UNIVERSITY OF CALIFORNIA

Los Angeles

Scheduling for Energy Aware Wireless Sensor Applications

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Science

in Electrical Engineering

by

Sridhar Sesha Vemuri

2004

The thesis of Sridhar Sesha Vemuri is approved.

William J. Kaiser

Kung Yao

Gregory J. Pottie, Committee Chair

University of California, Los Angeles

2004

To my parents…

who worked so hard all their lives

to see their children do well…

I hope I fulfilled a part of their dreams.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acknowledgements

I would first express my most sincere gratitude to Dr. Gregory J. Pottie for considering me eligible to be a part of his research group at UCLA. I started my work in the summer of 2002, just to get acclimatized to the nature of the research and to the attributes needed to deal with various challenges. Dr. Pottie has an amazing temperament and approach to his research group (especially to newcomers) and I personally feel that it worked out great for me. He gave me all the freedom and time to explore previous research works, digress into unknown and unrelated fields and even to take it easy when under course pressure. My coursework and research were in two unrelated areas so that kind of a support was the most I could have hoped. His insight in my field of research and his invaluable help and suggestions made it possible to finish my thesis work. It cannot be emphasized enough the amount of patience and understanding he displayed whenever I interacted with him. In short, his mentorship propelled me to a great extent and I thank him sincerely for the faith he had in my abilities in addition to the financial assistance.

Dr.William Kaiser had been equally responsible in ensuring that my research work progressed consistently despite constraints related to implementation issues and research requirements. Despite his busy schedules and the fact that he mentored in many areas in his research group, he always made time to listen and appreciate what our team had to offer. In my observation, all the students who work under him enjoy two things thoroughly- his keenness to understand what you have to offer and more importantly, his words of encouragement irrespective of how much you have achieved. I really consider

myself fortunate for working under these two very amicable and understanding people. In addition, I would thank Dr. Kung Yao for being in my committee to review my thesis. I am glad that he took interest in my research work and agreed to serve as a committee member.

I could not have completed my part of the work without the valuable help and advice offered along the way by my team-mates Ashutosh Verma, Winston Wu who I began with, and by Anitha Vijayakumar and Dustin McIntire, who joined in later. Ashutosh and Winston were always available to assist me in my algorithm implementation and improve the thought process. Their mastery over various aspects of the research work greatly enabled smoothness in my part. I wish to thank Anitha sincerely for contributing to my work by improvising my code and providing some important calculations to ascertain performance of my work. Dustin has always been supportive in rendering ideas and has appreciated the present work done, despite his busy work schedule.

I also take this opportunity to thank Ameesh Pandya for being of valuable help since the beginning of my graduate work. I really appreciate all of his efforts in letting me know of the various research endeavors in Dr.Pottie's group. In addition, his help whenever needed either in coursework or for some research-related suggestions will always be remembered. I offer a special note of gratitude to Srikanth Gondi, who has always shown a big brother affection and support in times of need. I cannot remember a more helpful person than him.

I was fortunate to have two very understanding roommates in Charanjeet Singh and Bravish Mallavarapu. I never missed out an opportunity to load them with any worry or concern I had and it was just amazing to receive such support from them in their own unique ways. Their intellectual guidance would always be appreciated and I am sure I learnt something valuable from both of them during the last two years.

During the past seven years of my educational pursuit in America, I was blessed with many friends and well-wishers who always supported me in my success and failures. I would acknowledge all my friends in San Jose especially Vasudeva Arramreddy, Azeem Karmally and Vikas Singhal for being trustworthy over the years. From my undergraduate and graduate years at UCLA, I would thank Arpit Antani for being a loyal friend in addition to being an extremely helpful and understanding project-partner in most of my graduate courses. In addition, I would thank Niket Sourabh, Deepak Alagh, Nitin Gambhir, Gaurav Ahuja and Rohini Reddi for their support and friendship over the course of my stay at UCLA.

My earliest friend in graduate school, Parul Gupta will never be forgotten. She is one of the most affectionate and understanding people I ever came across. I will never stop to appreciate the support she rendered as the truest of friends. Her wisdom and brilliance always astonished me but more importantly, she was always available to share her thoughts on every issue I presented to her. I am fortunate that I had such precious friends in my life and I really pray to God that they remain in such a way.

I cannot finish this section without making a special mention of all the family members who have been there since my childhood and have been responsible for my growth. Firstly, I would like to thank the late. Mrs. Kamala Paritala, my aunt for making sure our family had a secure future by sponsoring our immigration into the United Sates. I would always appreciate my cousin Sriniketh Nagavarapu and his dad Mohan for their initial support in the establishment here. Sriniketh has been a tremendous influence on me and always been a motivation by being the one of the most modest, helpful and amicable people I ever met. I would like to thank the Galivanche family, the Vedantam family and the Vadlakonda family for being the most helpful to me for as long as I could remember. Without their support, I could not have come this far in my life.

I made sure I included all the people I wanted to and if it had been possible more, in my acknowledgements. It is because all of them contributed significantly towards my present state- attaining a Masters degree from a reputed institute such as UCLA and enabling a secure future. In this process, the most important contribution in terms of emotional support and presence has been from my mother Mrs. Vimala Vemuri, my father Mr. Upendra Vemuri, my brother Srinivas Vemuri and my closest friend Sumanth Kodury. I would like to take this opportunity to thank my brother for being there during the tougher situations in my life and for a timely financial assistance. It is true that I can never repay my parents for all the sacrifices they made in their lives to ensure their two children have a secure future. The least I could do is to dedicate my thesis to them.

Sridhar Sesha Vemuri

ABSTRACT OF THE THESIS

Scheduling for Energy Aware Wireless Sensor Applications

by

Sridhar Sesha Vemuri

Master of science in Electrical Engineering

University of California, Los Angeles 2004

Professor Gregory J. Pottie, Chair

Present day wireless sensor platforms must support applications such as imaging, networking and security. Challenges that accompany such applications include complexity in signal and image data processing, high data rate in the wireless networking and supporting a fully secure system. In addition to the problem of supporting a suite of resource intensive and computationally burdensome sensor applications, there is also a need for continuous vigilance in unattended environments. The focus of this thesis is a resource-scheduling algorithm that enables a novel distributed processor platform to operate at the optimal point in energy while catering to such a class of wireless sensor applications.

CHAPTER 1

Introduction

1.1 A brief overview on Sensor Platform Requirements

Present wireless sensor applications have extended in scope and importance from simple day-to-day utilities like temperature and pressure measurements to complex exercises such as environmental and land surveillance, health monitoring and battlefield security [7]. An increase in the complexity of applications correspondingly brings about an increase in measures such as implementation costs, time to establish results and most importantly, in a sensor system's perspective, the energy required to cater to the very broad range of applications suite. These range from computationally intensive applications such as image sensing and processing to resource intensive applications such as wireless networking. These applications demand a complex, fully functional operating system thus enabling a secure platform for wireless sensor architectures. However, such an environment can only increase power requirements. There is a factor of 1000 in the ratio of the power needed to execute such an application set to what is typically supplied by a battery that is used to power up the sensor devices and the various hardware resources on the backend processing. This would essentially widen the energy gap between the energy demanded and what is present.

A specific example is in the context of NIMS or Networked Info Mechanical Systems, an area of wide interest in the sensor network community. These are mobile, distributed nodes that are deployed into the environment to perform image sensing and processing, networking and other resource intensive applications which demand power at the order of milli-watts. Thus, large batteries are used to supply power to the each of the distributed set of nodes. In short, the energy problem is one of the most important issues presently tackled by the sensor network community.

1.2 The Wireless Sensor Architectures

1.2.1 Previous Architectures

Previous architectures used for wireless sensor applications have been discussed in the literature. Most of them are equipped with a micro-controller that can display variable power modes, memory for holding the memory data and a typical sensor suite. Each component consumes a fraction of a Watt of power through the use of these low-power microcontrollers. However, there is a trade-off seen in the architectural complexity vs. the system applicability. The complexity of the algorithms executed on such platforms is fairly low thus, restricting their usage to specific applications. For example, [6] describes the Mica Mote wireless sensor system has been developed in UC Berkeley. It is an example of the unification of sensing, processing and wireless networking in a single package. The operating system that runs on the Mote, called the TinyOS, supports

components that provide efficient network interfaces and which perform with little processing and storage overhead. The reader is encouraged to read [6] to understand the design features of TinyOS. However, the Mote's processor, which is an 8-bit, 4 M-Hz Atmel processor, cannot perform floating-point arithmetic operations [9]. As an experiment, by adding system features into TinyOS, emulations were performed to compute and compare the execution times of computationally intensive algorithms. It was found that the execution time of FFT algorithms took tens of seconds, which could have been performed in far less time on a dedicated microprocessor running a fully functional operating system.

On the other hand, there have been distributed-processor wireless platforms implemented which have been successful in catering to the low-power requirement while performing computational and resource intensive applications. For example, [2] and [10] discuss the Infopad, a multimedia terminal from the University of California Berkeley. The applications that are executed on the Infopad consist of computationally intensive applications such as speech, video and graphics applications. All the application related computing is performed on remote servers. The components of the platform consisted of low power ASIC's with embedded processor cores and other general-purpose microprocessors. Wireless sensor platforms that support hierarchical and modular hardware architecture and which support a complex operating system for secure transmission of data are desired. We integrated such a platform similar to the above-discussed concept with Commercial Off the Shelf (COTS) units, where the power

intensive operations could be executed on backbone servers. In addition, the platform can drive towards the lowest energy operating point by a resource scheduling mechanism we have implemented. In addition, the advantages are yielded by the processor-preprocessor architecture while dealing with periodic applications with specified duty cycles.

1.2.2. Our Proposed Architecture

We have implemented the Smart Objects Platform, which is based on the principles of hierarchy in processing. The platform supports Linux, which is a fully functional operating system supporting security features necessary for specific applications. The hardware resources can be portioned into a processor-preprocessor architecture, wherein the processor is a power-intensive embedded unit that executes whenever a preprocessor triggers an event and requests services from the processor. Periodic applications such as monitoring and detecting changes can be effectively performed on the preprocessor modules. In addition there is a power-management preprocessor, which is essentially a micro-controller. There is a clear hardware/software partition on the platform as desired by the embedded systems community that governs the hierarchy on the platform. In addition, the platform uses a dedicated scheduling algorithm that aims to provide an optimal schedule to run the various hardware resources in both periodic applications and when event-driven applications arise.

