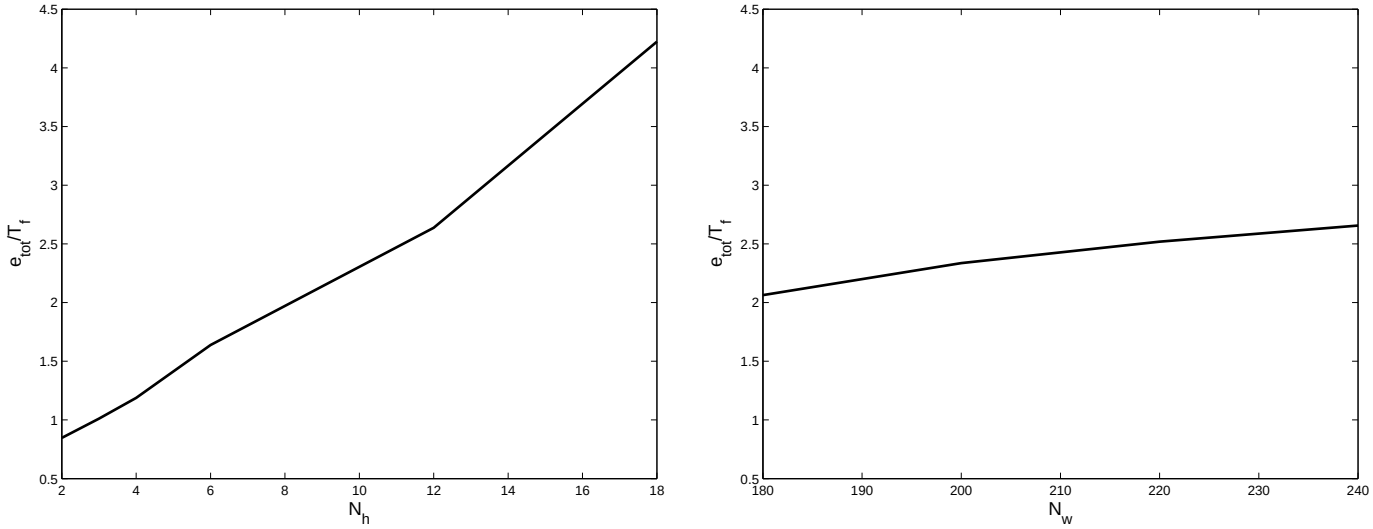
N_h	2	3	4	6	12	18
e_{SSL}	27.8	37.8	48.8	75.8	138.7	235.98
e_{FFT}	25.28	25.28	25.28	25.28	25.28	25.28
e_{SC}	1.15	1.62	1.92	3.8	4.8	9

(a)

N_w	160	180	200	220	240
e_{SSL}	92.67	103	120	130.4	139
e_{FFT}	25.28	25.28	25.28	25.28	25.28
e_{SC}	2.4	3.8	4.25	5.52	5.74

(b)

Figure 5. Measured task execution times values for Fig. 4



(a) (b)

Figure 6. Portion of samples that can be processed as a function of number of hypotheses (a) and frequency window size (b)

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 SSL. So, the dominant part of the time is required for the SSL task, which becomes even worse if noise correction algorithms are implemented ($O(N_m^2)$). 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. We measured the task execution times for the 60MHz mode, and studied the dependence with respect to application parameters. Fig. 4(a) and (b) show the results for the parameters N_h and N_w respectively. The values for the FFT and SC tasks are shown for all four task instances. The results confirm the linear dependence, which also enables ILP-based procedures if performance optimization over a certain parameter is desired.

Assuming the sampling frequency f_s=8KHz and sample block size N_{FFT}=512 the block of samples is collected in time $T_f = \frac{N_{FFT}}{f_s}$=64ms. When the total execution time $e_{tot} = e_{FFT} + e_{SC} + e_{SSL}$ 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 parameters. Namely, Fig. 6(a) and (b) show the ratio $\frac{e_{tot}}{T_f}$ 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 $\frac{e_{tot}}{T_f}$=2.6, which means that only every third sample can be processed. Although we did not implement the algorithm on the entire m-platform, i.e., distributed on both ARM and MSP processors, we have enough data about execution and communication times (c.f. [3]) to estimate the performance. If an m-platform stack consists of four MSP processors and one ARM processor, we would like to have each MSP executing FFT and SC tasks, and the ARM executing the SSL and AVG tasks. With the same values of the parameters as before the analysis shows that all processors can complete their load in about 140ms (including communication using $CPLD$). This results in $\frac{e_{tot}}{T_f}$=2.2 . However, this analysis assumes that the SSL task is executed in every period T_f which is not the case, since some blocks of samples are classifed as noise.

References

[1] Application Note 16. Implementing Fast Fourier Transforms on the ARM RISC Processor. Document Number: ARM DAI 0016c. Advanced RISC Machines Ltd (arm), 1996.

[2] Cha Zhang, Zhengyou Zhang and Dinei Florencio. A Maximum Likelihood Framework for Multi-microphone Sound Source Localization. Submitted.

[3] Dimitrios Lymberopoulos, Bodhi Priyantha and Feng Zhao. m-platform: A Flexible and Efficient Architecture for Sharing Data in Stack-based Sensor Network Platforms. MSR Technical Report ???

[4] http://www.bdti.com/procsum/arm7.htm.

[5] MSP430 Family Mixed-signal Microcontroller Application Reports. Literature Number: SLAA024. Texas Instruments, 2000.

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

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

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

[9] 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.

[10] Pierpaolo Bergamo, Shadnaz Asgari, Hanbiao Wang, Daniela Maniezzo, Ralph E. Hudson, Len Yip, Kung Yao, and Deborah Estrin. Collaborative sensor networking towards real-time acoustical beamforming in free-space and limited reverberance. *IEEE Transactions on Mobile Computing*, 3(3):211–224, 2004.

[11] 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.

[12] 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.

[13] Anand Eswaran, Anthony Rowe, and Raj Rajkumar. Nano-rk: An energy-aware resource-centric rtos for sensor networks. In *RTSS*, pages 256–265. ACM Press, 2005.

[14] Ben Greenstein, Alex Pesterev, Christopher Mar, Eddie Kohler, Jack Judy, Shahin Farshchi, and Deborah Estrin. Collecting high-rate data over low-rate sensor network radios. in CENS Technical Report 55.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

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

[24] Alice Wang and Anantha Chandrakasan. Energy-aware architectures for a real-valued fft implementation. In *ISLPED*, pages 360–365. ACM Press, 2003.

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

[26] 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.