DESIGN OF ADAPTIVE AND HYBRID ENERGY & QOS AWARE HETEROGENOUS MULTIPROCESSOR SCHEDULING STRATEGIES FOR EMBEDDED SYSTEMS

SIVANESAN KAILASH PRABHU

(B.Tech., National University of Singapore)

A THESIS SUBMITTED FOR THE DEGREE OF

MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

Acknowledgment

I would like to express my sincere gratitude to the following individuals who, in one way or the other, have helped me see through this dissertation. First and foremost my supervising Professor, Dr. Bharadwaj Veeravalli, for his guidance and valuable inputs throughout my research term. His patience and understanding during certain phases of my research work, especially with regards to the simulations which took an unexpectedly long time to complete, is greatly appreciated. I am grateful to Dr. Yajun HA for agreeing to be my examiner and appraise this dissertation.

I would also like to thank Mr. Goh Lee Kee and Mr. Sivakumar, both from the Institute for Infocomm Research (I^2R) Singapore, for their contributions to the user-interface tool. Specifically, Mr. Goh Lee Kee for his efforts in widget development as well as algorithm integration and Mr. Sivakumar for guidance and feedback during the development phase.

This dissertation would not have materialized without the Embedded & Hybrid Systems (EHS) II initiative of Agency for Science, Technology and Research (A*STAR), Singapore. A special thanks to A*STAR for this.

A warm thanks to the National University of Singapore for subsidizing my school fees. Needless to say, I am thoroughly pleased with the size of my wallet in my pocket now!

Last but not least, I would like to thank the staff of the Electrical and Computer Engineering department who have helped me with various administrative work.

Abstract

A key challenge faced by Body Sensor Networks (BSN) is the efficient utilization of energy at the various processing nodes. Being portable and unobtrusive, the preferred choice of energy source for these nodes is the battery. In this work we propose a static battery and QoS aware scheduling algorithm to schedule an application modeled as a Directed Acyclic Graph (DAG), with dependency and deadline constraints, on to the heterogeneous processing elements that comprise the BSN. This work is the first in literature to address the battery-aware multi-processor scheduling problem, which happens to be a traditional task, under the context of each processing node being powered by individual heterogeneous batteries. The proposed algorithm exploits *rate-capacity* effect and *recovery* effect to maximize the charge drawn from a battery. The algorithm also strives to achieve uniform wear-off of the batteries in the BSN. A novel battery model previously developed in literature is utilized to guide the scheduling process. The DC-DC converter is eliminated and the battery is interfaced directly to the processing element to minimize converter losses and achieve better control over the battery discharge current. The developed algorithm is suitable for systems that do not support Dynamic Voltage Scaling (DVS) and where charge utilization can be maximized by profiling the discharge current drawn from the battery. An application tool is also developed as part of this work to provide a graphical front-end to simulate and evaluate performance of scheduling algorithms.

Contents

List of Tables

List of Figures

List of Abbreviations

ANCC	Average Normalized Charge Consumption
BAPM	Battery-Aware Power Management
BSN	Body Sensor Network
DA	Damage Assessment
DAG	Directed Acyclic Graph
DAM	Damage
DAT	Data Available Time
DP	Data Preprocessing
DVS	Dynamic Voltage Scaling
FT	Finish Time
GUI	Graphical User Interface
LBMD	List-Based Minimum Damage
MSMS	Multiple Source Multiple Sink
MSSS	Multiple Source Single Sink
NCC	Normalized Charge Consumption
PVS	Passive Voltage Scaling
RF	Radio Frequency
RL	Ready List
RO	Rescue Operation
SSMS	Single Source Multiple Sink
SSSS	Single Source Single Sink

ST	Start Time
WCCC	Worst Case Current Consumption
WCET	Worst Case Execution Time

Chapter 1

Introduction

A hallmark of the past decade is the explosive growth of embedded computing, in terms of capability as well as pervasiveness. Advancements in microprocessors, sensors, wireless communications, semiconductor manufacturing, software/hardware design tools etc. have produced embedded devices that offer tremendous computing as well as networking power in form factors never witnessed before. A simple example of such a device would be the mobile phone which has witnessed double-digit growth rate in most parts of the glove [3]. This trend is expected to accelerate and culminate into the concept of Wearable Computing [4][5]. An exciting application of such a realization would be Body Sensor Networks (BSN). BSN consist of a number of tiny sensor nodes, each capable of communication over a wireless medium, typically tethered to the human body for autonomous 24-7 monitoring of physiological data. It is envisioned that BSN will provide a major breakthrough in medical practices since it facilities autonomous and remote administration of healthcare.

One of the key hurdles that BSN have to surmount before they can be adopted for widespread usage is the efficient utilization of energy available at the various sensor nodes. The sensor nodes of the BSN are wearable and perhaps even implantable, and are thus powered by a mobile energy source such as an electro-chemical battery. Being ambulatory in nature, recharging or replacing the node batteries might not be practical. Consequently, energy-aware design techniques to prolong the operational lifetime of BSN have to be employed. Moreover, the battery is a non-linear energy source whose behavior deviates significantly from that of an ideal energy source. Thus, designing BSN battery-aware, as opposed to just energy-aware, is of paramount importance. Energy losses such as those occurring in energy transmission components like DC-DC converters need to be minimized as well if battery utilization is to be maximized.

1.1 Research Scope

In this work we adopt a system-level approach to inject battery awareness into BSN as opposed to gate/circuit/board level optimization techniques. We specifically deal with the problem of scheduling a BSN application, modeled as a Directed Acyclic Graph (DAG), on to the nodes of the BSN in a battery-efficient manner. The predominant energy sinks in a modern sensor node are the microprocessor and the radio-frequency (RF) module [6]. In line with this, our work considers the energy requirements of these two sub-components only and ignores other energy sinks that might exist on a sensor node such as sensor or storage elements. To cater for the wide range of battery-chemistry available in today's market, the batteries powering

the sensor nodes are assumed to be heterogeneous. Due to complexities involved in accurately predicting battery run time behavior, only static (offline) scheduling is considered. We survey various battery models and adopt one suitable for our needs. Developing a battery model from grounds up would be out of scope of this research work.

1.2 Research Contributions

The research contributions of this dissertation are as follows:

1. We propose a static battery and QoS aware scheduling algorithm called *List Based Minimum Damage(LMBD)*to schedule an application DAG, with deadline constraints and non-negligible inter-task communication costs, on to the heterogeneous processing elements that comprise a BSN. The proposed algorithm exploits non-ideal battery characteristics, namely, rate-capacity and recovery effect to maximize the charge drawn from the battery. A network-wide optimization is adopted such that uniform wear-off of the batteries in the system is achieved. A novel battery model previously developed in literature is utilized to guide the scheduling process. The developed algorithm is suitable for systems that do not support the traditional Dynamic Voltage Scaling (DVS) technique and where charge utilization can be maximized by profiling the discharge current drawn from the battery. The proposed algorithm is validated via rigorous simulation runs.

We believe our work to be the first in literature to address the battery-aware multi-processor scheduling problem, which happens to be a traditional task, under the context of each processing node being powered by individual heterogeneous batteries.

2. The effect of the DC-DC converter on battery performance is studied and an energy delivery setup to mitigate converter losses is adopted. A novel physical setup known as Passive Voltage Scaling(PVS)[2] eliminates converter losses totally by interfacing the battery directly to the hardware. This setup is found to be highly suitable for wireless sensor nodes. It is also found to provide greater control over the battery discharge current as compared to conventional setups. We have adopted PVS as the physical setup of the BSN under study and the proposed algorithm operates under this setup.

3. An application called *Graphical Simulator Tool(GST)* is developed that acts as a graphical front-end via which an user can intuitively launch and evaluate the performance of LMBD along with other scheduling algorithms. All interactions with the user is graphical making the entire simulation process highly interactive. Simulation inputs such as DAGs and outputs such as schedules can be viewed graphically from within the tool itself. The tool also incorporates a DAG builder.

1.3 Organization of this Dissertation

1. Chapter1: The current chapter; provides an introduction to the research problem this dissertation has addressed. Research contributions are summarized as well.

2. Chapter2: Provides background information to fully appreciate the research contributions of this dissertation. BSN, the multi-processor scheduling problem, battery behavior, battery modeling and DC-DC converters are introduced. The distinction between energy-awareness and battery-awareness is established.

3. Chapter3: A literature review of the battery-aware multiprocessor scheduling problem is provided in this chapter.

4. Chapter4: The physical architecture as well as the application model of the BSN under study are formalized in this chapter. Various timing parameters related to the scheduling problem are introduced and defined as well.

5. Chapter5: Passive Voltage Scaling; a DC-DC converter free design methodology is introduced here. The suitability of this methodology to the BSN under study is explored. Hardware assumptions such as the operating frequency as well as the voltage of the BSN sensor nodes are specified as well.

6. Chapter6: The research problem addressed by this dissertation is formalized. The literature survey of chapter 3 is revisited to compare and contrast the problem being addressed with problems that have been answered in the literature. It is then shown that the problem

addressed by this dissertation is hitherto not satisfactorily addressed in existing literature.

7. Chapter7: This chapter introduces and explains the algorithmic contribution of this dissertation. The working of the proposed algorithm; LMDB, as well as its internals are throughly explicated.

8. Chapter8: The computational complexity of the proposed algorithm is derived here.

9. Chapter9: Performance evaluation of LBMD via simulation studies are presented, analyzed and inferred.

10. Chapter10: The application tool GST is introduced and some of its salient features are elaborated.

11. Chapter11: Provides possible future research directions and concludes this dissertation.

Chapter 2

Preliminaries

This section provides broad but necessary background information to fully appreciate the rest of this dissertation. We will start by looking into BSN in greater detail. Next we will look into the energy and QoS-aware multi processor scheduling problem and highlight the nuances of the problem from a BSN perspective. Next we will focus our attention on the battery, explore its unique and non-ideal characteristics and understand their implication. We will establish the difference between "energy-aware" and "battery-aware" paradigms. Finally we will look into the DC-DC converter and its impact on battery performance.

2.1 What is a Body Sensor Network?

The term Body Sensor Networks was first coined by Prof. Guang-Zhong Yang of Imperial College in 2002 [6] to distinguish a class of personal wireless networks for physiological monitoring from the more generic Wireless

Sensor Networks (WSN). The key distinguishing factor being the word *personal*. Whereas a WSN is primarily meant for environmental monitoring wherein a human is a variable within the environment, in BSN the human body is the environment to be monitored.

BSN consists of one or more sensor nodes, known as processing elements (PE), that are capable of wireless communication with one another. Each PE has the capability to acquire, process and disperse environmental stimuli, the stimuli typically being a biological process such as heart beat, blood pressure, posture etc. The PEs typically consist of a specialized low power microcontroller, a Radio Frequency (RF) module, a sensing element and an energy pack. The stimuli is acquired via the bio-compatible sensing element(s) which then is processed by the microcontroller and finally dispersed, if necessary, via the RF module. The BSN also contains a central node, typically a PDA, equipped with a general purpose microprocessor that acts as a gateway to facilitate information exchange between the BSN and any other external device or network, or within the various nodes of the BSN itself. Figure 2.1 shows an example of a BSN monitoring the physiological state of an athlete. In a typical application scenario, the various PEs collect, process and transmit local data to the PDA which then uploads this data to the internet for remote viewing by a doctor. In extreme cases the BSN itself could intelligently decide on an appropriate course of action to be taken (such as drug-delivery) without the intervention of the doctor.

Most applications of BSN require the BSN be operated around the clock, on a 24-7 basis. This imposes several design constraints and poses

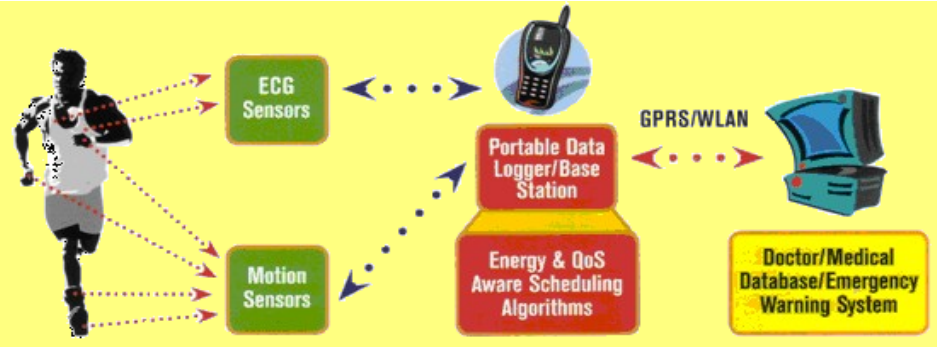

Figure 2.1: BSN employed to monitor health of an athlete

unique challenges. One of it is the choice of the energy pack. Being mobile and unobtrusive the energy pack of choice is predominantly the electrochemical battery. Due to the very application nature, considerable effort needs to be expended in replacing or recharging the batteries of the sensor nodes and in the case of implantable sensors this process could even be impossible. Thus, attention needs to be devoted to make the BSN not just energy efficient in terms of energy consumption but also to utilize every amount of energy available at the various nodes.