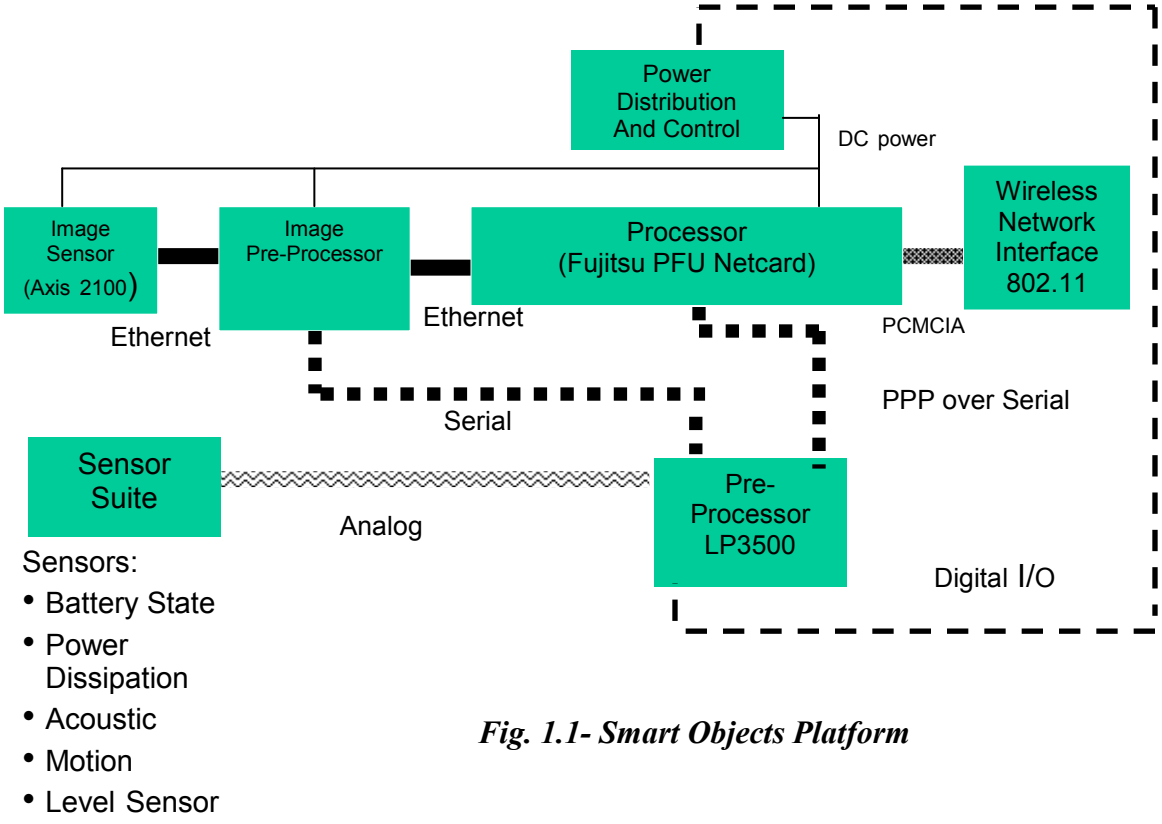


Fig. 1.1- Smart Objects Platform

1.3 An Overview of Scheduling

1.3.1 Kinds of Scheduling in Research

There have been many scheduling algorithms at various levels in embedded systems. Operating systems have their own scheduling mechanisms. Often heuristic approaches are employed to achieve a near optimal solution in scheduling algorithms, as task allocation and scheduling problems are known to be NP-complete problems in time [3]. Researchers have used scheduling techniques to optimize different parameters for applications such as deadlines, execution time of the entire schedule or the schedule

length and energy consumption, all of which can be represented as a cost function. The results of these algorithms do provide globally optimal solutions in certain cases while providing near-optimal solutions in other cases pertaining to the factors like the time to execute and the urgency of the schedule requirement. The two main optimization parameters that follow in our discussion are the net energy consumed and the deadlines requirements for the set of periodic applications. The energy-optimization based scheduling is described first followed by scheduling to achieve deadlines.

The standard equation for power consumed in a CMOS integrated circuit is given by:

$$P=\alpha*C_L*V^2_{dd}*f \quad (1.1)$$

where C_L at the CMOS gate, V_{dd} is the supply voltage, f is the clock frequency and α is the switching activity, which is defined as the probability of switching from an ON state to an OFF state during a clock cycle. Changing one or more independent variables on the right hand side of the equation affects the net power consumed.

A popular kind of power scheduling called voltage scaling is explained in detail in [1] and [8]. In this approach, the supply voltage allocated to each resource on a hardware platform can be controlled. This will have a large effect on the power due to the quadratic dependence on the supply voltage, V_{dd}. In addition, adjustment of the clock frequency factor, f can lead to substantial improvements in energy savings. Provided that hardware resources can operate in a variable clock environment, the clocks assigned to various

hardware resources can be slowed down or even shut off when the particular resource is not in use. These approaches form a part of dynamic power management.

Another class of scheduling algorithms deals with task allocation to satisfy deadline requirements provided by the application developer in addition to catering to the low energy requirement. The algorithms proposed in the literature are dynamic in nature, such as the Earliest Deadline First (EDF) algorithm, which assumes resource sufficiency i.e. all the set of tasks are schedulable irrespective the unpredictability in their times of arrival. This approach suffers a heavy degradation in the presence of a heavy overload [1]. Improvements over this approach are described in [1] and the algorithm is modified into the Slacked Earliest Deadline First (SEDF) algorithm, which asymptotically converges to the performance of the EDF algorithm. There have also been heuristic algorithms that provide globally optimal solutions in certain cases while providing near-optimal solutions in other cases pertaining what the requirements are. For a complete understanding of certain heuristic scheduling techniques such as Earliest Deadline First and it variations, the reader is encouraged to refer to [1]. However, in the case of wireless sensor platforms, where power optimization is the key concern, it is desired to operate near the minimum point, which might be achieved only through an exhaustive search of all possible schedules.

1.3.2 Our Scheduling Approach and Requirements

The scheduling problem considered in our research has been to optimize the global energy on the platform to perform a series of tasks over a period of time while enabling

applications to complete within a specified deadline. The applications to be executed provide additional constraints such as the period of operation, their execution times and a tolerance around which they can be executed. Optimal solutions are desired, which lead to the implementation of an exhaustive approach. The overview and details of our scheduling algorithms are explained in detail over the subsequent chapters. Some of the main features of the different approaches considered for a simple 2-application scenario and in general, for a multiple application case are mentioned below.

The first method proposed was a constrained approach where two periodic applications with specified duty cycles were considered. The scheduling procedure was based on a greedy approach wherein instances of applications whose period multiples mapped partially or completely were scheduled first. This maximum overlapping of processor execution times would save the energy related to the transition costs (shutting down and booting back again) of all the common resources. The remaining instances of these applications have situations where energy savings could not be achieved because they have to be executed in discrete time frames. Although this approach gave an initial intuition on scheduling for periodic applications and also converged to a solution in a very short execution time of the algorithm, the approach was not extensible in the case of a multiple application scenario.

In order to cater to the most general case while guaranteeing a globally minimal energy operating point, an exhaustive approach was considered, which essentially considers all

possible combinations of potential start times of all the instances of each application. The scheduler matrix created out of each such combination is tested for the optimal energy case. There were factors that influenced the exhaustive search space considered to trim the execution time of the search algorithm. The details of this approach are explained in chapter 3 of this thesis.

Finally, a certain kind of a variational approach involving hierarchical notions is also mentioned. Pre-computed schedules of already assigned applications could be utilized and new applications that ought to be scheduled could be best fit in into these optimal schedules. Although this approach would not guarantee an optimal solution, the advantage clearly is that the complexity of the search algorithm is always maintained at $O(n^2)$ i.e. that of a 2-application case. In addition, a good solution can always be worked out given earlier hierarchies of computed schedules.

1.4 Outline Of the Thesis

The remaining chapters of this thesis are organized as follows. In chapter 2, we present a brief architectural description of the Smart Objects platform. The various parameters for an efficient scheduling are discussed. A heuristic scheduling algorithm is described for a simple 2-application scenario, which gives an overview of the general concept of application scheduling. In chapter 3, we extend the scheduling algorithm into an optimal exhaustive-search approach to cater to any multiple application specification. The various factors that affect the optimality and execution time of the algorithm are discussed in

detail. Various test vectors that were validated by hand calculations earlier are tested in the simulation for correctness. Some results and plots that explain the characteristics of the scheduler and describe graphically the advantages of scheduling follow the test vector analysis. In the last chapter, we provide a conclusion and discuss the scope for future work.

CHAPTER 2

Smart Objects Platform and Scheduling Overview

2.1 Platform Description and Typical Application Suite

The Smart Objects platform has a defined hardware architecture that consists of a power-intensive processor unit and a series of preprocessor units whose functionalities are described during the course of this chapter. Application developers publish resource requirements, which are constituted from the set of the various processing and preprocessing units in the environment. The scheduler module observes application duty cycle, deadline demands and tolerance values. It then computes an energy-efficient operation schedule that meets the task and resource requirements at minimum energy consumption. The micro-controller preprocessor, LP3500 manages power and the platform management episodes through the schedule vector transferred to it by the scheduler. This chapter presents the scheduling mechanism that runs as an a-periodic application on the power-intensive processor. The demonstration of an energy-efficient environment is done through this efficient scheduling of the registered applications according to their resource pools. The scheduler algorithm can be best viewed as an application as it also utilizes resources such as the PFU processor and the pre-processor LP3500.

The applications that ought to be demonstrated in this distributed energy-aware wireless sensor environment include computationally intensive applications such as object tracking and image processing and resource-intensive applications such as networking and sensor node discovery. In addition, co-operative applications such as acoustic beamforming can also be demonstrated.

2.2 Scheduler Algorithm Stages

2.2.1 Registration

The initial prototype of the environment has a remote node networked with the Smart Object node that sends in the configuration files representing each application. The scheduler application has its own internal configuration file, which it reads along with the others. The first stage of the algorithm is the **registration** stage (Fig. 2.1), wherein it reads the published resources needed for each application. Each of the resources is constituted from the imager (camera), the imaging preprocessor, the PFU processor (for executing computationally intensive applications) and the wireless LAN unit (for information transfer across nodes and for discovery). The application descriptor files are transferred to the Smart Object node from a remote "application" node through a secure file transferring protocol. The files contain information such as the executables for each application, the approximate execution times for each application, the duty cycle around which the applications have to be performed, a tolerance value that measures the

feasibility level of deviation from the true period and most importantly the resources required for execution. The application developer publishes all these parameters in the respective configuration files of each application. Old descriptor files are deleted after resource allocation is provided and new applications that were queued previously would be scheduled to achieve a new optimization scheme. If no new applications exist, then the present scheme would execute unless prompted by any event that requires an immediate usage of the system resources. The entire environment runs on a Linux platform and the concept of parameter passing through signals is utilized in the demonstration. The Linux file systems are utilized to write all the required parameters of each application. This approach provides a simpler means of communication that we have extended to the scheduling of multiple applications.