2.2 The Multiprocessor Scheduling Problem

A multiprocessor system can be described as centers of computation interconnected by an information exchange medium. A BSN is an example of such a system wherein the PEs form centers of computation and wireless communication forms the information exchange medium. Application software can be represented by means of a DAG. A directed graph with no directed cycles is termed as a *Directed Acyclic Graph(DAG)* and is an

abstraction that is used to capture flow and dependency relationships in software. Figure 2.2 shows example of a DAG.

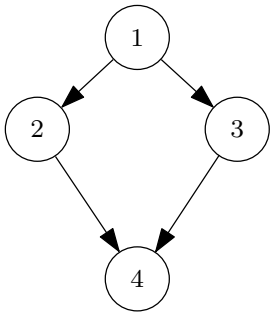

Figure 2.2: A Directed Acyclic Graph

A real-world BSN application would be the fall pre-detect software [1] whose DAG is shown in Figure 2.3. The nodes of the graph represent atomic computations that need to be executed in entirety. Edges between nodes capture precedence relationships which determine a partial ordering on the execution of nodes. Nodes that do not have any dependency relationship can be executed in parallel, provided, resources as required for parallel execution are made available.

Given a DAG and a multiprocessor system, the multiprocessor scheduling problem can be described as determining the exact allocation and order of execution of nodes on the individual processors such that a predetermined performance metric is optimized. Figure 2.4 shows a feasible schedule for the DAG in Figure 2.2 for a multiprocessor system consisting of two homogeneous computational centers. The performance metric mentioned previously could be the makespan of the final schedule, energy consumed by the computing centers, real time performance, QoS provisioning(deadlines) etc. or a combination of one or more individual metrics.

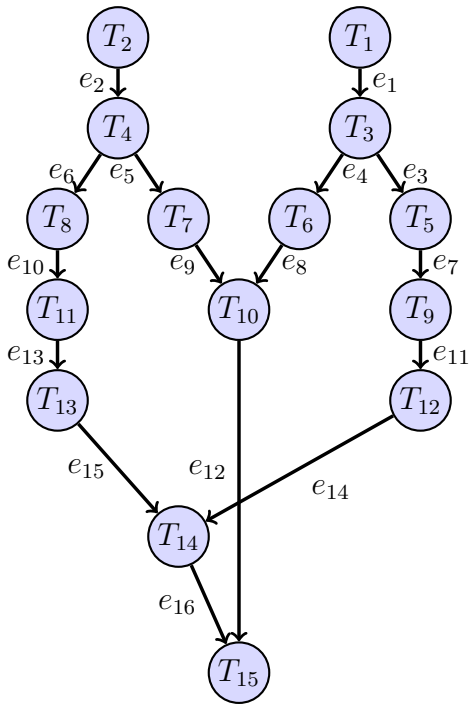

Figure 2.3: Fall-Detect application DAG[1]

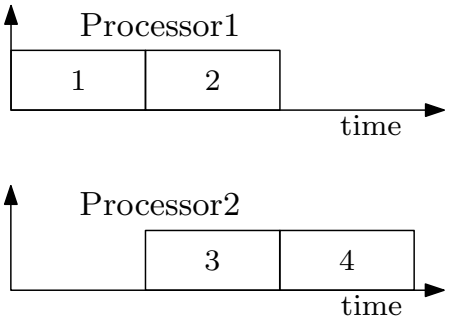

Figure 2.4: A schedule for the DAG in Figure 2.2

Additional constraints might be imposed on the scheduler as a results of system topology. For example, BSN is a loosely coupled system wherein communication costs, in time as well as energy, between PEs is non-negligible and cannot be ignored as in Figure 2.4. The wireless communication protocol could also impose additional constraints such as minimum hop-length. The PEs of BSN are inherently heterogeneous and could fur-

ther constrain the scheduler by imposing architectural constraints whereby certain tasks are executable on certain PEs only.

Chapter 6 provides a more formal treatment of the scheduling problem that this dissertation is dealing with.

2.2.1 Intractability of the scheduling problem

The multiprocessor DAG scheduling problem has been shown to be NP-complete in its most general form and also in several restricted versions[7]. Polynomial-time solutions can be found only for heavily simplified cases of the problem [8]. However, such heavy simplifications render the problem non-representative of the actual physical problem that is being attempted to be solved. When polynomial time solutions are not available the general alternative would be to employ heuristics to arrive at a reasonable solution. Heuristics, being 'rules of thumb' in nature, might not be able to produce a globally optimal solution and tend to provide solutions that are locally optimal. The proposed algorithm in this dissertation adopts certain battery based heuristics to guide the scheduling process. The details of which will be provided in further sections.

2.3 The Electro-Chemical Battery

A battery consists of two electrodes called anode and cathode and an electrolyte that separates them. Figure 2.5 illustrates the internals of a battery.

The electrical current obtained from a battery is a product of electrochemical reactions occurring at the electrode-electrolyte interface. A battery is characterized by an open-circuit potential V_{OC}, i.e. the initial potential of a fully charged battery under no-load conditions. As the battery supplies charge to an external load, its output voltage V_{BAT} drops and below a certain voltage known as the cutoff voltage V_{CUT}, the battery disconnects from the load and is termed to have been depleted.

It should be noted that the energy supplied by a battery can be abstracted as the amount of charge the battery supplies. Consequently the terms energy optimization and charge optimization mean the same, i.e to maximize the amount of charge that can be extracted out of a battery.

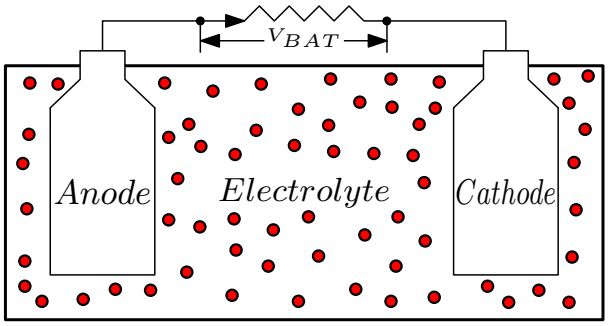

Figure 2.5: Simplified internals of an electro-chemical battery

2.3.1 Non-ideal characteristics of a battery

Two important characteristics contribute to the non-ideal performance of a battery as an energy source. These are:

1. Rate-Capacity effect

2. Recovery effect

Battery capacity can be described as the amount of charge that can be drawn out of a battery and supplied to an external load so as to get some work done. To put simply, rate-capacity states that the amount of charge that can be extracted out of a battery, to be supplied to a load, depends on the extraction rate. At higher rates this amount is lower and vice versa. Thus, if the total amount of charge inside a battery is C_T, the amount of charge that is available for useful work would be $C_U = \lambda C_T$, where $0 < \lambda < 1$. This implies a certain portion of C_T, given by $C_T(1-\lambda)$, is locked inside the battery and is unavailable for extraction. This behavior can be explained with the help of Figure 2.6. When fully charged, the electrode surfaces of the battery are in contact with maximum number of of active element(Figure 2.6(a)). When the battery is connected to a load, a current flows through the external circuit; active elements are consumed at the electrode surface which then are replaced from the electrolyte. This rate of replacement is usually lower then the rate of consumption and the replacement process cannot keep up with the consumption process. Due to this a concentration gradient builds up across the electrolyte (Figure 2.6(b)). As higher load current, resulting in higher rate of consumption, creates a higher concentration gradient and thus a lower concentration of active elements at the electrode surface. When this concentration falls below a certain threshold, which corresponds to the voltage cutoff voltage V_{CUT}, the electrochemical reaction can no longer be sustained at the electrode surface (Figure 2.6(c)) and the battery is said to have been exhausted. At this point, the charge that was unavailable at the electrode surface due to the gradient remains unusable and is responsible for the perceived reduc-

tion in battery capacity. This reduction in capacity can be interpreted as a certain portion of the battery's total charge that progressively gets trapped or locked inside a battery and which cannot be extracted and supplied to any external load.

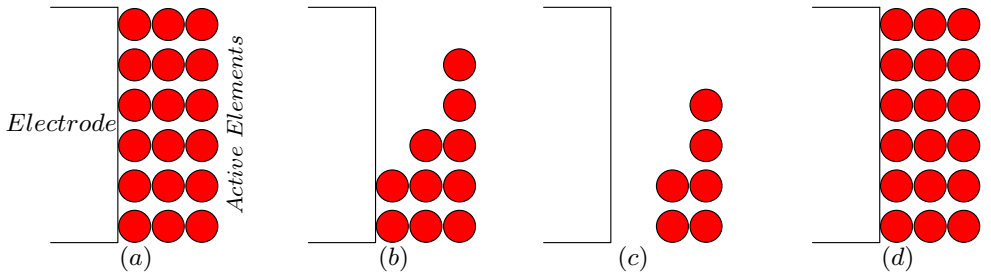

Figure 2.6: Charge gradient responsible for the rate-capacity effect. (a) Battery in pristine condition with no charge gradient (b) Charge gradient builds during powering of a load (c) Battery in exhausted state and unable to power any load (d) Charge gradient neutralized due to recovery effect

Figure 2.7 shows a snapshot of a battery in the midst of powering a load. The amount of charge that the load actually consumes is given by $I \cdot T$ where I is the current consumption of the load and T is the time duration of consumption. L represents the charge that has been locked due to the rate-capacity effect and A represents the net charge that is available for further extraction from the battery.

The charge that gets progressively locked as a result of the rate capacity effect is not physically lost but simply unavailable due to the lag between consumption and replacement rates. It is possible to recover this charge before the battery becomes exhausted. Decreasing the discharge rate effectively reduces the lag as well as the concentration gradient. If the batterys load goes to zero, the concentration gradient flattens out after a sufficiently long time, reaching equilibrium again (Figure 2.6(d)). The concentration of active elements near the electrode surface following this rest

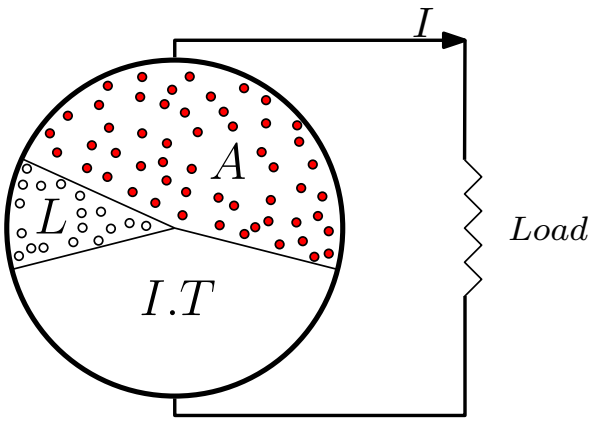

Figure 2.7: Charge distribution inside a battery while powering a load

period makes some previously unavailable charge available for extraction. This phenomenon is what is known as recovery-effect. Figure 2.8 illustrates this phenomenon. Certain portion of the locked charge(L) inside the battery gets released during the idle period after task T_1 and gets added to the pool of available charge(A), thus becoming available for the subsequent task T_2. Also, left idle for long enough, all the locked charge inside the battery get released and become available for usage by any external load. The charge consumed(C) by T_1 and T_2 is of course lost from the battery forever.

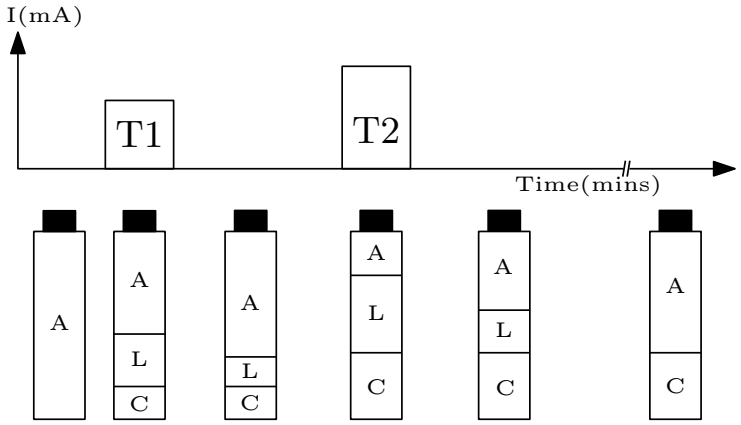

Figure 2.8: Movement of charge inside a battery

Apart from the above mentioned effects, the sequencing of tasks, as in the case of a multi-task application, is also found to affect the useful charge that can be derived from the battery. We call this as discharge-profiling effect. It has been shown in [9] that for a set of tasks of different current consumption, sequencing them in a strictly decreasing current profile results in maximum lifetime of the battery. For example consider Figure 2.9. Profile (d) which has three tasks in a strictly decreasing order of current consumption will result in minimum amount of charge getting locked inside the battery, thereby leading to greater amount of charge being available to any future task that might get scheduled at the end of the existing profile. Whereas profile (a) which has a strictly increasing current profile would lead to maximum charge lock-up and minimum available charge for any future task that might get scheduled at the end of the existing profile. Profiles (b) and (c) would fall in-between. The above mentioned example assumes that the tasks are independent and can be executed in any order. In a practical application, such as our BSN application, that enforces dependency among tasks, obtaining a strictly decreasing current profile might be impossible.

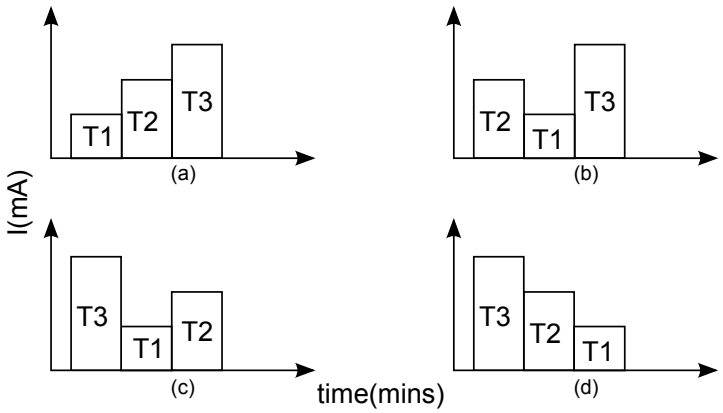

Figure 2.9: Different sequence of tasks on a PE

Rate-capacity, recovery and discharge-profiling are exploited in our algorithm to profile the discharge current in a manner conducive for extending battery lifetime.

2.3.2 Battery modeling

Various battery models are available in literature, each with its own merits and intended audience. Examples would be [10][11][12][13]. We have adopted the high-level analytical battery model of [9] owing to its high accuracy, ease of use and capability to provide deep analytical insight. The adopted model is aimed towards system designers, as opposed to battery designers, and provides an elegant, simple and accurate means to estimate battery properties and dynamic-state such as the available charge, locked-up charge and battery life-time. The model has been extensively tested by the developers and found to be accurate to within $\pm 3\%$ of actual charge and lifetime measurements of lithium-ion batteries. Lithium-Ion is the battery chemistry of choice for powering the constituent PEs of our BSN.

The adopted battery model is given as:

$$\alpha_I = \sum_{k=0}^{u-1} I_k F(L, t_k, t_k + \Delta_k, \beta) + I_u F(L, t_u, L, \beta) \tag{2.1}$$

where $F(x, y, z, \beta)$ is given as:

$$z - y + 2\left[\sum_{m=1}^{\infty} \frac{e^{-\beta^2 m^2 (x-z)} - e^{-\beta^2 m^2 (x-y)}}{\beta^2 m^2}\right] \tag{2.2}$$

The model requires two parameter α_I and β to model a particular battery. These two parameters can be obtained by means of a series of constant-load tests done on a sample battery of the type to be modeled. Once these two parameters are obtained the battery type is completely characterized and analysis of further samples can be done by using the model. α_I gives the total charge inside the battery when in pristine state and β is a measure of the non-ideality of the battery is. In our scheduling algorithm it is assumed that these constant-load tests have been done and all parameters necessary to model the battery are available at the beginning of the scheduling process.

The proposed scheduling algorithm utilizes the battery model introduced above to keep track of the states of the batteries in the BSN and to arrive at scheduling decisions. Various battery related parameters are used by the algorithm. Table 2.1 lists these parameters and defines them.

It should be noted that the adopted battery model does not give any explicit treatment to V_{BAT}, the output voltage of the battery and abstracts it in the form of α_A. V_{BAT} tends to drop as the battery enters deeper states of discharge and α_A at which V_{BAT} is equal to the lowest voltage at which a PE can operate is taken as the cut-off voltage V_{CUT} and the corresponding α_A is represented as α_{CUT}. α_{CUT} as well as α_A corresponding to various V_{BAT} can be easily found prior to the scheduling process when the modeling parameters α_I and β are found.

Table 2.1: Definition of Battery Parameters

Symbol	Definition
α_I	*Initial* charge inside a battery. This is the amount of charge a pristine battery would have. Measured in mA-mins [a].
α_T	*Total* charge inside a battery at any given point in time. This charge would be equal to the sum of α_A and α_L.
α_A	Charge inside a battery that is *Available* for use by an external load. This parameter can take negative values in the battery model. Such a case would imply that the battery has been over exerted. However a negative α_A does not mean the battery is dead as a significant amount of charge could be locked up inside the battery, waiting to be released.
α_L	Total charge that's *Locked* inside a battery that is not available for use by an external load. This charge is slowly released within the battery and adds to α_A with progress of time.
β	Characterizes the charge lock-up and recovery effects of a battery. A lower value of β indicates a higher deviation from an ideal energy source, resulting in a very high rate of charge lock-up and a large α_L, and vice versa. A lower value of β also indicates a low rate of recovery of α_L. β^2 is measured in sec^{-1}. A typical value of β for a lithium ion type battery, as measured in [11], would be 0.574.
SF	*Speculation factor*, computed as $\beta/(\beta+1)$.
α_{SA}	A speculative quantity that is indicative of the amount of charge that might be available in the battery in the immediate future. This *Speculatively Available* charge is given by $\alpha_A + SF \cdot \alpha_L$
α_C	The amount of charge that is *Consumed* by an event when it runs to completion on a PE. For example, the α_C of n_j on P_i is computed as $I_{P_i}^{n_j} \cdot T_{P_i}^{n_j}$ and gives the lower bound on the charge used by n_j to run till completion on P_i. α_C is used to execute the event and is non-recoverable and therefore lost from the battery forever.
α_U	The amount of charge *Used* by an event when it runs to completion on a PE. This quantity is given as a sum of the charge consumed and locked by the event. As mentioned earlier the charge consumed by the event is lost from the battery forever whereas the charge locked would eventually be released and added to the pool of α_A. α_U for a particular event can be obtained using the battery model that was introduced earlier.
HF	*Health factor*, having the range $0 < HF < 1.0$, is used as a threshold to assess the health of a battery. The health of a battery is defined as α_{SA}/α_T and gives the ratio of the charge that is available to an external load in the immediate future to the total amount of charge inside the battery.

[a] All charge quantities are measured in mA-mins

2.4 Battery Awareness Vs Energy Awareness

Battery-awareness needs to be incorporated into any energy optimization scheme that involves batteries as sources of power. The effectiveness of

any such scheme that excludes non-ideal behavior of the battery comes under question. We will now establish this important observation with an example.

Consider a multiprocessor system with two heterogeneous centers of computation. Let the two centers be powered by identical batteries with $\alpha_I = 20$mA-mins and $\beta = 0.374$. Consider an application DAG consisting of four heterogeneous nodes as shown in Figure 2.10(a). Assume inter task communication costs to be negligible for the sake of brevity. The current and execution time requirements of the DAG are given in Table 2.2. It is clear that tasks T_1 and T_2 consume less amount of charge (given as a product of current consumption and execution duration) on C_1 as compared to C_2. An energy-aware, in other words, battery-unaware scheduler might produce an energy-optimal schedule as shown in Figure 2.10(b). However, closer scrutiny of the schedule using the battery model given by Equations (2.1) and (2.2) reveals some startling results. The battery of PE1 is found to get exhausted at time 18.5mins which results in tasks T_2 and T_3 being incomplete. The battery of PE1 had an initial total charge of 20mA-mins which ideally is sufficient to run T_1, T_2 and T_3; whose total charge requirement is only 15mA-mins, to completion. However, due to charge lock-up phenomenon 10.73mA-mins of charge becomes unavailable to any external load and PE1 fails to complete its scheduled tasks.

Table 2.2: Execution requirements for the DAG in Figure 2.10

Task	**Current drawn on P1(mA)**	**Current drawn on P2(mA)**	**Execution time on P1(mins)**	**Execution time on P2(mins)**
T1	0.5	0.6	10	10
T2	0.5	0.6	10	10
T3	0.1	0.1	10	10
T4	0.1	0.1	10	10

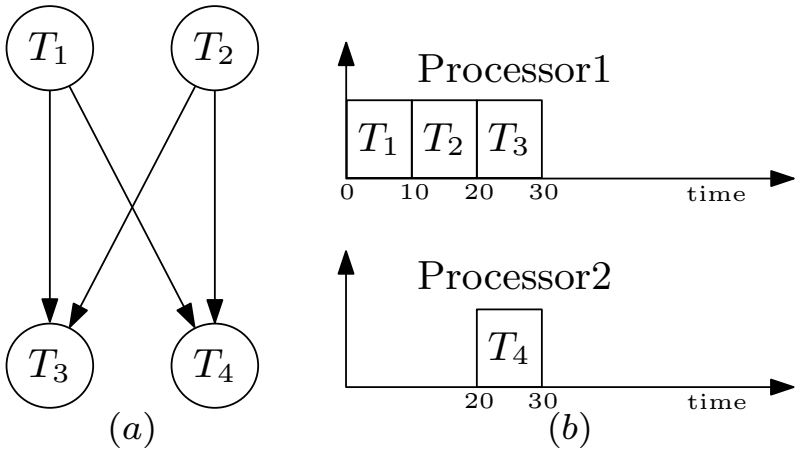

Figure 2.10: A DAG and its optimal schedule produced by an energy-aware scheduler

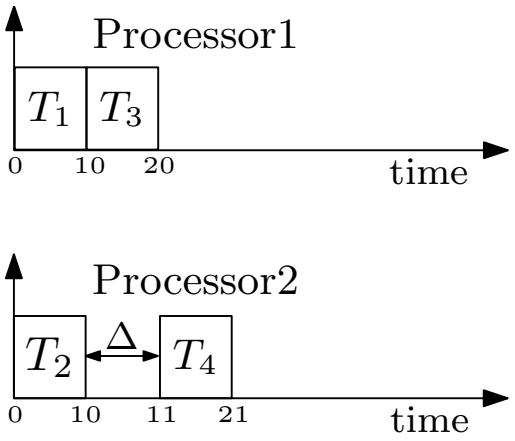

Figure 2.11: A schedule for the DAG in Figure 2.10(a) produced by a battery-aware scheduler

On the other hand, a battery-aware scheduler might produce a schedule shown in Figure 2.11. Task T_2 is scheduled for execution on C_2 though consuming higher charge than on C_1. A small rest period of duration 1min is inserted at time 10mins to allow for charge recovery at C_2. Scrutiny of the battery-aware schedule using the battery model reveals no surprises; tasks T_1, T_2, T_3 and T_4 are run till completion on the multiprocessor system with no PE suffering from exhaustion.

The above given simple, though being very simple, succinctly illustrates the unique challenges posed due to non-ideal battery behavior and shows why an energy-aware scheme does not necessarily mean a battery-aware scheme. Battery-awareness needs to be explicitly imparted to any energy-optimization scheme when the choice of underlying energy source is a battery.

2.5 The DC-DC Converter

A DC-DC converter is a switching circuit that interfaces the battery to the hardware to be powered. It usually consists of energy-storage elements such as capacitors to transfer power from the source to the destination. Most of the commercially available converters are suited for medium or heavy loads [14]. Ultra low power chips that are typically used in sensor networks operate on currents less than 10mA thereby operating the DC-DC converter in a very inefficient range. Losses incurred within this range can be as high as 40%[15]. Generally the efficiency of a DC-DC converter is found to decrease at low load currents and depends on the output voltage demanded as well. Powering a low power sensor node via a DC-DC converter would thus result in significant energy losses within the converter which cannot be ignored.

Another crucial factor to be considered is the impact the converter would have on battery performance. The output power P_{out} and the input power P_{in} of a converter are related as $P_{out} = \eta P_{in}$, where η is the converter efficiency. P_{in} can be expressed as $V_{BAT} \cdot I_{BAT}$, where V_{BAT} and I_{BAT} are

the battery output voltage and current respectively. At a given battery voltage, the current drawn by a converter from the battery depends on the output power demanded by the external load. At a constant demanded output power the DC-DC converter has to draw an ever increasing current from the battery in order to supply the demanded power. This is due to the fact that as the battery discharges, i.e. as current is drawn from it, its output voltage decreases and the converter now as to draw more current in order to meet the demanded output power. The result of this would be an ever increasing rate of battery discharge. As mentioned earlier, a battery is a non-ideal energy source which suffers from charge lock up phenomenon whereby a significant amount of charge gets locked within the battery thus becoming unavailable for any external load. The amount of locked up charge depends (among other factors) on the amount of current being drawn from the battery. It is thus evident that once the discharge current has reached sufficiently high levels the remaining life time that can be extracted out of the battery is minimal.

This drastic reduction in life time could have been avoided if one is able to regulate the amount of current drawn from the battery. With a DC-DC coveter interfacing the hardware and battery, this regulation becomes difficult to achieve. Even if this could be achieved, the losses within the converter would still reduce the effectiveness of any energy/battery optimization scheme. Thus, attention needs to be devoted to mitigate the detrimental effects of the DC-DC converter. Moreover the affordability of having an output adjustable dc-dc converter in terms of space and cost needs to be questioned when dealing with low-cost, high-volume, size-restricted hardware such as the sensor nodes of the BSN.