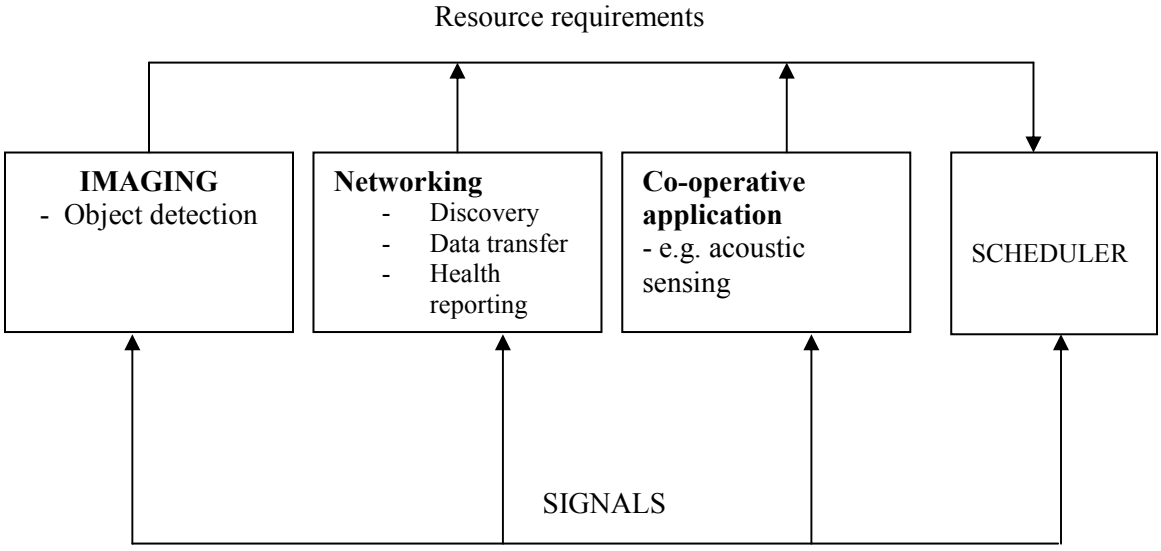


Fig. 2.1- Registration stage of the Application set

2.2.2 Optimization for a Static Scheduling Scenario

2.2.2.a Theoretical Details

Once the resources for a particular set of applications are obtained, they are passed onto the **optimization** stage of the algorithm. The scheduling optimization algorithm runs in a variable time after the initial registration has taken place. During this time, the resource requirements are considered for each separate application. The idea is to complete as much preprocessing work as possible for each application so that applications can utilize the power-intensive processor within a common time frame. It is experimentally verified that the power consumed by the PFU while running simultaneous applications is insignificantly different than running separate applications. Once the power optimization algorithm determines at what time frames each processing unit is set to work and which are shutoff during those times, a matrix table is created which is then passed into the LP3500 (the power-management preprocessor) to read the table sequentially and operate. Whenever the PFU processor is ON, an updated scheduler table is copied onto the LP3500, which caters to both situations; a new application is added or that no new application occurs (wherein the table remains unchanged). The algorithm for the creation of the scheduler vector itself takes in the physical attributes of each of the processing and preprocessing units such as the boot up/shutdown times, the wake/sleep times, the processing times of the applications on the respective processing units and most importantly, the power dissipated through each of the processing units in different modes.

The optimization algorithm in its simplest version considers the above-mentioned attributes and resource parameters and assigns begin and end times for each application in its respective duty cycle frame. The scenario that is to be demonstrated on the platform is a periodic surveillance of the environment for routine versus interesting objects through simple differencing between images and if required, template matching. The frequency-domain template-matching algorithm is computationally intensive and can be demonstrated efficiently on the platform. The optimization algorithm results in a resource management mechanism (in the form of a matrix that describes the states of each resource at various time frames) transferred from the power-intensive processor to the low-power power-management preprocessing unit. The schedule matrix displays the resources to be switched on and switched off appropriately. Resources common to the applications are dealt with in accordance to when they are required to be switched i.e. depending on the smallest begin time of the applications. The time axis should be considered to be granular by creating smaller time ticks to ensure the various processing units start and end as close as possible to the determined values.

2.2.2.b A Heuristic Approach for a 2-application Scenario

The algorithm is described by an interval partitioning method wherein the duty cycle frames of each application are matched into three different cases and scheduling of applications into appropriate execution times are done individually for these three cases.

Case 1: The current duty cycles are such that one frame is embedded in the other frame.

Case 2: A partial overlap exists between two duty cycle frames.

Case 3: The current duty cycle frames are completely exclusive of each other.

In cases 1 and 2, all resources that are common to both applications have to be managed such that the tasks are handled in parallel when such resources are in the execution mode. In other words, it makes sense to schedule tasks while making sure that the particular resource is back to shutdown/idle mode only after the completion of the scheduled tasks. In case 3, the scheduling has to be done individually to all the frames that have not been dealt with in either cases 1 or 2. Fig. 2.2 shows the demonstration for a typical two-application scenario. It can be seen that scheduling of the two applications is done such that the partial and complete overlapping cases are utilized extensively and the remaining unscheduled frames are scheduled independently. However, if power optimization can be

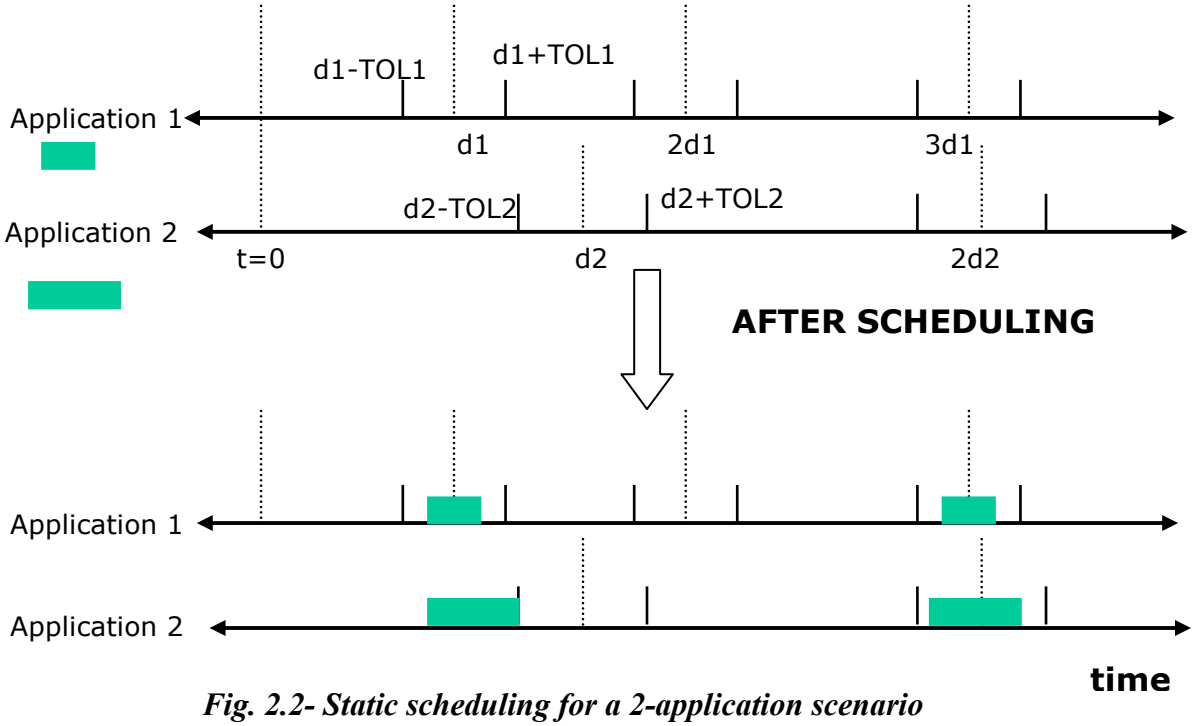


Fig. 2.2- Static scheduling for a 2-application scenario

achieved by maintaining resources to be in idle as opposed to a complete shutdown, the algorithm searches for all possibilities and determines the states of the resources (when not utilized) such that the net power over the desired sets of duty cycle frames is minimized. The above process wherein scheduling of applications takes place before the actual activation is termed as **static or off-line** scheduling. The schedule for the entire sequence of task execution (as in the periodic surveillance scenario given duty cycle and tolerance values) can be conveniently stored in a vector table.

2.3 A Brief Overview of Dynamic Scheduling

The present section deals with scheduling of tasks, which form a part of **dynamic scheduling**. There are two kinds of scenarios that constitute dynamic scheduling. The first case is that of *event detection*- a case wherein resources are woken up immediately in order to verify a noticeable disturbance in the environment. The other case is that of an online scheduling of tasks where new application registration and scheduling decisions are to be taken at run-time on the set of active tasks. The scheduling of multiple tasks on the distributed processor platform is explained in detail in the next chapter. In the present section, the phenomenon of event detection is explained along with the communication interface between the processor and the power-management pre-processor.

The scenario of periodic surveillance of the environment is demonstrated effectively on the platform through the optimization scheme proposed above. However,

there might be instances where simple differencing of images is not sufficient to determine if there was any significant change in the environment. The arrival of a new object in the environment creates a significant change, which would be detected by the motion sensor. The infrared sensor communicates directly with LP3500 in case of detection of a new object. The networking application is invoked to obtain templates for the frequency-domain template-matching algorithm to execute. An action is then taken depending on whether the new object is interesting or not. However, there are some additional cases that do not constitute event detection. For example, there might be instances where the motion sensor fails to detect the presence of a new object if it appears either too slowly or too quickly. There might also be an instance where the sensor detects a moving object that appears and disappears from a scene. In this case, networking need not be invoked since there was no change in the scene. Such cases do not count as the occurrence of an event.

In case of an event occurrence, all the required resources, which are in idle, sleep or suspend states, are immediately woken up. In the present architecture, the power intensive PFU processor has idle, a cold shutoff/boot up and suspend/wake states whereas all the remaining pre-processing units have a hard shutdown and boot up states. Once the IR sensor communicates to LP3500 about the presence of a new object, the preprocessor halts the present execution of the schedule vector transferred by the scheduler and wakes up the needed resources (if not awake already) to begin the necessary application. Once the required application is executed, the PFU can go back to the sleep state or continue

executing other applications depending on the command issued by the power-management pre-processor, LP3500.

2.4 The Communication Interface

The communication interface between the LP3500 and the PFU should be made as minimal and yet as flexible as possible. As mentioned earlier, the initial prototype of the platform has the components talking through file systems. There are two types of commands issued from the LP3500 side to the PFU side. The interface has as its parameters, the schedule matrix (determined from the optimization algorithm) and a file to read sent from the LP3500 side. This file contains a value that enables the interface to read the schedule matrix and execute either of the two commands- an EXE command that indicate the schedule to be followed by the various processing units when the processor is ON and an SCH command, which indicates what schedule by the other processing units when the processor is OFF. In other words, the interface function presents a convenient mechanism by which the representation of the schedule matrix (through a huge sparse matrix of 1's and 0's) is translated into a form that can be viewed as one of two commands executed by the various components of the platform. The command is written from this interfacing function into another file that is accessed by the LP3500 processor and executed accordingly.

2.4.1 Description of the Scheduler Matrix

For a clear communication between the Scheduler algorithm and the interface, the tabular matrix is represented as follows.

1. A time index is the first column of the matrix, which can be synchronized to the real-time on the platform either directly or through some scaling factor.
2. The next eight columns represent the “hardware bits”. Presently, there are three bits whose states can be toggled between 0 and 1. The first bit corresponds to the ON/OFF state of the PFU processor. The third bit corresponds to the ON/OFF state of the AXIS processor and the fourth bit corresponds to the Imager. The remaining hardware bits are presently inactive. For the future, the SUSPEND state of the PFU processor is to be integrated into the platform. It is to be noted that some of the hardware bits are “don’t care” bits, which have been set to 0.
3. The next eight bits corresponds to the time indices where a particular application or a group of applications start, beginning from application 1 onwards. The interface function scans the tabular matrix for all the positions where a particular application starts and informs the LP3500 to start the specific hardware resources and in addition execute the appropriate software executables that constitute a particular application.

As an example, as shown in fig 2.3, a row of a sample schedule matrix at time t=100 is shown along with the various hardware bits.

Time Index	PFU (ON/OFF) BIT 1	PFU (suspend) BIT 2	AXIS (ON/OFF) BIT 3	IMAGER (ON/OFF) BIT 4	BITS 5-8
100	1	0	1	1	DON'T CARE bits

Table 2.1- Description of the hardware bits of the tabular matrix.