Chapter 3

Literature Review

Various scheduling algorithms have been proposed in literature to tackle the battery-aware multiprocessor scheduling problem. A summary of such work is provided in Table 3.1. We denote the hardware platform for an algorithm using the following notation. The energy source (battery) is referred to as *source* and the energy consumer (micro-processor/controller) is referred to as *sink*. Consequently, we have the following four combinations:

1. SSSS (Single Source Single Sink)
2. SSMS (Single Source Multiple Sink)
3. MSSS (Multiple Source Single Sink)
4. MSMS (Multiple Source Multiple Sink)

We will now proceed to categorize and survey the relevant algorithms.

3.1 Dynamic Voltage Scaling Approach

Dynamic Voltage Scaling (DVS) based algorithms have been the predominant approach to address the battery optimization problem. DVS, when supported by the underlying hardware platform, is a powerful and flexible tool that is employed to scale the operating voltage as well as operating frequency of a processor. This reduces current consumption of the processor and consequently results in energy savings. The reader is referred to the Appendix C for a more formal treatment of DVS.

Chowdhury *et al.* propose static multi-PE scheduling algorithms in [16]. To the best of our survey their work is the first in literature to address the multi-PE battery-aware scheduling problem. The proposed algorithm is suitable for real-time applications consisting of independent as well as dependent tasks that are either periodic or a periodic. The proposed algorithm operates in two steps. In Step1, a battery and deadline aware list scheduler is used to produce an initial feasible schedule. The initial schedule strives to achieve a decreasing load current profile, and ensures that all tasks run to completion. In Step2, voltage scaling is extensively used to scale the tasks towards their deadlines. The scaling process starts from the last task and proceeds towards the start of the schedule to endures a decreasing current profile. The performance of the algorithm was evaluated by means of synthetic load profiles, generated by observing the current consumption of a pocket computer.

Cai *et al.*[17] propose improvements to [16]. The researchers observed that slack reclamation procedure in [16] operated on each PE individually

and thus might not lead to a globally optimal schedule. The authors improve the slack reclamation procedure by dividing the given schedule into multiple steps. A step being defined as one or more tasks, each residing on a different PE and sharing a common period of execution along the timeline. An iterative algorithm is then used that incrementally allocates slack to the individual steps such that the global discharge current is minimized.

Jameel *et al.* proposed one of the earliest dynamic battery-aware schedulers in [18]. Their work is based on the principle of slack forwarding. Slack forwarding can be described as follows: slack generated by the early completion of a task can be allocated to its successor if the release time of the successor is earlier than or equal to the time point at which the slack originated. The proposed algorithm operates in two steps. Step1 is an offline process whereby an initial valid schedule is obtained based on the worst case execution time (WCET) of all tasks in the task-set for one hyper-period. Step2 is an online process whereby any slack generated during run time is forwarded to the next task. Upon completion of a task, the finishing time of the next task, which is based on Step 1, in the run queue is computed. If the finish time is lesser than the release time of the subsequent task, the voltage level of the task is adjusted such that slack is allocated to it. If otherwise the task runs at its assigned voltage level, thereby forwarding the slack to the subsequent task in the queue.

An iterative algorithm suitable for FPGAs is proposed in [19] by Khan *et al.* The researchers term a particular DVS voltage/frequency setting of a processor or a particular hardware configuration of a FPGA as a design point. A heuristic list scheduler produces an initial valid schedule on the

various PEs. The thus scheduled tasks are then assigned design points iteratively until a battery cost function as given by equation 2.1 is minimized. To reduce the complexity of the design space search, the authors also propose the use of a window function that iteratively expands the design space to be searched. The proposed algorithm is evaluated on a real-world robotic arm controller application on a DVS platform and the authors report improvements in battery lifetime as high as 65%.

3.2 Other Approaches

Peng Rong *et al.* [20] have integrated a general-model of a power managed electronic device with that of its power source, and present a unified model based on continuous-time markovian decision process. To the best of our knowledge this work is the first of its kind. A Battery-Aware Power Management policy (BAPM) is then developed based on the unified model. The BAPM technique aims to maximize the battery life span while meeting all timing requirements. As a result of using the unified model, the BAPM technique to able to arrive at decisions not just based on incoming task features but also based on the state of batteries powering the system. The BAPM dynamically selects the operating mode of the system (operating frequency) as well as the battery that is to power the system (for multi battery systems). The battery-aware dynamic power management problem is formulated as an optimization problem which is then solved by linear programming technique. Simulation results of up to 20% improvement in battery life is reported.

Luo *et al.* propose battery-aware static schedule transformations for DVS enabled and non-DVS processors [21]. Based on results from [22] and [23], the researchers employ Peukert's law [23] to arrive at schedule transformation schemes that results in reduced peak power and reduced variance in discharge current distribution. Two schemes, Local-transformation and Global-shifting are proposed. An initial feasible schedule is formed satisfying deadline and precedence constraints by any multiprocessor scheduling algorithm. Global-shifting is performed on this initial schedule followed by local-transformation to induce battery-awareness into the schedule. The transformations modify the start times of the task in the schedule such that the cost function given as:

$$p^{act} = \frac{1}{hyperperiod} \int_0^{hyperperiod} \frac{p(t)}{c_p(t)}$$

where p^{act} is the actual average power drawn out of the battery, $p(t)$ is power consumption at time t and $c_p(t)$ is the battery utilization factor, is minimized. For DVS systems, global shifting and local transformation are performed to allocate optimal slack time to the tasks, which is then followed by voltage scaling. Simulation results report improvements in battery lifespan up to 29% for the non-DVS case and up to 76% for DVS case over battery-unaware schedules.

Table 3.1: Summary of the Surveyed Work

System Level Design	**Targeted Hardware**	**Battery Technique**	**Type**	**Application Model**
J. Ahmed and C. Chakrabarti [18]	SSSS, SSMS, DVS-PE	Discharge current profiling via slack reclamation	Dynamic	N.A for single PE case. Independent task set for multi PE case
J. Luo and N. K. Jha [21]	SSMS, DVS-PE, Non-DVS PE	Discharge current profiling	Static	N.A
P. Chowdhury and C. Chakrabarti [16]	SSSS, SSMS, DVS-PE	Discharge current profiling via slack reclamation	Static	Independent periodic and aperiodic task set for single PE case. Periodic and aperiodic DAG for multi PE case
Y. Cai *et al.* [17]	SSMS, DVS-PE	Discharge current profiling via slack reclamation	Static	Task Graph
J. Khan and R. Vemuri [19]	SSMS, DVS-PE, FPGA	N.A	Static	Task Graph
P. Rong and M. Pedram [20]	SSSS, MSSS, DVS-PE	N.A	Offline	N.A

Chapter 4

System Modeling

In this section we formalize the physical and application model of the BSN under study.[1]

4.1 Physical Topology Model of the BSN

The physical topology of our BSN is represented as an un-directed graph $G_T = (P, L)$. The nodes of G_T, denoted by set P, consist of the individual processing elements (PE) and the central gateway. Such a node is denoted as P_i. The total number of nodes in G_T is denoted as N_T. The edges in G_T, denoted by the set L, constitute the communication links among the nodes of the network. Such a link is denoted as L_{ij} and represents a bidirectional communication link between nodes P_i and P_j. The communication links though bi-directional are not of full-duplex capability. A node's RF module can either be transmitting or receiving but not both simultaneously.

[1]This is the BSN that is in use by the embedded hybrid systems phase II project funded by A*STAR, Singapore and this dissertation component attempts to design and evaluate performance of static algorithms

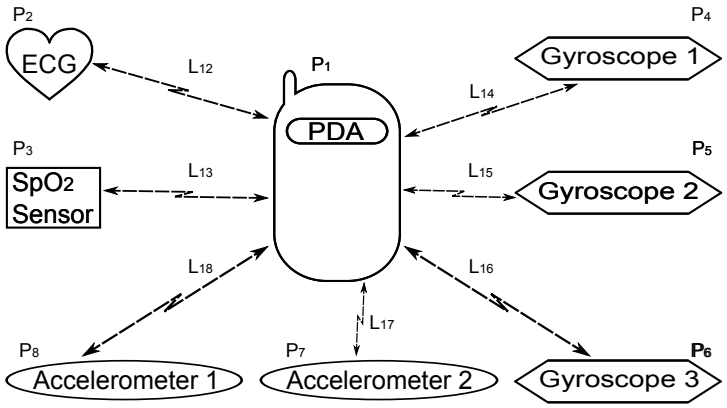

Figure 4.1: An abstract representation of the BSN

The physical architecture under study is a *star* topology. Figure 4.1 represents an abstraction of the physical topology of our BSN. The central node in the star network is the gateway. Other nodes in the network communicate with one another through this central gateway. Thus, a communication hop length of two is required for data transfer between any two PEs in the network. Each PE has a dedicated communication module and is capable of communicating (transmitting or receiving) to and from the central gateway while simultaneously executing some application task. Communication and task execution are non-preemptive. Also, the gateway is able to communicate with one and only one PE at any given point of time. Simultaneous communication with multiple PEs is not supported by the communication protocol. The gateway is assumed to have infinite energy at its disposal and is excluded from the scheduling process. Our battery-aware algorithm is restricted to scheduling the BSN application on to the PEs only. The gateway is excluded from the scheduling process and is abstracted as a communication link that is used to facilitate data transfer between the PEs of the BSN.

4.2 Application Model for the BSN

The application to be scheduled is represented as a DAG $G = (V, D)$. An example is shown in Figure 4.2. The nodes of G, denoted by the set V, form a collection of discrete tasks that taken together constitute the application at hand. The discreteness of a task imposes the restriction that it cannot be broken down into smaller tasks and it should be completed in its entirety on a single processing node. We denote such a task as n_i and the total number of tasks in G is represented as N. The terms node and task are used interchangeably with respect to G from here onwards. The set of directed edges that constitute D represent dependency relationship among the tasks in V. An edge $e_{ij} \in D$, is said to be directed from task n_i to n_j and represents the precedence constraint of n_j. This means, n_j cannot start until n_i is completed. Also, scheduling n_i and nj on different PEs necessitates the need for data transfer from the PE on which n_i is scheduled to the PE on which n_j is scheduled. This data transfer is non-negligible in time and energy requirements. A task can have more than one predecessor and for each such predecessor scheduled on a different PE, data transfer needs to be scheduled.

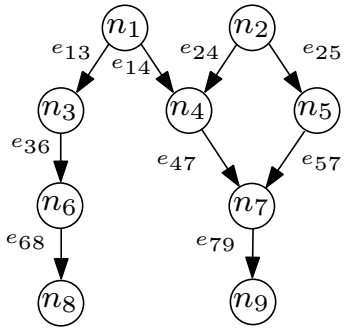

Figure 4.2: An example of a BSN application DAG G

Characterizing nj is a tuple of length N_T. Each element of the tuple in turn is a set of elements that denote the worst case current consumption (WCCC) and worst case execution time (WCET) requirements of n_j on a particular PE. Architectural constraints whereby a particular task is non-executable on a particular PE are captured in form of entries denoting infinity in the tuple. Task n_j is also associated with a deadline, denoted as D_{nj}. The deadline of a task gives the maximum time (with respect to the start time of G) before which the execution of the task must complete.

An edge e_{ij} is also associated with a tuple of length N_T. Each element of the tuple in turn is a set of elements that denote the current and execution time requirements of e_{ij} on a particular PE. For example, the entries $I_{P_i}^{e_{ij}}$ and $T_{P_i}^{e_{ij}}$ respectively denote the current and communication time requirement for data transfer, either transmission or reception, pertaining to the edge e_{ij} on P_i. It should be noted that communication costs is borne by the transmitting and receiving PEs only, and the gateway which routes this data is excluded as per the assumption made with regards to the gateway earlier. From this point onwards the term *event*[2] is used to denote a task or a communication.

4.3 Timing Definitions

The scheduling process requires definition of certain timing parameters as listed below:

[2]An event could be dormant or active depending on whether a task is lying idle or currently under execution by a PE. We refer to these states simply as an an event.

$$ST_{min}\left(n_i, P_j\right) = max\left(FT\left(P_j\right), DAT\left(n_i, P_j\right)\right) \tag{4.1}$$

$$FT\left(n_i, P_j\right) = ST\left(n_i, P_j\right) + T_{P_j}^{n_i} \tag{4.2}$$

Here, $ST_{min}(n_i, P_j)$ gives the earliest possible *start time* of n_i on P_j. $FT(P_j)$ gives the *finish time* of the last task scheduled on P_j or the finish time of the last *idle period*, if any, on P_j. Idle period insertion during the scheduling process will be elaborated in later sections. It may be noted that $FT(P_j)$ does not depend on the communications scheduled on the P_j but rather depends only on tasks or idle periods scheduled on P_j. $DAT(n_i, P_j)$ is defined as the *Data Available Time*. It gives the earliest time at which the data required by n_i becomes available at P_j, when one or more predecessors of n_i have been scheduled on PE(s) other than P_j. If all predecessors of n_i are scheduled on P_j or if n_i does not have any precedence constraint, $DAT(n_i, P_j)$ is set as zero. $FT(n_i, P_j)$ gives the time at which n_i would complete execution on P_j.

An example schedule for the DAG in Figure 4.2 is shown in Figure 4.3. The DAT for $n4$ depends on the availability of free slots in the communication links. Note the hop length of two that is required before data from PE2 can reach PE1. Also note that communications cannot overlap with one another since the physical topology imposes the restriction that only one PE can be engaged in communication with the central gateway at any given time.

The current consumption of a PE at any time can be taken as the summation of the current consumed by its processor executing a task, and the

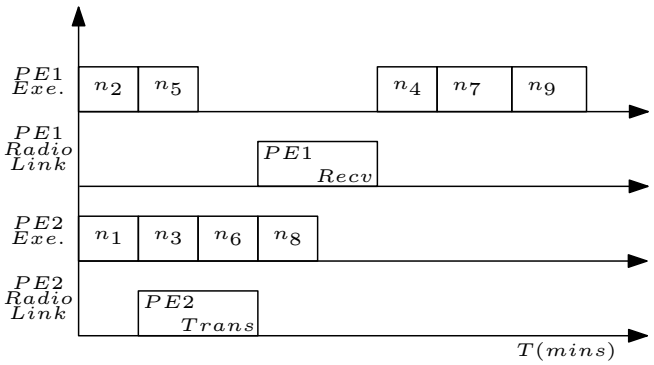

Figure 4.3: An example schedule for the DAG in Figure 4.2

current consumed by its RF module communicating with the gateway. The current consumption is assumed to be negligible when no task or communication activity is scheduled on the PE.

Chapter 5

Passive Voltage Scaling

In a typical portable embedded system, a DC-DC converter interfaces the battery to the hardware to be powered. The output voltage of a battery is not constant and depends on the residual charge inside the battery. A converter circumvents this variation and supplies the hardware with a steady operating voltage; stepping-up or stepping-down the input battery voltage as necessary. Most hardware circuits require strict operating voltage levels and cannot be operated without a DC-DC converter. Moreover, an embedded device could consist of heterogeneous components demanding different operating voltages. Under these conditions a DC-DC converter becomes an absolute necessity.

Significant energy losses occur inside the DC-DC converter during the conversion process. The negative impact of the converter on the battery especially in a low power device such as a sensor node was already established in Chapter 2. The sensor nodes of the BSN are no different. The microcontroller in the sensor nodes of our BSN is the TIMSP430 which

has a peak current consumption of only 0.6mA while the RF module has 18.8mA. Under these low current situations losses inside the converter can be as high as 40% [15]. Thus, an effective means has to be adapted to mitigate the negative impact of the DC-DC converter.

5.1 Converter-free Hardware Design

The TI microcontroller of the BSN sensor node is a versatile device capable of operating under a wide range of input voltages and frequencies [24]. The acceptable voltage range spans 1.8v to 3.6v and the corresponding operating frequency spans 4.15 MHz to 8 MHz. The microcontroller though having excellent voltage-frequency scaling does not support dynamic voltage scaling. Thus, the traditional methodology of scaling the operating voltage and frequency to achieve energy-performance trade-off is not feasible. Having discounted the traditional DVS technique, possibility of employing other techniques that utilize this excellent voltage-frequency scaling open up. One such techniques is called Passive Voltage Scaling (PVS).

PVS is a technique that eliminates losses in power transmission devices by interfacing the energy source (battery) directly to the hardware to be powered. Energy-performance trade-off is achieved by operating the hardware at a frequency that is proportional to the output battery voltage. The battery voltage is constantly monitored via a monitor circuit and the operating frequency is scaled as the monitored voltage drops. PVS has been explored and studied in literature with regards to sensor networks [2] and has been found to prolong sensor node lifetime. Lifetime improvements

as high as 64% have been reported. Various other works in literature also lend support to the converter free design of wireless sensor nodes as in PVS [14][25].

In conventional power-supply setups that use a DC-DC converter, the processor current I_b is inversely proportional to the battery voltage V_b. In the configuration where the DC-DC converter is eliminated and the processor is operated at a constant frequency f_c, I_b is directly proportional to V_b. In PVS setup, the frequency is not held constant and is scaled according to the input voltage V_b. In such a case I_b is directly proportional to the square of V_b. It has been shown that among all the configurations PVS extracts maximum lifetime out of the battery [2].

A potential draw back to PVS is that the battery monitor requires additional hardware circuitry to be built into the sensor node leading to an increase in cost, size and complexity of the sensor node. Also the performance(in terms of maximum operating clock frequency) of the sensor node degrades as the battery voltage drops.

5.2 The Adopted Hardware Setup

We take cues from [2] and adopt a setup similar to PVS. The DC-DC converter is eliminated and the sensor node hardware is interfaced directly to the battery. The output voltage offered by the battery is assumed to span the range 1.8v to 3.6v and the TI microcontroller is operated at a constant frequency of 4.15 Mhz, which corresponds to its lowest operable frequency, through out this voltage range. The RF chip of the sensor node

is also capable of operating under a wide range of input voltages [26] and is assumed to be operable within the range offered by the battery powering the sensor node.

Figure 5.1(a) shows the operating frequency assigned to the TIMSP430 as a function of the battery voltage V_{BAT}. Figure 5.1(b) shows the operating current values that will be used during the scheduling process. The current consumed by an event when V_{BAT} is in the range 1.8v to 2.7v is taken as being equal to the current consumed at 2.7v. The current consumed by an event when V_{BAT} is in the range 2.7v to 3.6v is taken as being equal to the current consumed at 3.6v. Since the current consumed by an event drops as the battery output voltage reduces, we are essentially assigning an event its worst case current consumption with respect to V_{BAT}.

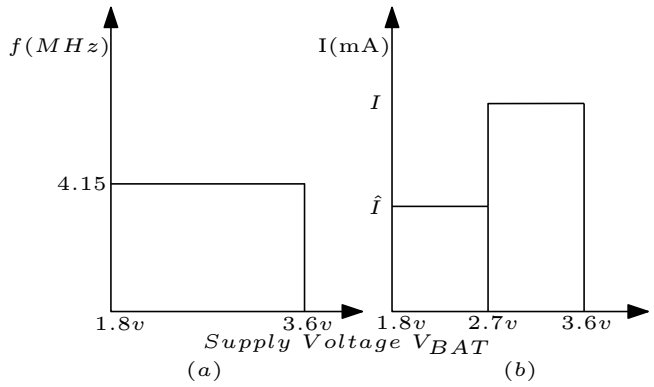

Figure 5.1: Assigned frequency and current consumption levels of the underlying hardware

To summarize, we adopt a PVS based setup for our sensor node as a viable alternative to the traditional converter driven power supply for the following reasons:

1. Converter losses, which could be as high as 40% for ultra low power devices, is completely eliminated.

2. Negative impact of the converter on battery discharge current is eliminated and consequently the discharge current can be predicted more accurately.

3. Battery models require accurate discharge current inputs to model a battery. Adopting PVS would thus enable us to utilize battery models with greater confidence.

4. The TI and RF chips of the sensor node are found to be versatile enough to operate under a wide range of input voltages.

5. DVS techniques are redundant.

Chapter 6

The Research Problem Formalized

We now formally present the research problem this dissertation has addressed.

6.1 Problem Statement

Given a multiprocessor system $G_T = (P, L)$ and an application DAG $G = (V, D)$, determine for each task $n_i \in V$ the parameters $ST(n_i, P_j)$ and $FT(n_i, P_j)$ where $P_j \in P$, such that $\mathfrak{S}$ is maximized and the following conditions are satisfied:

- For task $n_i \in V$; $FT\,(n_i, P_j) \leq D_{ni}$ (Deadline Constraint)
- For edge $e_{ij} \in E$; $FT\,(n_i, P_j) \leq ST\,(n_i, P_j)$ (Precedence Constraint)

- For $P_j \in P$, $\alpha_A \geq 0$ (Endurance Constraint)

The number of instances of G that are scheduled on G_T is represented as $\mathfrak{S}$. The deadline constraint serves QoS requirements of G. The Endurance constraint states that the battery of P_j must be 'alive' during the entire duration of the partial schedule on P_j. In other words, the battery of P_j must be able to supply enough charge to run all scheduled tasks and communications to completion.

$\mathfrak{S}$, the performance metric of choice is the number of instances of the application DAG G that can be run to completion on the BSN. That is, G is a periodic application with period being equal to $max(D_{ni})$ for $n_i \in V$. G is repeatedly scheduled on G_T until the BSN runs out of energy or suffers a deadline failure. At first glance it might seem that $\mathfrak{S}$ is a rather obscure performance metric for an algorithm that is trying to maximize charge utilization from a battery. The reason for adopting $\mathfrak{S}$ as the performance metric of choice are as follows:

1. We are interested in system lifetime. That is, the lifetime of the BSN as a whole rather than just the lifetime of its individual nodes. A BSN could suffer single or multiple node failures (nodes running out of battery) but might still be able to function if the remaining nodes take over the responsibilities of the inactive nodes. The scheduling algorithm must be able to discern nodes with exhausted batteries, discount them from the scheduling process and schedule the application on the remaining nodes.

2. The traditional performance metric for energy-aware schedulers has been the total energy consumed by an application. However, owing to the inherent difference between energy-awareness and battery-awareness, as pointed out in Chapter 2 , this metric cannot be used.

3. The traditional performance metric for battery-aware schedulers has been the total charge utilized by a single instance of the application DAG G. This again is not suitable for our scheduling algorithm since multiple instances of G are scheduled and each instance of the schedule might be different form another, utilizing different amount of charge.

4. The BSN nodes are to be operated under a PVS-like setup where the DC-DC converter is eliminated and the processing elements are operated at the lowest possible frequency. This means that an application under this setup would run for a greater amount of time till completion compared to the traditional setup that utilizes a DC-DC converter and operates the processing elements at a higher frequency. Thus, a question arises on whether it is beneficial to run the application at higher frequency so that it completes as soon as possible or at lower frequencies such that its completion time is delayed.

$\mathfrak{S}$ encompasses all the above given concerns and thus is the choice of performance metric.

6.2 Literature Survey Revisited

We now take a critical view of the literature survey presented earlier and show that the problem addressed by this dissertation; as given above, has not been dealt in its entirety in existing literature.

Firstly, the various battery-aware multiprocessor scheduling strategies in literature(Table 3.1), with the exception of [20], are designed for hardware architectures that share a common energy source. That is, even though the processing elements of the underlying hardware are distributed they draw energy from a common and single battery pack. Embedded systems, as it was pointed earlier, are becoming more and more pervasive where processing elements are not only distributed but also disjoint from one another. This means that each processing element is powered by its own dedicated battery pack. As with the processing elements itself, the batteries powering these also come in myriad types. Distributed and heterogeneous energy supplies impart unique challenges to the multiprocessor scheduling problem such as:

- Processing nodes with different energy capacity
- Processing nodes with different rate-capacity and recovery behavior
- The difference between 'energy-awareness' and 'battery-awareness' being exacerbated

which existing literature fails to address. As for [20], multiple energy sources are considered but only under a uniprocessor environment.

Secondly, existing strategies are predominantly designed for DVS enabled processing elements. These strategies employ discharge current reduction and profiling via voltage and frequency scaling. DVS though being a powerful energy saving tool is not available on all hardware platforms. As it was pointed earlier, the BSN of interest to this dissertation does not support DVS. Under such a case, non-DVS based optimization schemes need to be explored. Once gain there is a dearth of literature that accomplish this. [21] does propose a methodology for non-DVS processing elements. However, the methodology does not consider scheduling but rather given an existing schedule it proposes techniques to make the schedule battery friendly.

Finally, the performance metric used in existing literature is the total charge utilized by a single scheduled instance of the application DAG G. Owing to point 1 in the enumeration listed in section 6.1 this might not be an apt choice.

Chapter 7

The Proposed Algorithm

We now present the algorithmic contribution of this dissertation. We have developed an algorithm, which we call as List-Based Minimum Damage Battery Aware Scheduling Algorithm, to schedule an application DAG with deadline constraints and non-negligible inter-task communication costs, on to the heterogeneous processing elements that comprise a BSN. Various battery related parameters which were listed in Table 2.1 are utilized in the algorithm.

7.1 List-Based Minimum Damage Battery Aware Scheduling Algorithm (LBMD)

LBMD comprises the following three steps as listed below:

1. Data preprocessing (DP)

2. Damage assessment (DA)

3. Rescue operation (RO)

7.1.1 Data preprocessing (DP)

This step accepts as input the application DAG G and outputs the Normalized Charge Consumption (NCC) of the nodes and edges in G. The NCC value of a node or edge is computed for each PE on which it is executable. The current consumption as well as the time to completion of a task or communication might be different on different PEs, thereby leading to distinct NCC values with respect to each of the PEs. WCCC and WCET are used in computation of NCC.

The NCC of an event E with respect to a PE P_i is denoted as $NCC^E_{P_i}$ and computed as $\alpha_C/max(\alpha_C)$. Here, α_C is the charge consumption of E on P_i and $max(\alpha_C)$ is the maximum charge consumption among all events executable on P_i. Any task that cannot be executed on P_i as a result of architectural constraints is discounted from the computation of NCC. Let the computed NCC values form set $\mathbb{A}$. It should be noted that though DP computes NCC for all the edges in G, the edges that will eventually be part of the final schedule, i.e., the eventual communications between PEs in the final schedule is unknown at this point and will be known only after the completion of the second step.