With such a tabular description, it not only becomes easier to communicate with the interface but also the total energy of the platform is easier to calculate. The total energy of the system is the sum total of the boot, the execution and the shutdown modes. The table is scanned and the net energies of each resource in different modes of operation can be accumulated to determine the total energy of the platform. The detailed description of the scheduling function and the interface function can be viewed through the c-files, **scheduler.c** and **schexe.c** respectively. These basic models for the schedule optimization and the interface can be extended to include online scheduling of events. A generalized optimization algorithm that is developed for an arbitrary number of applications based on a pure exhaustive search is discussed in detail in the next chapter.

CHAPTER 3

Scheduling for Multiple Applications

3.1 A Generalized Approach

There is often a greater demand to be able to perform a larger set of sensor applications on the Smart Objects low-power platform. The sensor application suite typically includes network discovery, peer discovery, event detection and scheduling. Each of these applications has their own application descriptor files. Initially, the Smart Objects interface observes that there is no schedule vector present. It then downloads a schedule from a server, which has been set up specifically to share the computational burden. The scheduler algorithm that is executed on the remote server is the most robust method for the energy optimization scheme on the platform. It performs an exhaustive search on all the possible schedules of the applications to find an energy minimum. The schedule that determines the lowest possible energy on the platform over the entire application episodes is decided to be the optimal schedule. There is a computational complexity vs. energy optimality tradeoff involved in such an exhaustive method. In order to perform a complete search, it is imperative to perform the search algorithm offline on another node specifically set up for the computational burden, especially when the application set is large (i.e. >= 4 applications). This would also enable the platform to be devoid of any computational complexity for long periods of time, thus supporting the low-power capabilities.

The advantage of the exhaustive approach is that there will not be any computational stress on the low-power platform. However, this approach would not be feasible whenever a new application with a very high duty cycle needs to be registered into the application pool and needs to be executed. The platform has to wait for the optimization to be finished in order to accommodate the new application, leading to an over dependence on a remote unit. Furthermore, the network resources on the platform are also burdened to communicate with an offline server to transfer the schedule. To counter this, approaches involving scheduling hierarchies can be employed wherein old optimal schedules for the previous application set are maintained and a new application is accommodated (according to its own constraints) into the existing schedule in such a way that the net energy on the platform remains optimized. The advantage of such a scheme is that the algorithm complexity would remain at the complexity of a 2-application case. Moreover, the platform can slowly adapt to the lowest operating energy point through a more complete search starting from the initial estimate, which might be a coarse evaluation. In addition, certain other techniques would result in the narrowing down the search space for task allocation. To be more specific, placing the constraint of the deadline values before the exhaustive procedure reduces the search space. This would essentially guarantee a search through a smaller space of possible schedules. The tradeoff however is that it might also overlook better possibilities that were eliminated during the determination of the near-optimal scheme. A brief explanation of the hierarchical approach is given in the next section. However, the exhaustive approach is considered in all our simulations and the results are discussed in the next chapter.

3.2 kinds of Multi-application Scheduling

3.2.1 The Exhaustive Approach

The complexity of the optimization scheme is clearly in polynomial-time with a degree of the number of applications. The algorithm implemented as an exhaustive approach to determine a schedule guarantees the global minimum for the net energy consumed. In this approach, all the applications are considered in all possible alignments over the entire time frame of consideration (typically the least common multiple of the duty cycle times of all the applications). Each of the applications is placed in all possible combinations with the remaining ones and the set of schedules (that display the start times of each application) are gathered. Out of this set, a schedule is chosen that also satisfies the constraints of the deadline values. The schedule for this entire frame can be repeated from the beginning, provided the number of applications remains fixed for extended periods of time.

3.2.2 The Hierarchical Approach

After the registration stage, a scheduler matrix (or possibly a list of scheduler matrices if the global minimum occurs at different sets of search points) is created out of the existing scheme (involving an exhaustion approach e.g. for a 2-application case). Each application thereafter that is already registered is introduced onto each of these lists and the resultant energy is examined to determine if a global minimum is achieved again. To be specific,

such variational approaches best serve to operate when new applications are accommodated wherever the computationally intensive resources are already in execution, performing the earlier applications.

3.3 The Exhaustive Approach for Scheduling

3.3.1 Registration for Multiple Applications

The present scheduling algorithm schedules a set of sensor applications specified in a configuration file sent by the application developer to the scheduler. The algorithm is classified into two stages.

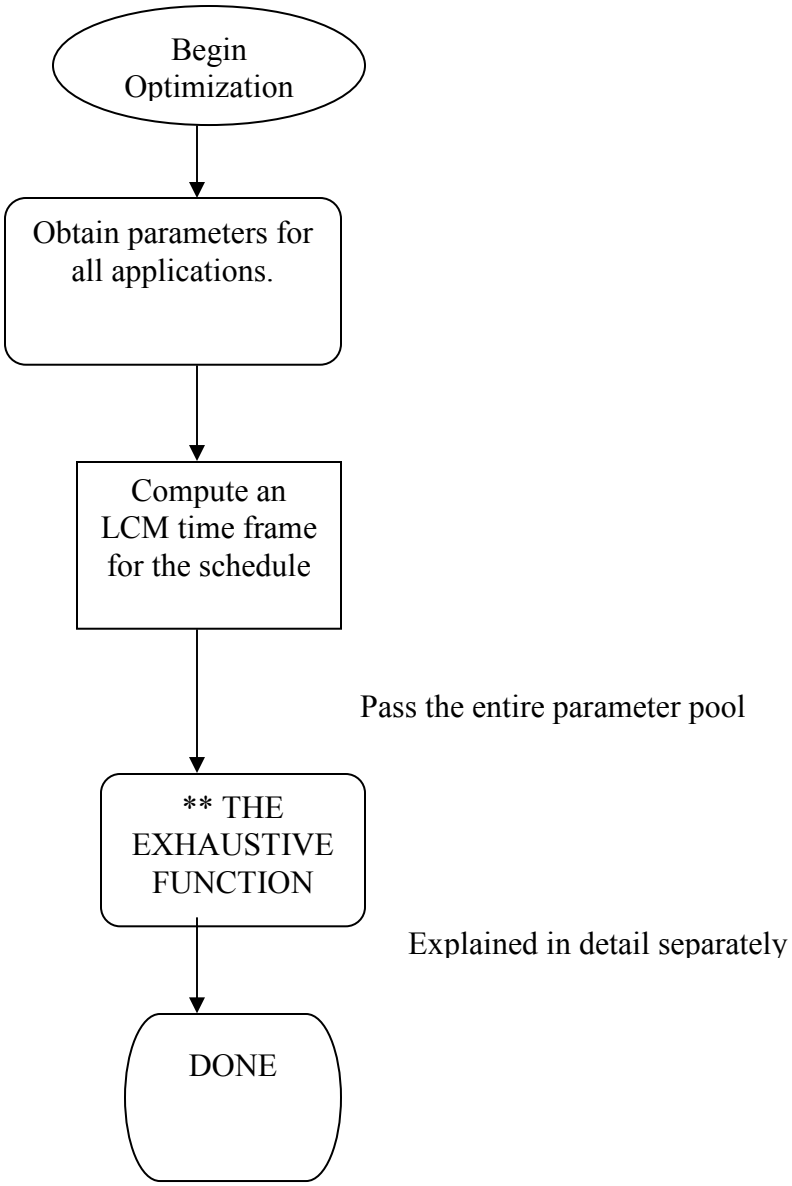
A: The registration stage

B: The optimization stage

The registration stage as discussed in the previous chapter, registers a dynamic set of sensor applications that need to be executed with specified duty cycles. The descriptor files contain all the required parameters to implement an efficient schedule that observes a global minimum point in energy. For demonstration purposes, a scenario can be considered for an offline scheduling wherein the registration stage of the algorithm accommodates a variable set of applications. A configuration file is sent by the application developer that contains the number of applications to be scheduled along with the filenames of each of the descriptor files. Following this, the descriptor files for each application are read and all the resources are registered.

Below is shown a flowchart of the entire optimization structure, with all the important functions highlighted. The iteration approach to the exhaustive process is depicted in a separate flowchart. The scheduler matrix that results from the exhaustive search represents the optimal scheduling scheme that can be re-used until a new set of applications are provided by the developer for an efficient scheduling or if a new application is added to the existing application pool.

Fig. 3.1 – Optimization stage for multiple applications



The present approach is verified for a smaller scale of data (to verify the correctness of the algorithm) in both the cases of

A: Larger periods for applications

B: Application number >= 4

are being considered for a timing analysis. In the next chapter, various test vectors are generated to examine the features of this approach. The test cases vary from simple ones in which the duty cycles are multiples of each other to more general cases where the LCM is the product of all the periods. The scheduler performance is analyzed for these different cases to determine the energy savings attained by operating at the minimum point as opposed to an unscheduled event where the probability of working on a worst-case energy point is very high. In addition, the peak to average power analysis is made for the entire exhaustive search in which a fixed battery supply is assumed and the power consumed by all the resources in each mode is provided by application developer. The histogram provided in the next chapter supports the importance of scheduling in order to obtain the minimum operating point.

3.3.2 Description of the Exhaustive Function

The exhaustive approach is brute-force implementation of the search involved in the determination of the global minimum. The idea is to construct a cardinality set for each application j, which basically corresponds to all the possible combinations of the start times of each instance of each application. All these sets are then considered and each combination across such sets is sent into the energy minimization function. The absolute