DP also computes the *Average Normalized Charge Consumption* ($ANCC$) for all the nodes in G. The $ANCC$ of node n_i is represented as $ANCC_{n_i}$

and is computed by averaging the NCC of the nodes comprising the subgraph induced by n_i. Note that only the NCC of the nodes in the subgraph contribute to $ANCC_{n_i}$ and the NCC of the edges are ignored. $ANCC_{n_i}$ can be represented as:

$$ANCC_{n_i} = \frac{\sum_{P_j \in V_T} \sum_{n_z \in \mathfrak{N}} NCC_{n_z,P_j}}{|NCC_{n_z,P_j}|} \tag{7.1}$$

Here, $\mathfrak{N}$ represents the subgraph induced by n_z. Invalid NCC due to architectural constraints is discounted in the above computation. Here, $|NCC_{n_z}, P_j|$ gives the number of valid NCC included in the summations. Let the $ANCC$ values of all nodes form set $\mathbb{B}$.

At this point the terms *battery-failure* and *event-failure* need to be introduced. Consider P_j on which event E, with minimum charge requirement α_U, has been scheduled. P_j or E is said to have suffered battery-failure if the total charge available from the battery of P_j; α_T, is not sufficient to run E to completion. This would in turn imply that $\alpha_T < \alpha_U$. In this case, E is non-recoverable and the failure is permanent. Given that $\alpha_T \geq \alpha_U$, P_j or E is said to have suffered event-failure if the available charge inside the battery of P_j; α_A, becomes negative before the completion of E. This implies that the battery has been overexerted and is not able to supply enough charge to run E to completion. The point to note here is that, event-failure can be recovered by inserting some idle time period before the start time of E, i.e., by postponing the start time of E. This idle period would allow some α_L inside the battery to be released, due to recovery phenomenon, and be available to E. Thus, event-failures can be recovered by exploiting recovery effects whereas battery-failures are non recoverable.

7.1.2 Damage assessment (DA)

This step takes as input G, $\mathbb{A}$, $\mathbb{B}$, the battery speculation factor SF, the health factor HF and outputs a valid schedule S, satisfying deadline and precedence constraints as dictated by G. DA only attempts to satisfy the final constraint requirement, i.e., the endurance constraint, and does not guarantee that this constraint would be satisfied. Any violation of the endurance constraint, if detected, is attempted to be rectified in the final step RO. The battery model introduced previously is put to extensive use in DA to keep track of the state of the batteries powering the PEs of the BSN. At any point during the scheduling process, DA can call upon the model to find any of the state of a battery such as α_A and α_L. From this point onwards α_A would be used to represent the available charge in the battery with respect to the cut-off charge α_{CUT}. As mentioned earlier, at α_{CUT} the output voltage of the battery would reach V_{CUT} and the PE would no longer be operational. The α_A produced by the battery model is subtracted with the cut-off charge α_{CUT} and the result is taken as α_A. Thus, unless specified otherwise, any reference to α_A from here onwards has already taken α_{CUT} into consideration. DA follows the traditional list scheduling procedure and is depicted as a flow chart in Fig. 7.1. The flow chart does not indicate the exit points or termination points in the scheduling process for the sake of clarity. However, they are indicated in the description of the DA, which we now present.

A *ready list*(RL) is maintained consisting of the nodes in G that are deemed as being ready. A node n_i is said to be ready if all its predecessors have already been scheduled for execution or if n_i does not have any

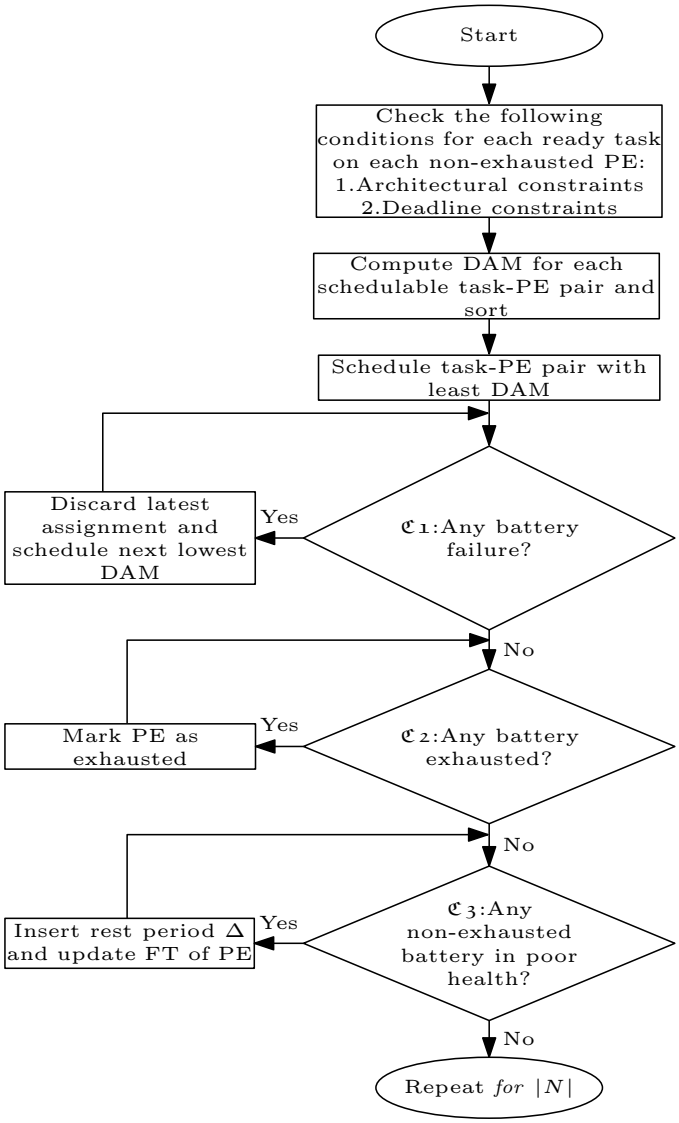

Figure 7.1: Flowchart of DA

predecessors at all. The list scheduler takes exactly N steps to schedule G. At each scheduling step an RL is formed. Consider a scheduling step where rl is non-empty, i.e., $rl = n_l, n_m, \ldots n_z$. Each non-exhausted $P_j \in V_T$ is iterated over $n_i \in RL$ and at each iteration architectural and deadline constraints are checked. A PE is termed as non-exhausted if its battery has enough charge to execute at least one of the nodes in G. The first feasibility check removes non-feasible assignments due to architectural constraints

as imposed by the BSN. The second feasibility check discounts any assignment that would lead to a violation of a node's deadline. The satisfaction of the deadline constraint is checked by means of equations (4.1) and (4.2); WCET are used in evaluation of these equations. If a node fails any one of the above two checks for all non-exhausted PEs, or if no non-exhausted PE can be found, the scheduling process stops and returns indicating *failure*.

A positive scalar value termed *Damage*(DAM) is then assigned to each of the feasible assignments. DAM is an indication of unsuitability of scheduling a node on a particular PE. The DAM assigned to a particular task-PE assignment depends on the partial schedule constructed so far on the PE. The task-PE assignment with the least DAM is then scheduled. Communication to fulfill precedence relationship, if any, is scheduled on the communication links. Note that the partial schedule of more than one PE gets modified if communication is scheduled.

The next step in DA is the battery-failure check which we denote as $\mathfrak{C}\mathfrak{1}$.$\mathfrak{C}\mathfrak{1}$ checks whether the latest assignment causes any battery-failure in any of the PEs. $\mathfrak{C}\mathfrak{1}$ is performed on all the PEs whose existing partial schedules have been modified due to the latest assignment. If any such PE is found to fail $\mathfrak{C}\mathfrak{1}$, the just made assignment is discarded and the next lowest DAM task-PE assignment is scheduled. $\mathfrak{C}\mathfrak{1}$ is performed again. DA returns failure if no task can be scheduled without violating $\mathfrak{C}\mathfrak{1}$.

$\mathfrak{C}\mathfrak{1}$ is performed as follows. Consider P_i whose partial schedule has been modified due to the latest assignment. Let the modified partial schedule be PS_i. For event $E \in PS_i$ such that E does not overlap with any other scheduled event on the timescale of P_i, in other words, P_i has not been

scheduled any other event between $ST(E)$ and $FT(E)$, $\mathfrak{C}\mathfrak{1}$ is checked as $\alpha_T \geq \alpha_U$. Here, α_T is the total charge of P_i computed at $ST(E)$ and α_U is the charge used by the event E. Obviously the given condition needs to be satisfied if E is to run till completion on P_i. $ST(E)$ and $FT(E)$, which denote the start and finish time respectively of E, would be known from PS_i.

For events in PS_i that do overlap with other scheduled events, all the overlapping events are grouped to form a block B and $\mathfrak{C}\mathfrak{1}$ is performed as:

Let $\alpha_A[ST(B)] = \alpha_T[ST(B)]$
  for each event $E \in B$
    if $(\alpha_A[FT(E)] < 0)$ ***return*** *failure*
  endfor
return *pass*

The first line of the above given pseudo code assigns the total amount of charge (available + locked) in the battery as the available charge. That is, an optimistic view is adopted by assuming that the block B has at its disposal the total amount of charge that can ever be derived from the battery. The ***for*** loop then checks whether each of the constituent events of B can be run till completion. If any of the constituent events of block B is unable to run till completion, $\mathfrak{C}\mathfrak{1}$ returns with a failure. The start time of B, denoted as $ST(B)$, is given as $\min(ST(E))$ and finish time of B, denoted as $FT(B)$, is given as $\max(FT(E))$, where $E \in B$. Fig. 7.2, an example of a partial schedule, illustrates $\mathfrak{C}\mathfrak{1}$ which is given in Table 7.1. Note that $\mathfrak{C}\mathfrak{1}$ need not be performed for the entire duration of the modified partial schedule but rather from the point of modification to the end of the partial schedule. The point of modification can either be the start time of an event on the partial schedule or the start time of a block on the partial schedule. Once a node has been scheduled passing $\mathfrak{C}\mathfrak{1}$ it is removed from G.

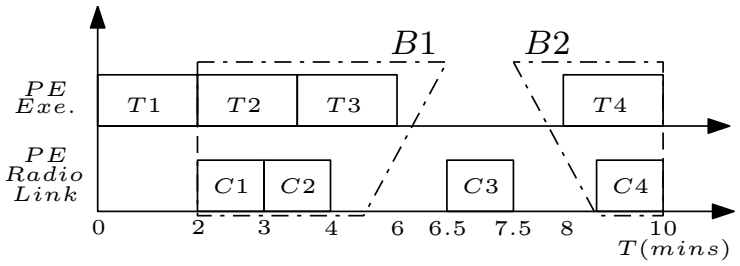

Figure 7.2: An example of a partial schedule

Table 7.1: Battery-Failure check for the partial schedule of Figure 7.2. (Numerical values in brackets denote time)

Event/Block	**Battery-Failure Check**
T1	$\alpha_T(0) \geq \alpha_U(T1)$?
B1	Let $\alpha_A(2) = \alpha_T(2)$; *then*, $\alpha_A(3, 3.5, 4, 6) \geq 0$?
C3	$\alpha_T(6.5) \geq \alpha_U(C3)$
B2	Let $\alpha_A(8) = \alpha_T(8)$; *then*, $\alpha_A(10) \geq 0$?

The next step in DA is termed $\mathfrak{C}_2$ and discounts PEs that have exhausted their batteries from subsequent scheduling steps. P_j is termed as exhausted if its α_T, is less then the minimum α_U among all unscheduled tasks in G. Thus, $\mathfrak{C}_2$ is checked as $\alpha_T < \alpha_U$; for all unscheduled $n_i \in G$. The α_T of all PEs is computed at the end of their respective modified partial schedules and one is termed as exhausted if the computed value is not sufficient to run any of the unscheduled tasks in G.

$\mathfrak{C}_3$ checks the health of PEs whose partial schedule have been modified due to the latest assignment. P_j is termed as healthy if it satisfies the condition $(\alpha_{SA}/\alpha_T) \geq HF$. The condition finds the ratio between the speculatively available charge from the battery to the total charge in the battery, at the end of the partial schedule of the P_j. Note that this partial schedule includes tasks and communications that have been scheduled at the current scheduling step of DA.

α_{SA} is an indication of the charge that might be available to an external load, in the immediate future.

The rationale behind this check is that at the end of a scheduling step a battery could end in a deep state of discharge where a significant amount of charge could be locked-up and the amount of available charge is minimum or even negative. Further scheduling of events when the battery is under such poor health is not desirable. If P_j is found to fail the health check, a small rest period Δ is inserted at the end of the partial schedule of P_j, postponing its finish time FT. The purpose of this idle rest period is to allow some of the locked charge to be released, as an effect of the recovery phenomenon and become available to any event that might be scheduled in subsequent scheduling steps. It should be noted that any event that might subsequently be assigned to P_j is scheduled such that it does not overlap with any rest period Δ that might be present in the partial schedule of the P_j. The total rest period to be inserted can be incrementally obtained using the battery model [27]. It is now evident from the manner in which health is calculated that any rest period Δ will be finite. If P_j satisfies the health check, no change is made to its partial schedule. HF and SF provide some insights into rest period period insertion. Refer to Appendix B for further details.