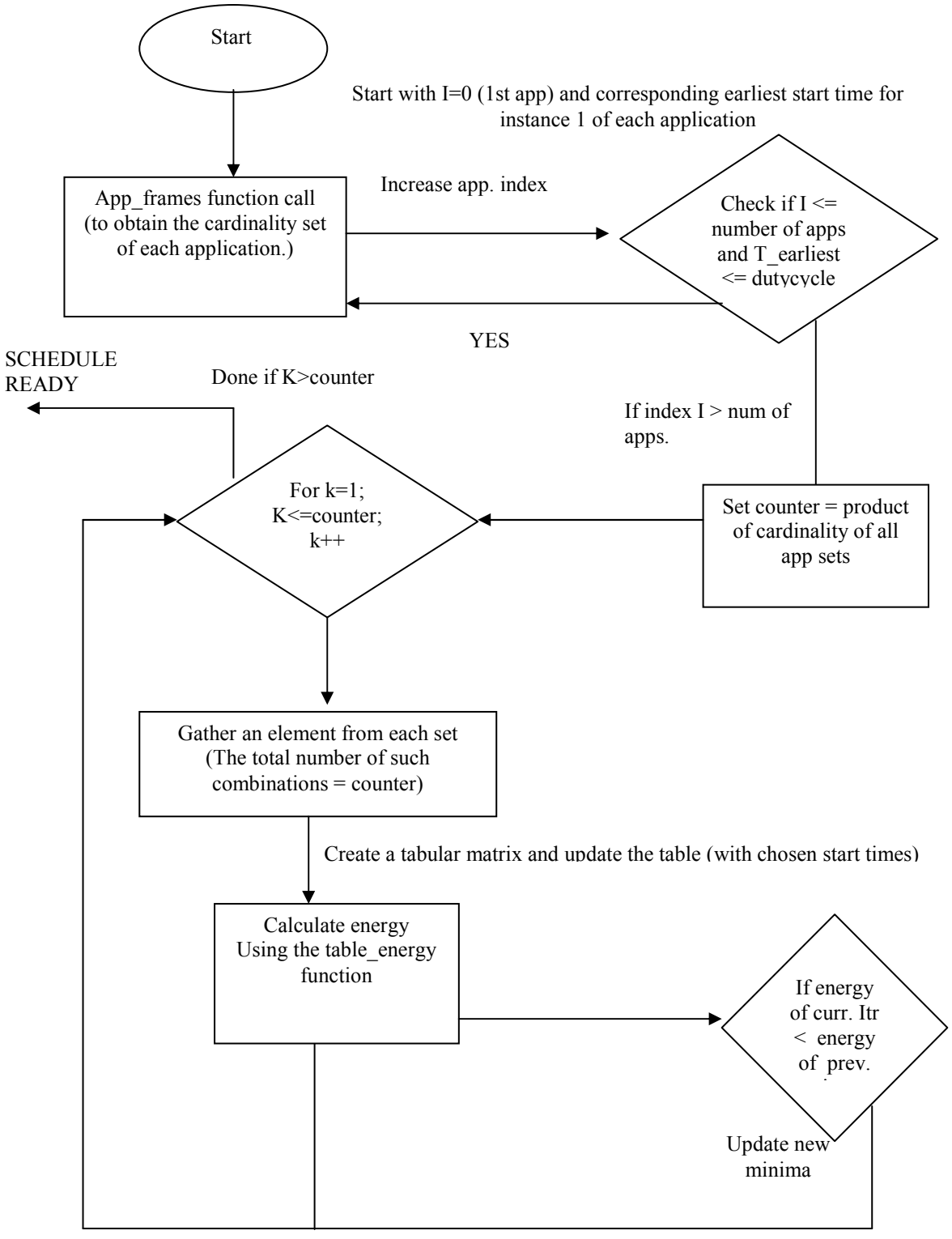


Fig. 3.2 – Flowchart of the exhaustive loop

energy minima point is determined by evaluating for all iterations, the corresponding table matrix to output an energy value for the entire running time. The exhaustive function is divided into smaller subroutine calls (in the sequence shown in the C-code):

1. **AssignTimes**: This sub-routine is called in the function ExhaustionApps function. In this function, each combination of possible start times from the cardinality set of each application is sent in to update the created tabular matrix. The update corresponds to the corresponding hardware bits being set for the entire runtime of each application.
2. **CreateTable**: This subroutine allocates dynamic memory and initializes the tabular matrix.
3. **IncrementApps**: This function makes sure all the possible combinations are considered across each cardinality set. Since the number of such sets is a variable, the index of the element position (corresponding to a specific cardinality set) is incremented to indicate the completion of a single iteration.
4. **IncrementFrames**: This function is similar to the **IncrementApps** function but differs in the way that it runs across all the possible instances of each application and exhaustively gathers for every iteration, all the possible start times of all instances for a particular application. Each time this function is called, a cardinality set for each application is updated.
5. **Exhaust_frames**: The Exhaust_frames function calls the IncrementFrames function repeatedly to create the cardinality set for each application. Once such a set is created for each application, the ExhaustionApps function can run across

every cardinality set, pick an element from each set and pass into the energy determination function.

6. **ExhaustApps**: This function represents the main exhaustive search loop, which incorporates the entire possible combinations of start times of each instance of each application.
7. **Table_energy**: This function takes in the table created for every iteration and determines the total energy associated with it.

3.4 Details of the Energy Measurement Function

3.4.1 The Table Description

The function table_energy takes in as its parameters a specific schedule table created for every iteration and measures the total energy consumed for the entire length of time allocated by the LCM value of all the duty cycles. The main features of this function are described with an emphasis on how each source of energy consumption comes into the energy calculations. To begin with, a brief re-explanation of the structure of the scheduler table is necessary. The rectangular matrix consists of 17 columns. The first column corresponds to the time index followed by 8 hardware bits and 8 software application bits. The applications are identified in an ascending order from bits 9 to 17 i.e. Application 1 corresponds to bit 9 while application 8 corresponds to bit 17. A unity in any time position of the 1^{st}, 3^{rd} and 4^{th} bits indicate that the hardware resources PFU,

AXIS pre-processor and the camera are in the ON state. Similarly, a unity in any time position across bits 9 through 17 indicate that the corresponding application is in the execution state till the runtime of the application.

3.4.2 The Energy Management Scheme

The hardware resources PFU, AXIS and the camera can be in three different states- the boot, executing/idle and the shutdown states while the power-management preprocessor, LP3500 and the IR motion sensor exist in a high-power execution mode and a low-power sleep mode. The power.txt file contains the power consumed by each resource in its respective mode of operation. The total energy measured is a sum of the component energies of each resource in all the possible states it exists as a function of time. For each iteration involving a specific set of start times for the application set, the total energy thus calculated, is sent back to the optimization stage of the algorithm. The optimization stage replaces the existing energy minimum whenever a new minimum occurs. At the end of the exhaustive search, the matrix that achieves the global minimum is chosen to be the optimal schedule table. The schedule is then sent by the interface to the preprocessor LP3500 as a series of commands understood by LP3500, which would enable/disable the various operating states of the hardware resources.

3.4.3 The Components of the Energy Function

The main functionality of the energy determination function is explained in the present section. For each potential schedule vector that is sent as an input to the energy calculation function, it is essential to check if it is a feasible schedule or not. Each schedule that is constructed is clearly a function of the start times of the various instances of each application. In a specific configuration it might occur that a specific set of instances of applications may be aligned in such a way that the resources are transitioned from an executing state to a shutdown state and then back to an executing state in a net time that does not take into consideration the total shutdown time of the resource. Hence, every time a resource transitions from an ON state to a shutdown state i.e. transitioning from a 1 to a 0, it is checked that the next time the resource is switched back to a 1 state, the time index difference between both the events is at least the shutdown time of the resource. If this condition fails, then the resource under consideration is set to remain in the 1 state without shutting off. In other words, it is forced to remain in the idle state. Another modification that the function brings about in the schedule vector before the energy measurements is to observe the BETA values, which denote interference values between pairs of applications. Based on the values provided, realignment of application start-times can occur. Specifically, applications that are scheduled to occur at the same start time and have temporal dependencies or resource conflicts between them are executed sequentially depending on the priority in the dependency chain. In the present algorithm, cases of all BETA values=0 and all values=1 are considered.

Each of the resources, PFU, AXIS and the IMAGER are then perceived as hardware bits in the scheduler table and the net energies are analyzed in various states of each of these resources. For resources that can be present in the idle state and have a defined power associated in that state, a break-even point in the state is determined every time the resource transitions from an ON state to a shutdown state. A comparison is made between the net transition energy (the shutdown energy after the present execution frame + the boot energy for the next execution frame) and the net idle energy for the entire frame. In case the net idle energy of the resource is lesser than the transition energy, the resource is allowed to settle in an IDLE state for the entire duration. For the means of these computations, it is important to observe the states in which the remaining components exist. Specifically, the LP3500 and the IR sensor can only exist in the high-power state when the other hardware resources are in the ON state. They can be put in the sleep state, which is the low-power state, only when the other resources are shutdown. The scheduler table is then evaluated sequentially for each resource to get the energy in various modes of operations of all the hardware resources of the platform. The calculated energy for each potential schedule is then analyzed for the absolute energy minimum in order to qualify as the optimal schedule.

An overview of the entire exhaustive search structure is depicted in the form of a flowchart above. The total number of iterations depends on the coarseness of the exhaustive search. Each of the applications are defined by their own cardinality set, which contains the pre-computed results of all the possible start times of all the instance

of each application in the allocated maximum cycle of observation. The detail of the effects of certain scaling factors that alter the coarseness of the search and thus, the optimal value of the energy value is explained in the next section.

3.5 Factors Affecting the Exhaustive Search

3.5.1 Earliest Instance Scaling (linear translation of the first instance)

There are two scaling factors that affect the granularity (coarseness) of the exhaustive search. In the scheduler algorithm, the earliest instance scaling factor, represented by EAR_SF indicates the jumps in which the first instance of each application occurs. It is seen that the first instance can start anywhere from time t=0 to time t=d where d represents the period of that specific application. All the remaining instances of each application depend on the start time of the first instance. However, the algorithm considers that every successive pair of instances of each application is separated by its period embedded within a little tolerance frame.

3.5.2 Tolerance Scaling (transitions in the tolerance frame of an instance)

The other scaling factor that affects the performance of the exhaustive search and in a more dominant way is the TOL_SF or the tolerance-scaling factor, which determines the jumps in which the start time of each instance of a specific application can make in its

tolerance frame. This applies to all the instances except the first one as the start time of the first instance is determined by the linear translation, described by the term EAR_ SF above. To be specific, the tolerance value varies from –T to T around each potential start time where T represents the tolerance value specified in the descriptor file of each application. As explained earlier, the tolerance values of each application is set to be at 10% of the duty cycle value. The simulations support the intuition that the number of jumps of the start time of a particular instance in each tolerance frame is the dominating factor in the total number of exhaustive searches.

These scaling factors can be varied according to the duty cycle values of each application. From the discussion above, it is clear that both these factors simultaneously affect the length of the execution time of the exhaustive search and also the least possible minimum energy value of the platform. In order to have a reasonable running time, both the factors, EAR_SF and TOL_SF are to be placed in order of the magnitude of the tolerance values. A detailed set of simulations were performed to examine the effects of varying both the scaling factors simultaneously and to observe the effect on the minimum energy determined by the scheduler. The main idea was to observe the minimum values and determine whether they would converge i.e. the energy gap between successive pairs of energy outputs would reduce while increasing the granularity of the search space. Specifically, in order to observe the effect on the energy minimum over a more granular search space, the scale of the search is decreased through the two scaling factors. Once the global minimum is determined for a given set of parameters i.e. TOL_SF and

EAR_SF, these are scaled down proportionally to perform a more detailed search and get a better estimate of the energy minimum. This enables examination of the effects of these parameters over a much wider order of magnitudes of both the parameters, at the expense of some uncertainty as to whether the true minimum was found.

Chapter 4

Results from scheduler simulations

4.1 Test Vector Analysis

A set of test vector cases was analyzed in detail for a 4-application scenario. The simulations were performed to check the sanity of the scheduling algorithm itself by comparing it to a pre-computed scheduler and hence its energies. The hand computations were done for an ideally chosen set of duty cycle values and these values were compared to the simulated results from the scheduling algorithm.

The simplest test vector was one involving a set of duty cycle values, which shared a greatest common divisor belonging to the specified set of duty cycles. Hence to begin with, hand calculations were performed with values for 4 applications and the BETA values considered were the "all zero" case and the "all one" case. Period values of 1000,2000,2000 and 4000 and execution times of 1,2,3 and 4 units respectively were used. For this simple specification set, it could be seen by observation that the best-case energy occurs at a point where application start times overlap as shown in the figures above, for both cases. Though this could be one among the set of optimal schedules, it is the most visible solution. It is to be noted that the execution time of each application is small in magnitude compared to the duty cycles, tolerances of that application and the boot times of each resource. So the number of resource boots saved dominates the energy

savings. For this test case and in general, the resource and application parameters used are tabulated below.

Application	Resources	Execution time (seconds)	Application Period (seconds)	Tolerance value (seconds)
App 1	• LP3500 • PFU • IR	1	1000	100
App 2	• LP3500 • PFU • Imager • Axis	2	2000	200
App 3	• LP3500 • PFU • IR	3	2000	200
App 4	• LP3500 • PFU • Imager • Axis	4	4000	400

Table 4.1 – Application parameters for the test vectors

Resource Name	Power consumed	Boot time in Seconds
LP3500	0.296watts	Combined value of 15 seconds for each application
PFU(Boot)	2.5 watts	
PFU(Exec)	11.4 watts	
PFU(Idle)	5.8 watts	
Imager(Boot)	3.18watts	
Imager(Exec)	0.8watts	
Axis(Boot)	2.808watts	
Axis(Exec)	0.500watts	
IR	0.108	

Table 4.2 – hardware resource parameters for the test vectors

These assumptions hold for all the energy calculations described below.

1. The combined boot time of all the resources is fixed at 15 seconds.

   So energy calculations are: ***E* = (Sum of Powers of all the resources) * 15.**

2. The shutdown energy for any resource is 0 and the total idle power is 10 m watts.

The figures below give a pictorial description of the optimal schedule for the parameters mentioned above. For the Betas=0 case, all applications can start concurrently if scheduled together, while in the Betas=1 case, they have to be executed sequentially. These two represent the bounds for the best and worse-cases of the energy representation.

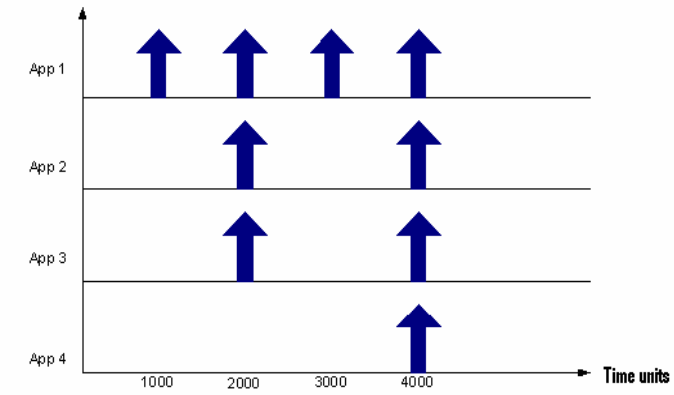


Fig 4.1.a- scheduling for the Betas=0 case

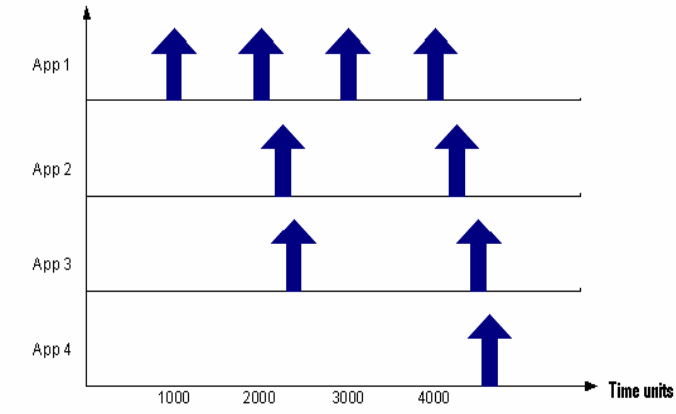


Fig 4.1.b- scheduling for the Betas=1 case

The scheduler simulations were performed in two separate scenarios and compared with the hand calculations. The first case was when the resources were shutdown completely when not in use and rebooted when needed. Here the shutdown energy was assumed to be 0 for simplicity of calculations. The energy savings are mainly due to the transition costs. The second case is when the resources can be kept in a low-power idle mode wherein the total idle power is negligible. The main goal of this test vector analysis was to make sure that the algorithm could adapt itself to different specifications provided by the developer and also is robust to different hardware platforms. In the last chapter of the thesis, we propose other platforms on which the algorithm can be implemented on to analyze the energy savings on the platform.

The table shown below compares the hand calculations to the scheduler results in the two cases discussed above. All minimum energy calculations lie between the Betas=0 and the Betas=1 cases and the simulations were performed under reasonable granularity.

BETA VALUES	***Hand Calculations (Resources are shutdown when not in use and no idle mode)***	***Scheduler Simulation***	***Hand Calculations (With total idle time power = 10 m watts)***	***Scheduler simulation***
$\beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$	***2018.4 J***	***MIN:2046.6 J***	***466.14 J***	***MIN: 476.66 J***
$\beta_1 = \beta_2 = \beta_3 = \beta_4 = 1$	***2126.3 J***	***MIN:2200.6 J***	***615.29 J***	***MIN: 629.6 J***

Table 4.3 – Energy comparison between simulations and hand calculations.

4.2 Histogram Analysis of the Scheduler Algorithm

In order to achieve a qualitative analysis on resource scheduling for energy harnessing, we take a statistical approach towards a scenario where the platform runs without the support of a scheduler. The histogram shown below in fig.4.3 plots the energy ranges achieved on the platform for a specified set of specifications on the x-axis and the frequency of the occurrence of these energy ranges during the entire exhaustive search on the Y-axis. From this figure, we can argue that in case of assigning the platform with an arbitrary schedule, the probability of falling into the category of an average case energy scenario is very high. It can be seen from the figure that the largest frequency of samples occurs towards the average of the best and worse case energies, which differ by a factor of 2.

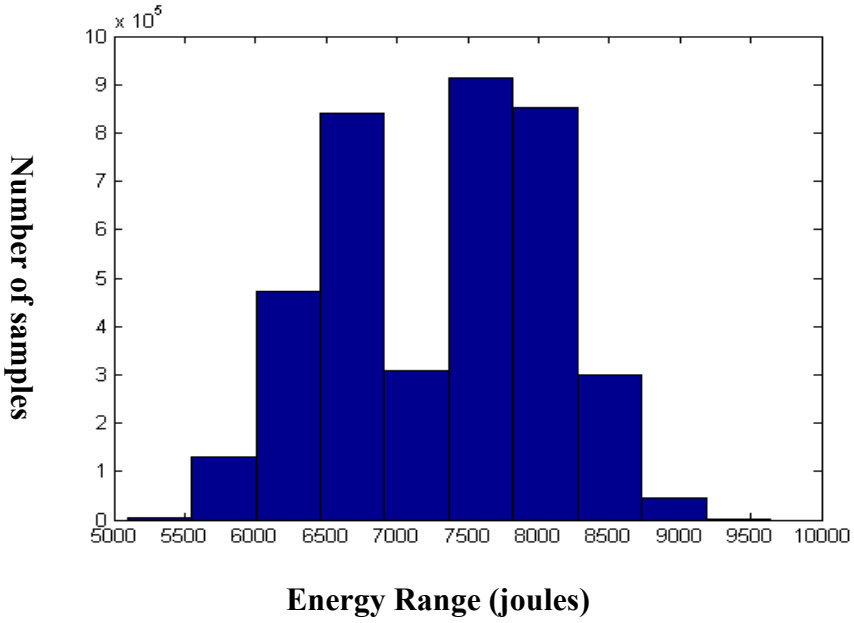


Fig. 4.2- Histogram of range of energy values

Since the median of the entire set of samples is more than the average energy, we can see that there is a much larger probability associated with arriving at an operating point greater than the average energy and moreover, weighing more towards the worse case energy. Thus, scheduling can be of considerable benefit. If a complete exhaustive approach is employed with the finest amount of granularity, it is guaranteed to achieve the optimal point, which has the least probability of occurrence. Even with a constraint on the execution time on the exhaustive algorithm, we can aim to achieving operating points towards the optimal point, by allowing the scheduler to search with a higher level of granularity.

4.3 The Working of the Scheduler with the Interface

The interface, which we talked about briefly in earlier chapters, communicates between the schedule matrix and the power-management preprocessor, LP3500. The optimal schedule matrix is transformed into a series of “sch” and “exe” commands as understood by the preprocessor. The schedule that resulted out of the test vector with the assumptions stated earlier was used to measure the real-time power measurements obtained from the platform. In fig. 4.3 shown below, two plots are shown; the one termed “matlab simulation” is a power plot of the optimal schedule through the simulation. The plot overlapping it is the real power plot from the platform. This plot is smoothed by removing the randomness in the system current around the states where the platform is undergoing some activity. It can be seen that the real power plot has system current in the

shutdown mode which is the low power mode current of the LP3500 and the IR sensor. In addition, just to separate the two plots, the shutdown power was assumed to be 0 in the simulation, which is shown as a difference in the real plot during the shutdown state of the platform.

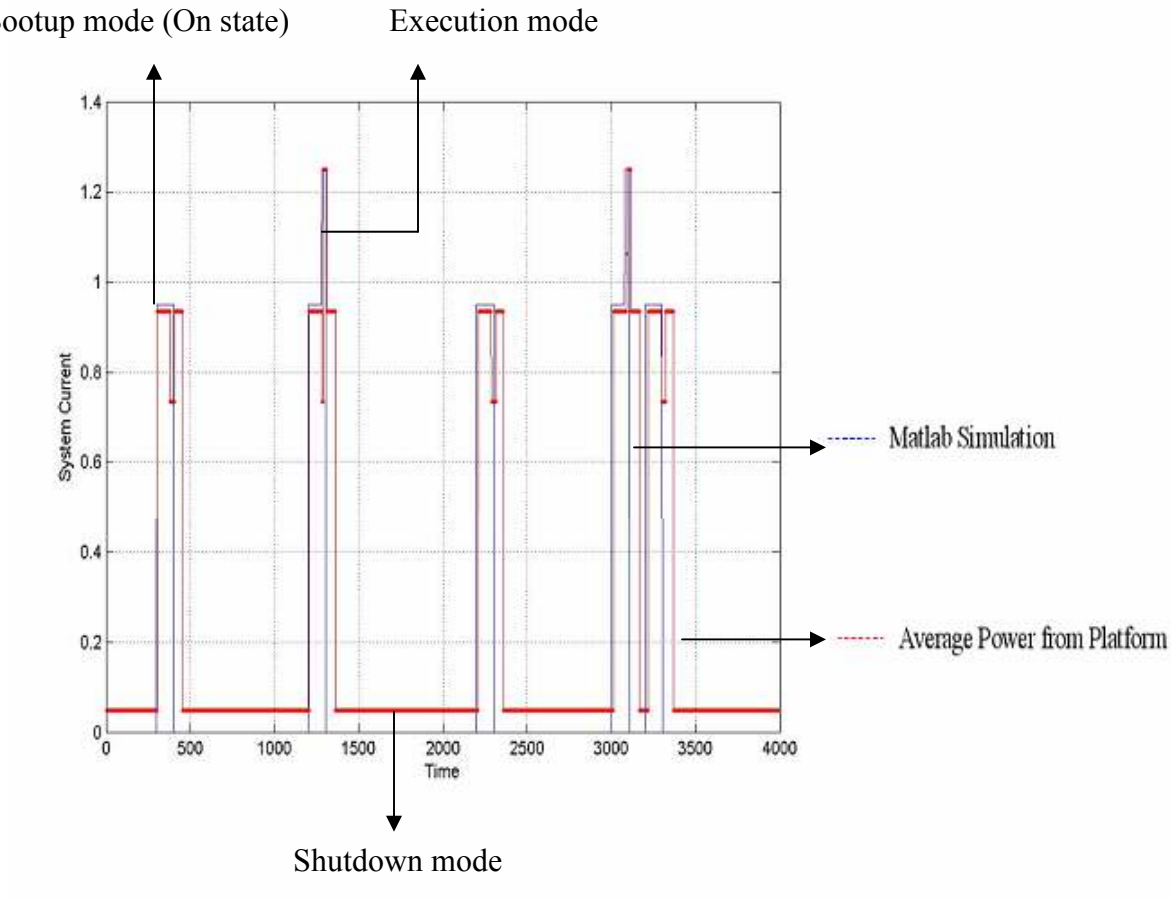


Fig. 4.3 - Power Comparison of Platform vs. Simulation

It can be seen that over a long period of time frame (the LCM period of all the periods for example), the average power of the platform can best be treated as the average of the low power mode and the power during the active cycle. Thus, we analyzed periodic

applications with low duty cycles in this scenario and observed that the average energy on the platform can be brought down to a much lower value when compared to the energy consumed when all the resources are always in the active state. With low duty cycled applications, it is highly advantageous to drive the platform to the lowest energy operating point by making sure that the platform remains in the shutoff mode whenever applications are not scheduled or no events take place. In addition, the applications that can be performed sequentially are executed in such a way so that the transition costs of the common resources are saved.