The above described scheduling procedure is repeated until all the nodes in G get scheduled or a failure is encountered. We state and prove certain important properties that are used in the design of our algorithm in Appendix A.

As mentioned earlier, we assign a positive scalar value termed *Damage*(DAM) to each of the feasible assignments. We now show how DAM is obtained and the rationale behind it.

DAM Function: The DAM for a task n_i with respect to P_j is denoted as $DAM_{P_j}^{n_i}$. It can be regarded as the damage a particular task n_i would do to the

batteries of the PEs in the BSN, if assigned to the P_j. Its an indication of the amount of charge the batteries powering the PEs of the BSN would expend in order to fulfill the execution of n_i on P_j, along with any communication that might arise to fulfill such an assignment. $DAM^{n_i}_{P_j}$ is composed of three functions that are listed and elaborated below.

$$\mathfrak{F}1\colon \frac{\left(NCC^{n_i}_{P_j}\right) \cdot PF^{n_i}_{P_j}}{\alpha_{SA}\left(P_j\right)} \tag{7.2}$$

Here, partial factor $PF_{P_j} n_i$ is given by $I_{P_j} n_i / I^{ps}_{P_j}$ where $I^{ps}_{P_j}$ is the average current consumption of the partial schedule on P_j. It is obtained by simply summing the current consumption of all events scheduled so far on P_j and dividing by the number of such events, to obtain the average. When a task and a communication overlap, their respective current consumptions are added together to give the total current consumption during the period of overlap.

$$\mathfrak{F}2\colon \frac{NCC^{e_{ji}}_{P_i} \cdot PF^{e_{ji}}_{P_i}}{\alpha_{SA}\left(P_i\right)} + \frac{NCC^{e_{ji}}_{P_j} \cdot PF^{e_{ji}}_{P_j}}{\alpha_{SA}\left(P_j\right)} \tag{7.3}$$

Here, n_j is a predecessor task of n_i that has already been scheduled on P_i. Thus, dependency data needs to be transmitted by P_i and this dependency data needs to be received by P_j. There will be zero or more instances of $\mathfrak{F}2$ depending on the number of dependencies of n_i and on the PEs they are scheduled on.

$$\mathfrak{F}3\colon \frac{ANCC_{ni}}{Ave\left(\alpha_{SA}\right)} \tag{7.4}$$

Here, $Ave(\alpha_{SA})$ denotes the average value of α_{SA} computed over all non-exhausted PEs.

In the above given three functions the quantity α_{SA} is computed at the end of the partial schedules of the respective PEs. It should be noted that even though α_{SA} used in the above functions are at the end of a PEs partial schedule, the exact time location where a task or communications gets scheduled might not be at the end of the partial schedule. For example, the task to be scheduled might rely on the arrival of dependency data from other PEs, which in turn might postpone its start time, communications could get scheduled before or after the end of the current partial schedule. Thus, α_{SA} gives an indication of the charge available at a PE at the end of its partial schedule and not at exact time-points where tasks or communications actually get scheduled.

$DAM_{P_j}^{n_i}$ is finally given as:

$$\frac{(\mathfrak{F}1 + \mathfrak{F}2) \times \mathfrak{F}4}{\mathfrak{F}3} \tag{7.5}$$

Rationale behind DAM of a task

As mentioned earlier, $DAM_{P_j}^{n_i}$ gives the unsuitability of scheduling n_i on P_j. The three functions given above capture this unsuitability from a 'charge-needed to charge-available' ratio point of view.

From the definition of α_{SA} it is clear that it speculates on the available charge from the battery of P_j, adopting an optimistic view based on the fact that the locked charge inside the battery α_L, would be released and added to the available charge α_A. The degree of optimism is given by the speculation factor SF, which indicates how close the batterys behavior is to an ideal energy source and how fast one can expect the locked charge inside the battery to be released. In the DAM function, α_{SA} is used as a normalizing factor. This is intuitive since a

battery with a higher α_{SA} would be in a position to take a task requiring more charge than a battery with a lower α_{SA}. Also, using α_{SA} as a normalizing factor captures the heterogeneity in the available charge at different PEs of the BSN during the scheduling process and ensures uniform wear-off of all the constituent PEs of the BSN.

The $NCC_{P_j}^{n_i}$ is closely related to the upper bound on the charge consumption of n_i. This is because $NCC_{P_j}^{n_i}$ is computed using the worst case current and execution time of n_i on P_j. The partial factor $P.F_{P_j}^{n_i}$ is used to scale this quantity. The importance of the partial factor becomes evident once we consider discharge current profiling. This importance of decreasing discharge current profile was already highlighted in the *Battery Behavior* section. The partial factor captures this profiling requirement as it is obtained as a ratio of the current consumption of n_i to the average current consumption of events already scheduled on P_j. Thus, n_i becomes more unsuitable to be scheduled on P_j if it consumes a high amount of current compared to the events in the partial schedule already on P_j and vice versa.

$ANCC_{n_i}$ gives an indication of the charge requirement of the sub graph induced by n_i. It is desirable to schedule a task whose children are heavy consumers of charge as early as possible. This would enable scheduling of the heavy children early. This helps to obtain a decreasing current profile on all the PEs. Also, scheduling these heavy consumers earlier on would put less stress on the batteries towards the end of the schedule and would facilitate easy rescuing of any task that fails the endurance constraint. Thus, $\mathfrak{F}3$ appears in the denominator and serves to lower the DAM of task n_i. $\mathfrak{F}3$ is similar to $\mathfrak{F}3$ and it captures the charge requirements for scheduling a single two hop communication instance.

The multiplicative term in the DAM function introduces deadline awareness and serves to lower the DAM of a task whose deadline is very tight. The tightness

of a tasks deadline can be due to two factors. One would be the total deadline laxity, given as the sum of the positive time differences between the finish time of the task, on schedulable PEs, and its deadline. The other factor would be the number of PEs on which the task is schedulable, as governed by architectural constraints. Thus the multiplicative term in the DAM function takes the above two factors into consideration and gives the average deadline laxity that a task as at its disposal. The average deadline laxity of ni is given as:

$$\mathfrak{F}4\colon \frac{\sum_{P_j \in \mathfrak{D}} \left(D_{ni} - FT\left(n_i, P_j\right)\right)}{N} \tag{7.6}$$

Here, $\mathfrak{D}$ represents the set of PEs on which n_i has passed the deadline check and N the total number of PEs on which n_i is schedulable. Note that N is given by the architectural constraints of the BSN and is fixed for a task throughout the scheduling process.

7.1.3 Rescue operation(RO)

DA guarantees a schedule satisfying deadline and precedence constraints but not the endurance constraint. DA guarantees against battery-failures though event-failures can still occur in the schedule. A task suffering from event-failure is termed as *starving task*; starving communication is defined in the same manner. A PE can have a starving task as well as a starving communication occurring at the same time if the respective task and communication overlap along the time scale and the α_{SA} of the battery turns negative before either of them complete. In other words, the block formed by the overlapping task and communication has suffered an event-failure and is termed as *starving block*.

The purpose of RO would be to identify such starvations and employ recovery phenomenon of the battery to salvage them. RO introduces recovery periods in the schedule in an attempt towards rescuing them. A recovery period is nothing but an idle period inserted just before the start time of the starving event or block. In other words, starving event or block, along with their dependents, is pushed towards their deadline. The resulting idle period would allow a certain portion of α_L; the charge locked-up inside the battery, to be released and added to the available charge α_A. Thus, an event or block scheduled after a recovery period would have more charge at its disposal than if it were to be scheduled without the recovery period. The side effect of RO is elongation of the schedule length. Also note that any rest periods Δ that might have been inserted in the schedule by DA are taken as 'holes' into which events or blocks can be pushed. During RO the blocks produced by DA are pushed as a whole. The start times of all the events composing a block are postponed by the same amount. This implies that the relative schedule of events within a block never changes. This ensures that no new battery or event failure is introduced in the schedule during the RO process. As an example consider Fig. 7.2. Assume α_A of the battery turns negative at time 3.5s resulting in starving events T3 and C2. During RO, starving block B1; comprising of T2, T3, C1 and C3, would be pushed as a whole, rather than just events T3 and C2. Blocks/events on other PEs are also pushed to maintain precedence constraints.

RO terminates successfully once all starvations have been rescued. RO terminates with failure if a starvation cannot be rescued. This implies that the particular starving block or event cannot be rescued even after inserting the maximum possible recovery period subject to precedence and deadline constraints.

Chapter 8

Complexity Analysis of DA

We will first look into the complexity of a single iteration of DA, i.e. the complexity of scheduling one task, before giving the total complexity of DA.

Consider an instance of DA where $RL \neq \{\}$ and $R = |RL|$. Let partial schedule of P_j be denoted as PS_j and $|PS_j|$; the number of events scheduled in PS_j, be N_j. Let maximum in-degree in G be I. Let the number of PEs in the BSN be N_T. Each call to the battery model is $O(1)$.

Architectural constraint check for $n_j \in RL$ on a non-exhausted P_j is accomplished by a simple indexing operation into a table and is of $O(1)$. Deadline constraint is checked using equations (4.1) and (4.2). Determination of DAT depends on the number of parent tasks of n_j. Under worst case condition, I number of parents could be scheduled on distinct PEs other than P_j. This gives complexity for deadline check as $O(I)$. Thus, the two feasibility checks taken together have a complexity $O(RN_TI + RN_T)$. The next step computes DAM for each feasible assignment. Consider DAM computation for n_j on P_j. Assuming the number of schedulable PEs to be N_T, $\mathfrak{F}4$ and $\mathfrak{F}3$ have complexity $O(N_T)$.

$\mathfrak{F}_2$ is computed for each instance of data transfer and depends on the number of parents of n_j that are scheduled on distinct PEs other that P_j and its worst case complexity is given as $O(I)$. $\mathfrak{F}_1$ is $O(1)$. Thus, DAM computation has a total complexity of $O(RN_T(N_T + I))$. The next step is a sorting process that sorts RN_T elements and can be accomplished by means of a sorting algorithm of $O(RN_T log(RN_T))$ complexity.

Consider P_j with modified partial schedule. $\mathfrak{C}_1$ has worst case complexity when N_j events form a single block. The battery model is then called for $(1 + 2 + \ldots + N_j)$ times, which is equal to $(N_j^2 + N_j)/2$. Now, N_j is the number of scheduled events in PS_j and can always be represented as $N_j = x_j N + y_j E$ where N and E are the total number of tasks and edges in G respectively and $x_j > 0$ and $y_j \geq 0$ are some scalars. The battery model is similarly called for all PEs whose partial schedule have been modified. Thus, $\mathfrak{C}_1$ has a final worst case complexity of $O(N^2 + E^2 + NE)$. The $\mathfrak{C}_1$ loop can be executed for a maximum of RN_T times. That is, $(RN_T - 1)$ assignments fail $\mathfrak{C}_1$ leading to a new assignment and the final assignment passing $\mathfrak{C}_1$. Thus, the $\mathfrak{C}_1$ loop has complexity of $O(RN_T(N^2 + E^2 + NE)$.

$\mathfrak{C}_2$ needs to perform at most $(I + 1)$ comparisons to determine exhausted PEs. The comparison is done against the minimum α_U among all unscheduled tasks. Assuming that a sorted list of α_U of all tasks is available apriori and maintained throughout the scheduling process, it would take at most N steps to determine the least α_U among unscheduled tasks. Thus, complexity of $\mathfrak{C}_2$ can be given as $O(N + I)$.

$\mathfrak{C}_3$ is performed on P_j by calling the battery model N_j number of times. If P_j fails $\mathfrak{C}_3$, its FT is updated with a rest period Δ which can be obtained by N_j calls to the battery model. Proceeding as in the case of $\mathfrak{C}_1$, $\mathfrak{C}_3$ has a complexity of $O(N + E)$.

Combining the complexities obtained above and noting that R is bounded by N we obtain the complexity for a single iteration of DA as $O(NN_T(I+N_T+N^2+E^2+NE+log(NN_T)))$. DA being a list scheduler takes exactly N iterations to schedule a DAG consisting of N nodes. Thus,the total complexity of DA is $O(N^2N_T(I+N_T+N^2+E^2+NE+log(NN_T))) \approx O(N^4N_T)$.