It is seen that most energy savings occur when the set of periodic applications are scheduled to execute in a cluster as opposed to discrete executions in a particular time frame. This would mostly happen in cases where some periods in the set are multiples of others. We have simulated various test cases for a multi-application scenario that satisfy this simplified condition and compared the results with manual calculations that achieve the optimal schedule. The optimal energy point is a lower bound for the simulations and the minimum energy through the simulations should converge to this bound as the granularity of the search space gets finer. The ratio between the worst-case energy and the minimum energy through each simulation quantifies the amount of energy saved. Based on the application parameters such as the periods and the tolerances associated with the duty cycles, the savings ratio can be of magnitude 2, 3 or even higher. For instance, we have performed simulations for 4-applictions to determine a maximum of a

three-fold improvement. The next section deals with two different sets of simulations, which show the improvement in energy consumption.

4.4 Analysis of the Results for a 4-application Case

In the present description, we considered two different scenarios for a set of four applications. The duty cycles, the tolerances and the execution times for these sets of simulations were chosen so that the energy optimization depended only on the duty cycle and the tolerance values. The execution times of the application set were small compared to the duty cycle, tolerances and the boot times of the hardware resources. This setup was considered to analyze the energy savings in transition costs. Also, the exhaustive search on both the sets of applications gave improvements of values close to 2 and 3 respectively. We considered examples wherein the duty cycles were such that a granularity of order equal to the tolerance values of the applications provided the required near optimal solution. The examples, however, differ in the manner in which we represent the granularity of the search space. Also importantly, in these simulations the values of Betas were all assumed to be zero, permitting concurrent execution of applications. It should be realized that the entire exhaustive process is a non-linear calculation in general. Hence, no particular scaling factor pair is "good enough" for all specifications, unless the granularity is made very fine compared to the application parameters. However, both the examples verify the intuition that the minimum energy through the exhaustive search converges asymptotically to the theoretical energy

minimum. Convergence to the optimal point can be observed through the increase in the granularity and the closer the solution is to the optimum, the more monotonically convex is the behavior of the energy curve. In the next couple of sections, we detail out the applications parameters use for the 4-application case and discuss the influencing factors.

4.4.a Simulation That Demonstrates a Two-fold Improvement

In the first set of simulations, the parameters used were as follows:

Duty cycles: 1000,1500,2000,3000

Tolerances: 10% of duty cycles i.e. 100,150,200 and 300.

Execution times: 1,2,3 and 4 seconds

The boot times, duty cycles and the tolerances were much larger than the execution times to demonstrate the energy saving obtained in transition costs. In addition, the idle energy of the resources was assumed to be negligible compared to the transition energies.

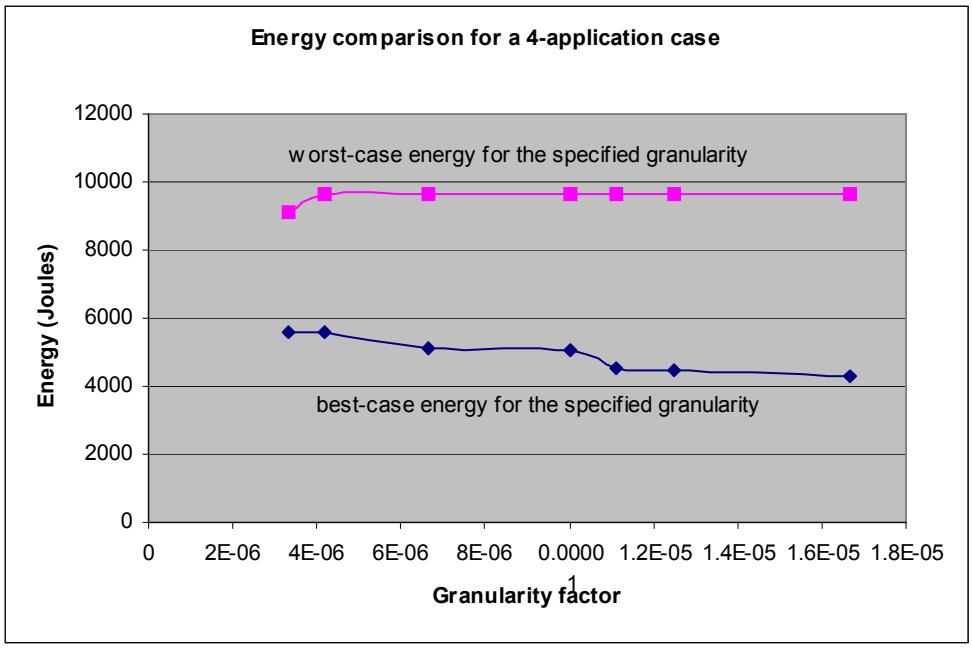


Fig. 4.4 – Energy Savings plot for the 1st simulation

EAR_sf	T_sf	1/EAR_sf	1/T_sf	Granularity Factor	Min.ener	Max.ener
600	500	0.001667	0.002	3.33333E-06	5593	9098.4
600	400	0.001667	0.0025	4.16667E-06	5566	9638.6
500	300	0.002	0.003333	6.66667E-06	5138	9638.6
500	200	0.002	0.005	0.00001	5052	9638.6
300	300	0.003333	0.003333	1.11111E-05	4505	9638.6
400	200	0.0025	0.005	0.0000125	4480	9638.6
300	200	0.003333	0.005	1.66667E-05	4320	9638.6

Table 4.1- Granularity scaling factors and energy values for the 1st simulation

There are two subplots shown in fig. 4-4 above. The subplot on the bottom half shows the comparison of the minimum energy on the platform as a function of the granularity factor. In this plot, the granularity factor is on the x-axis and the minimum energy for each granularity factor is shown on the y-axis. The granularity factor is defined as the inverse of the product of the two scaling factors, discussed in the previous chapter. The scaling factors are increased in granularity gradually for each set of simulations such that the granularity factor increases on the x-axis and correspondingly achieves a decrease in the minimum energy. Instead of providing subplots while fixing one factor and varying the other, we consider this product variable for convenience, which results only in a single curve. It yields a monotonically convex behavior in convergence.

The subplot on the top half of the plot is the worst-case energy as a function of the same granularity factor. It is seen that in the initial coarser searches, the worse case maximum energy is not achieved but on a lower granularity it is achieved and remains fixed at the

same maximum value at every search of a lower granularity, which agrees with the intuition of a theoretical upper bound for the worse-case energy.

In short, the comparison between the two subplots shows two things:

1. The convergence of the minimum energy curve to a constant value as the granularity factor increases.
2. The energy ratio between the worst-case and the optimal energy from the plot is equal to 2.3 (with the applied granularity). It asymptotically approaches the value of 2.5 (15/6), which is the energy savings obtained though a manual analysis. The energy savings is dominated by saving the transition costs of the resources.

4.4 b Simulation That Demonstrates a Three-fold Improvement

In the second set of simulations, the following parameters were used:

Duty cycles: 1000,1000,2000,3000

Tolerances: 10% of duty cycles i.e. 100,100,200 and 300.

Execution times: 1,2,3 and 4 seconds

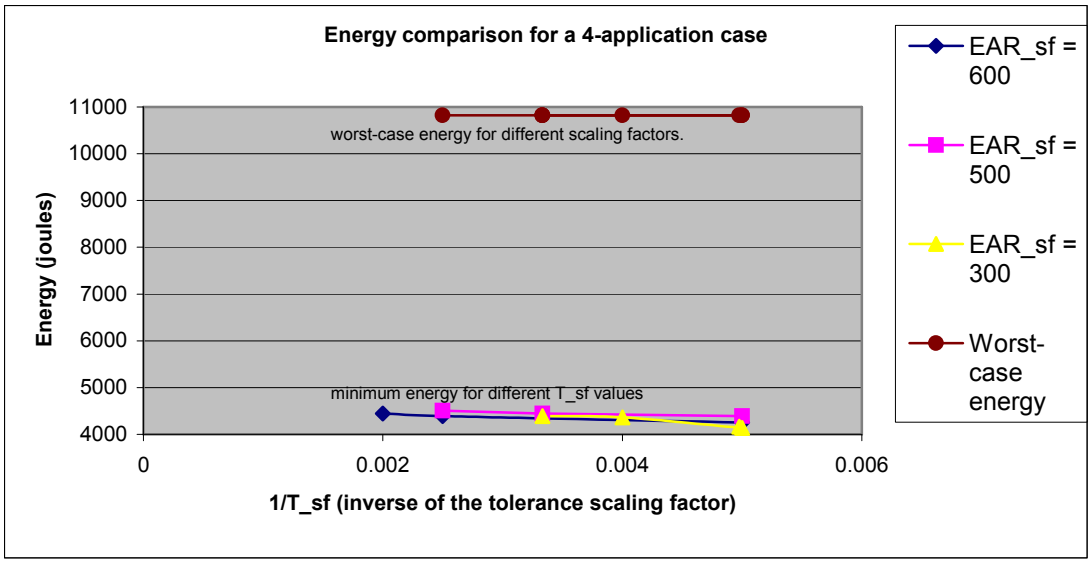


Fig. 4.5 – Energy Savings plot for the 2nd simulation

EAR_sf	T_sf	1/EAR_sf	1/T_sf	Granularity Factor	Min. ener	Max.ener
600	500	0.001667	0.002	3.33E-06	4442.74	9162.8
600	400	0.001667	0.0025	4.17E-06	4390	9742.76
600	200	0.001667	0.005	8.33E-06	4250	10824.3
500	400	0.002	0.0025	0.000005	4506	10824.3
500	300	0.002	0.003333	6.67E-06	4443	10824.3
500	200	0.002	0.005	0.00001	4390.07	10824.3
400	300	0.0025	0.003333	8.33E-06	4315	10824.3
300	300	0.003333	0.003333	1.11E-05	4391	10824.3
300	250	0.003333	0.004	1.33E-05	4362	10824.3
300	200	0.003333	0.005	1.67E-05	4139.46	10824.3
300	201	0.003333	0.004975	1.66E-05	4164.44	10824.3
200	300	0.005	0.003333	1.67E-05	4315	10824.3
100	300	0.01	0.003333	3.33E-05	4264	10824.3

Table 4.2 - Granularity scaling factors and energy values for the 2nd simulation

In the bottom part of the plot in fig. 4-5 above, the minimum energy subplot consists of three curves, which are dependent on the EAR_sf or the earliest time scaling factor. For each value of EAR_sf, the T_sf or the tolerance-scaling factor is decreased, so that the tolerance granularity is increased. In this plot, three separate curves are shown (as opposed to the combined curve in the first plot) for two reasons.

1. The granularity factor (i.e. the inverse of the product of the two scaling factors) did not always behave monotonically with the energy, as in the earlier case.
2. More importantly, between the two scaling factors, we observe the dominance of the tolerance-scaling factor to yield the asymptotic convergence of the minimum

energy. Clearly, for a fixed EAR_sf, increasing the granularity of T_sf pushes the energy value towards the optimal point.

The subplot on the top half of the plot has the worse case energy through the entire set of simulations. In short, the comparison between the two subplots is summarized as follows.

1. The energy savings ratio must be asymptotically bounded by the value of 17/6 or 2.83, which is the theoretical calculation of the energy savings achieved for the specified parameters. In our set of simulations, the ratio of the maximum energy value (10824 J) to the minimum value (4139 J) stands at 2.6.
2. The worst-case energy value remains close to 11000 J and the best-case energy converges to 4000 J respectively, which provide an energy saving ratio of 2.75.

In the above two cases, we have discussed the possibility of a two-fold and a three-fold improvement in the energy savings. In fact, the energy savings can be even higher and can be verified by similar simulations as the number of applications to be scheduled increase.

4.5 Discussion of Further Possible Test Capabilities

4.5.1 Scheduling for a General Set of Application Duty Cycles

The above simulations were chosen as two simple cases to demonstrate the energy savings obtained for a 4-application scenario. We had simulations performed for sets of 2 and 3 application scenarios of periodic applications to compare the energy savings obtained for varying number of applications. Intuitively it makes sense that the savings increase directly with the number of applications provided the applications have periods multiple of each other, or at least having a common factor amongst them. The result of such simplifying conditions is the conspicuous visibility of the energy savings ratio increase. However, all the results of the algorithm presented were so far, confined to these simple cases. It is also important to analyze all the scheduler features for an application suite with periods that are relatively prime to one another. The Least Common Multiple, which is considered to maintain a schedule that is periodic, cannot be used as the schedule length if the product of the periods is very large. We have considered alternatives for the Least Common Multiple, such as a large multiple of the highest period is substituted for the schedule length. There are certain drawbacks with this approach.

1. The number of application instances in that schedule length has to be rounded off for some applications. This would definitely perturb the energy consumption and the energy savings.

2. The schedule created cannot be repeated periodically because the schedule length is an approximation to the required length of the Least Common Multiple. This approximation can offset the minimum energy point through the simulation from the optimal energy point by a large value.
3. In certain cases, if the period values are closer in value to one another, the schedule should be chosen to be a large multiple of the larger period value just to accommodate more instances of each application in that time frame. This would enable a better result in the energy savings.

4.5.2 Scheduling for Applications with a Definite Dependency

In our scheduling approach, we have considered only parallel or sequential execution of applications. In other words, the optimal energy point generated through the exhaustive search for the Betas=0 case and the Betas=1 case are in fact, energy bounds for the all cases, even those which include task dependency. We have not implemented the feature of task dependency between subsets of applications i.e. some of the pairs can execute concurrently while others have to be executed sequentially. This is considered a separate area of research and is considered in the future work. Furthermore, dependency can be classified also into the case of resource dependency wherein application can effectively share the processing time of each processor in execution. The scheduler can incorporate this dependency by considering Beta values between 0 and 1 and indicating the subset of applications that require this dependency. Such cases of application suites are more realistic to occur, as the set of resources need to be shared in wireless sensor applications.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have seen an increasing need for energy consumption in wireless sensor applications. The present suite of applications have evolved from an earlier class of acoustic and seismic applications restricted to target tracking and sensing to a complex set involving distributed monitoring, surveillance and networking applications. There is a great need for secure communications and reliable date transfer in such applications. For the earlier simpler class of applications, dedicated integrated-circuit logic could be effectively utilized to obtain good performance with incorporation of efficient power-optimization schemes. As discussed earlier, the evolution to a more complex nature have widened the demands for energy. Traditional hardware architectures with dedicated logic processing units cannot certainly satisfy the present requirements. We proposed a newer hierarchical architecture with a processor-preprocessor division and the software consisting of a complex operating system. The hardware platform runs Linux, which supports all the secure lower-level communication protocols and the higher-level complex applications. However, the need for higher energy is conspicuous with the requirement of supporting such complex networking protocols and image processing algorithms. The platform features were given an overview in the first chapter with the concept of resource scheduling for a set of energy-aware wireless sensor applications.

The main area of discussion of this thesis is to describe the necessity of scheduling for such computationally and resource intensive wireless sensor applications. In the first part of the thesis, we defined the various parameters required for the scheduler in the context of the application suite. Most importantly, we explained in detail the fundamental differences in the present approach and the previous algorithms described in literature. We then proposed a solution to determine the optimal schedule that operates the platform at the minimum energy point and outlined some variations.

The remaining part of the thesis is divided into two separate discussions. The first discussion deals with a simple heuristic approach that gives an intuition on a 2-application scenario. The second discussion, which forms the crux of the thesis deals with scheduling of multiple sensor applications, each of them with their specifications. The minimum energy point in the scheduler algorithm can be discovered through an exhaustive search, which should converge to the theoretical optimum value as the granularity of the search space is increased. We discussed the variation in granularity through two scaling factors; the lowering of whose values has an increase in the execution time of the search algorithm. However, as we discussed in the earlier chapters, the energy cost function is a non-linear function in its variables, so it is not possible to generalize the effect of increasing granularity to a continuous decrease in the minimum energy determined. Two simulations were shown that support the notion of an asymptotic convergence to the optimal value as the resolution increases and it was also discussed that it is not possible to model the behavior of the energy curve (as a function of the granularity) to be entirely convex in nature. The last part of the thesis included a

discussion on some non-typical cases of application parameters and the performance improvement that could be obtained through scheduling.

5.2 Future Work

In the present work, we had discussed scenarios in which the applications scheduled were periodic and moreover, had their periods be multiples of one another. In such situations, the least common multiple would be larger compared to the individual periods by a small order of magnitude. However, in the previous chapter we mentioned that there existed a limitation in the scheduling algorithm to decide the length of the schedule when the periods were relative primes. In an extension to the present algorithm, we would be working to include the scheduling mechanism for such scenarios. A proposition being considered is to construct schedules for windowed subsets of applications based on the exhaustive approach. This would result in discrete "superframes" of application instances with their own schedules. It is evident that no periodic nature exists between the set of superframes. Moreover, we do not consider the product of all the periods as the whole schedule length but partition the time frame into smaller lengths to accommodate multiple instances of each application. The exhaustive approach guarantees an optimal schedule for each of these partitions. In addition, a heuristic approach could be used to adjust for minimal energy operations at the boundaries of each of these superframes. Thus, it can be concluded that the larger scheduling problem could be decomposed efficiently into smaller problems to achieve an

optimal schedule for a large enough period of time. However, it can be seen that, the number of superframes has a direct effect on the energy optimality in that many scheduling computations would be required.

In addition, we highlighted the importance of scheduling to achieve the optimum point. The prototype of the Smart Objects platform we considered was shown in the first chapter of this thesis. The scheduler algorithm had as its input, the resource parameters of the various hardware blocks shown in fig. 1.1. In addition, the power consumed in each time unit for the various modes of these resources and the application parameters determined the optimal schedule. However, we have demonstrated a robust scheduling algorithm that behaves as a many-one mapping of the all the above inputs to produce the energy minimum. In other words, this algorithm works for any application and resource data supplied by the developer. Presently, another sensor platform is being developed with the main processor being the STARGATE processor that has it own advantages in terms of the boot and the shutdown times.

The main aim of this future work would be to execute the same set of applications across these two different platforms and make a comparison of the energy savings that occur between the two platforms. In this way, the scheduling algorithm could be efficiently tested for robustness in addition to its precision to arrive at a near optimal solution. With the advent of such a generalized algorithm, it only becomes easier to deal with the varying range of complexities in wireless sensor applications and at the same time, successfully implement different hardware platforms to execute those applications.

BIBLIOGRAPHY

[1] A. Sinha and A.Chandrakasan. “Energy Efficient Real-Time Scheduling”. Computer Aided Design, ICCAD 2001. 458-463, Nov 2001.

[2] M. Srivastava, A.Chandrakasan and R.Brodersen. “Predictive system Shutdown and other architectural techniques for energy-efficient programmable computation”. IEEE Transactions on Very Large Scale Integrated (VLSI) systems. VOL 4. 42-55. March 1996.

[3] H.El-Rewini, T.G.Lewis and H.Ali. Task Scheduling In Parallel And Distributed Systems, Prentice Hall, 1994.

[4] A. Chandrakasan, A. Burstein and R.W. Brodersen. “A low-power chipset for portable multimedia applications”. ISSCC, Feb 1994 pp 82-83

[5] E.L. Lawler and C.U.Martel, “Scheduling periodically occurring tasks on multiple processors”. Information Processing Letters. 12(1): 9-12 (1981).

[6] J.Hill, R. Szewczyk, A. Woo,S. Hollar, D.Culler and K. Pister. “System Architecture Directions for Networked Sensors”. ASPLOS 2000, Cambridge, November 2000.

[7] G.J. Pottie and W.J.Kaiser, “Wireless Integrated Network Sensors”. Communications of the ACM. Volume 43 Issue 5, 51-58, May 2000.

[8] A. Manzak and C. Chakrabarti. “A Low Power Scheduling Scheme With Resources Operating At Multiple Voltages”. IEEE Transactions on Very Large Scale Integrated (VLSI) systems. VOL 10. No.1, February 2002.

[9] F.Zhao, J. Liu, J. Reich, M. Chu and J. Liu. “ Programming Embedded Networked Sensor Systems”. CODES+ ISSS 2003. October 1-3 2003, Newport Beach, CA, USA.

[10] R.W. Brodersen, A. Chandrakasan and S. Sheng. “Technologies for personal communications”. VLSI circuits symposium , Japan 1991.

[11] T.Perling, T.Burd and R. Brodersen. “The simulation and evaluation of dynamic voltage scaling algorithms”. Int’l Symosium on Low Power Electronics and design, pages pp 76-81, Aug 1998.

[12] M. Weiser, B.Welch, A.Demers and S.Shenker. “Scheduling for reduced cpu energy”. The first symposium on Operating Systems Design and Implementation (OSDI), pages 13-23.

[13] Apple Computer Inc., “Power manager IC, and reduced power modes” in Technical Introduction to the Macintosh Family. Reading, MA: Addison-Wesley, Oct. 1992, ch 20, 2nd ed.

[14] H.S. Stone, “Multiprocessor performance”, *in high-performance computer Architecture.* Reading MA: Addison-Wesley, 1987, ch 6.

[15] C.-L. Su, C.-Y. Tsui, and A.M.Despain, “Low-power architecture design and compilation techniques for high-performance processors,” *Digest of Papers, IEEE COMPCON Spring ’94,* Mar. 1994.

[16] M.Muller, “The ARM6: Power efficiency & low cost,” *Hot Chips Symp*., Aug. 1992, pp. 3.3.1-3.3-11.