Chapter 9

Simulation Results

We conducted rigorous simulation runs to study the performance of our proposed algorithm. Simulation runs were carried out on a $Pentium^{©} - D$, 2.8GHz, 1.5GB RAM machine running under $Fedora^{©}$ Linux natively. The programming language of choice was C. The application DAGs for simulation input were randomly generated using the graph generator tool TGFF[28]. The total number of nodes in each DAG was set to 100, with each node having a maximum in-degree and out-degree of 2 and 5 respectively. The DAGs could have multiple source nodes. A total of 100 such random DAGs were generated. The BSN under study is assumed to be composed of a maximum of 12 PEs. Heterogeneity in the BSN is captured by assigning each PE a type. A maximum of 3 different PE types are made available in our simulation environment and each PE in the BSN is randomly assigned a type. Tasks and communications produced by TGFF are also assigned different types. A total of 10 task and communication type is used.

The WCCC of a task type on a particular PE type was generated as a uniform distribution between 0mA and 3.774mA, with the mean at 1.887mA. The

WCET of a task type on a particular PE type was generated as a uniform distribution between 0ms and 3.2ms, with the mean at 1.6ms. The WCCC of a communication on a particular PE type was generated as a uniform distribution between 9.25mA and 27.75mA, with the mean at 18.5mA. The WCET of a communication type on a particular PE type was generated as a uniform distribution between 20ms and 180ms, with the mean at 100ms. Note that the WCCC for a communication does not depend on the communication type but only on the PE type. Random architectural constraints are also introduced at this stage.

Task deadlines were automatically generated by TGFF assuming average WCET of 200ms for communications and 1.6ms for tasks. 200ms would cater for a communication hop length of 2. The deadlines produced by TGFF are further divided by parameter UF; $0 < UF \leq 1.0$, which represents the utilization factor of the sensor network.

A total of 5 different battery types are made available with each type differing in α_I and β. The α_I of the battery types are generated as uniformly distributed values between 27mA-mins and 33mA-mins, the average value being at 30mA-mins. The α_I has been purposefully kept low to avoid extremely lengthy simulation durations. Consequently, 33mA-mins would correspond to a V_{BAT} of 3.6v and 30mA-mins to 2.7v. β values are generated uniformly between 0.273 and 0.637, with the mean at 0.5. A battery type is randomly assigned to each of the PEs in the BSN.

The performance metric that we are interested in is the lifetime($\mathfrak{S}$) of the BSN. In other words, the number of instances of a given application DAG that can be successfully run to completion on the BSN before it suffers either a *deadline failure* or a *starvation failure*. A deadline failure is said to have occurred if the scheduler is unable to allocate a ready task to any PE without violating

its deadline. A starvation failure is said to have occurred if any scheduled task suffers a starvation that cannot be rescued. The WCCC of events in the application DAG are updated at the end of each successfully scheduled instance. The α_A of the PEs is assumed to be equal to their respective α_T and the corresponding V_{BAT} is found. WCCC are then updated as explained in Section 5.2. The time period of the input DAG is assumed to be equal to its deadline. Δ is set as 50.8ms, which corresponds to the average of WCET of tasks and communications.

9.1 Lifetime vs. Number of PEs

We set UF as 0.1, HF as 0.2 and determined the lifetime with respect to different number of PEs. The result is shown in Fig. 9.1. All lifetime values are an average over 100 DAGs. Its obvious that with more PEs at its disposal the algorithm has more freedom during the scheduling process and thereby able to make better scheduling decisions. Also, any PE nearing depletion can easily be replaced by other PEs. Of course, it is expected that as the number of PEs increase the lifetime of the BSN increases. The average lifetime for a 12 PE BSN is 119.21 as compared to 73.49 for a 6PE BSN, which corresponds to an increase in lifetime of 62%. For all subsequent simulation runs the number of PEs is set as 6. The lifetime performance is also compared to two other most commonly used scheduling policies, namely, the popular EDF scheduler and Random scheduler. Our proposed algorithm XYZ being the first of its kind in addressing the scheduling problem with independent heterogeneous batteries, our choice of benchmarking is left with only against the above-mentioned algorithms. It is clear from Fig. 9.1 that our algorithm completely outperforms the two benchmarks across all PE inputs. Lifetime improvements in the range 1641%-3506% is observed over Random and 64%-464% over EDF.

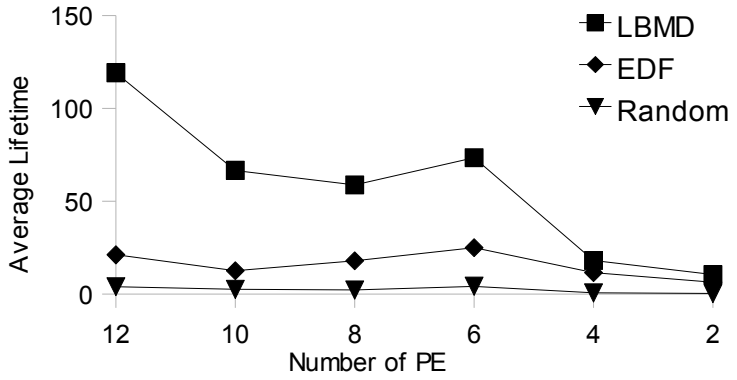

Figure 9.1: Lifetime Vs Number of PEs in BSN

9.2 Lifetime Vs Health Factor

The health factor HF dictates the amount of relaxation the batteries of the BSN enjoy. This consequently affects deadline as well as recovery performance of the algorithm. We now study the impact of HF with the help of two DAGs, namely, *dagA* and *dagB*. *dagA* and *dagB* were randomly picked form the set of 100 DAGs. We vary HF from 0.1 to 1.0, in increments of 0.1, and obtain lifetime results for values of UF equal to 0.1, 0.5 and 1.0. The results are shown in Fig. 9.2a and Fig. 9.2b.

The trend of the lifetime is found to be generally increasing as HF increases. However the increase in lifetime is predominant for $1.0 \geq HF \geq 0.5$ and tends to approach zero for $0.1 \leq HF < 0.5$. This indicates that lifetime improvements can be significant at higher values of HF and at lower values of HF the lifetime of the BSN tends towards a constant value. For example, consider *dagA* in Fig. 9.2a with $UF = 0.5$. At $HF = 0.7$ a lifetime of 6 was obtained and at $HF = 0.5$ a lifetime of 30 was obtained. This corresponds to an increase in lifetime of 400%. However, lifetime of only 32 was obtained for $HF = 0.1$ which corresponds to an increase of only 6.67% over the 0.5 case. This rather curious behavior can be attributed to the non-linear relaxation effect of the battery, which is shown

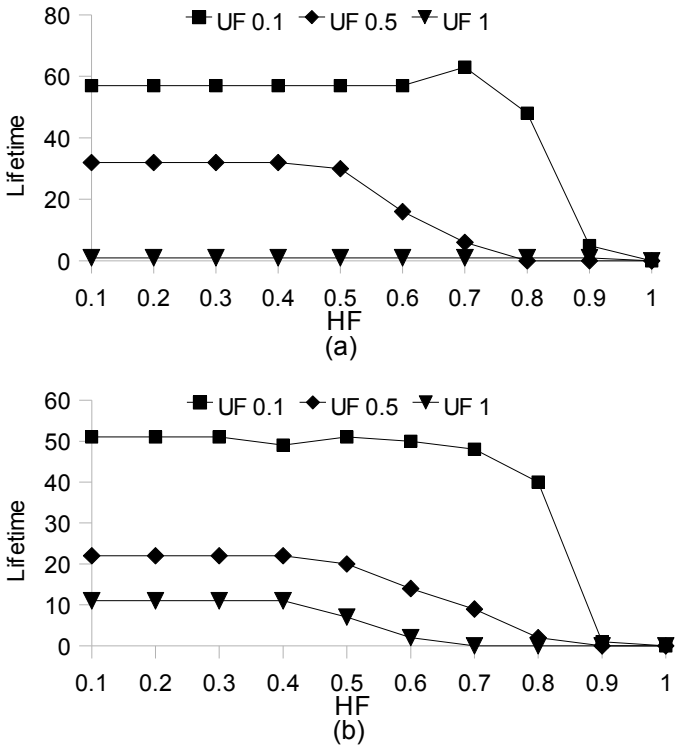

Figure 9.2: Lifetime Vs HF

in Fig. 9.3. The figure shows α_L of a battery with $\beta = 0.273$. It is clear from the figure that a large portion of α_L is released and added to the pool of α_A immediately after the external load is cutoff, which would result in a significant improvement in the health of the battery within a short time period after the load is cut off. This is the precise reason for the earlier observed behavior of the algorithm. Health improvement corresponding to lower values of HF can be obtained with almost identical rest periods. However health improvement corresponding to higher values of HF needs an increasing amount of rest period. Consequently, the lifetime is highly dependent on HF for $1.0 \geq HF \geq 0.5$.

Note that lifetime at $HF = 1.0$ is 0 for both *dagA* and *dagB*. Rest periods corresponding to this high value of HF would be very long in duration, consequently leading to deadline failures during the scheduling process.

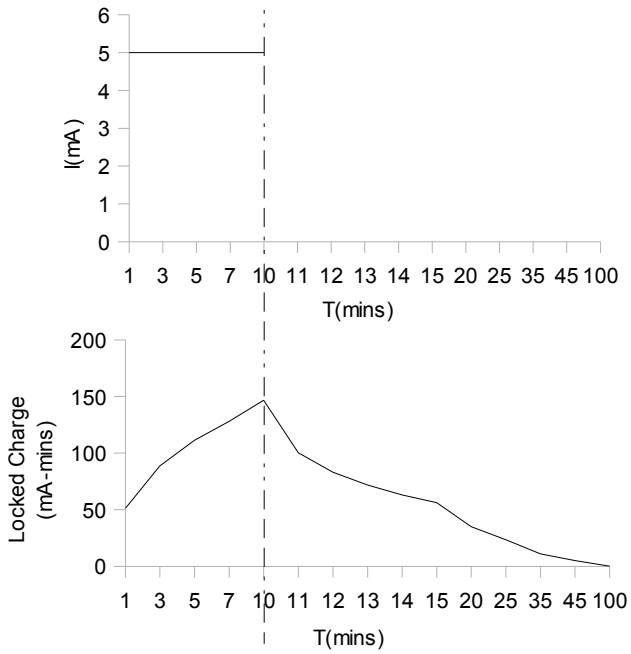

Figure 9.3: Lock-up and Recovery of Charge Inside a Battery

9.3 Lifetime Vs Utility Factor

We now study the effect of the utility factor on lifetime using *dagA* and *dagB*. We vary UF from 0.1 to 1.0, in increments of 0.1, and obtain lifetime results for values of HF equal to 0.1, 0.5 and 1.0. The results are shown in Fig. 9.4a and Fig. 9.4b. It is observed that as deadlines are further relaxed the lifetime increases. Its obvious that as deadlines are relaxed further the algorithm would have more freedom during the scheduling process and is thereby able to provide better schedules. Also, starving events can now be easily rescued since recovery period insertion will not have much adverse effect on deadline performance. Thus, it is not surprising that lifetime increases with relaxed deadlines. Once again lifetime at $HF = 1.0$ is 0.

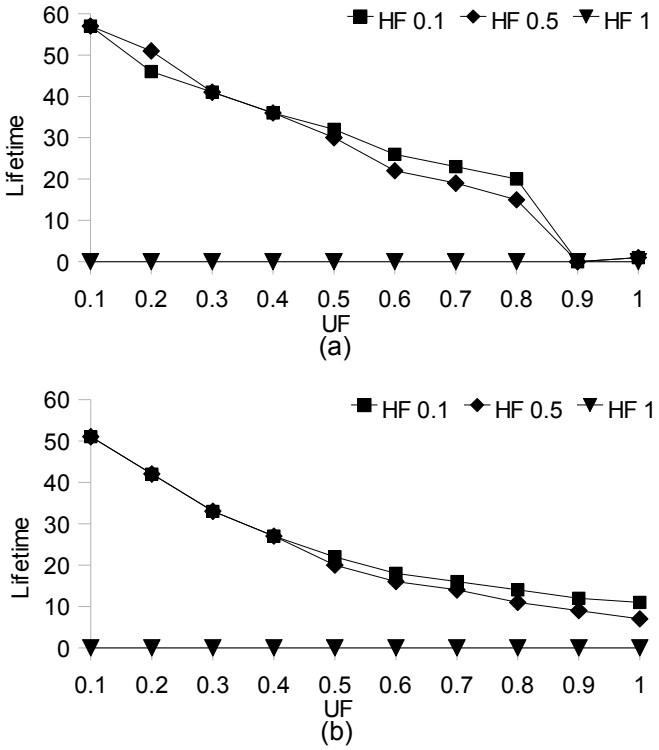

Figure 9.4: Lifetime Vs UF

9.4 Failure Mode

Deadline and starvation failure are the two main modes of failure our algorithm is susceptible to. Fig. 9.5 tabulates the failure mode that was obtained during simulation runs of Fig. 9.2 and Fig. 9.4Fig. Deadline failure is the predominant mode for higher values of HF and UF while starvation failure is the predominant mode for lower values of HF and UF.

Higher values of HF imply longer Δ rest periods in the schedule. Consequently, latter tasks might miss deadlines leading to deadline failures. Lower values of HF imply that the algorithm is free to allocate events to PEs that might not be in good health. Consequently the chances of an unrecoverable starvation increase. Higher values of UF imply tighter deadlines which lead to

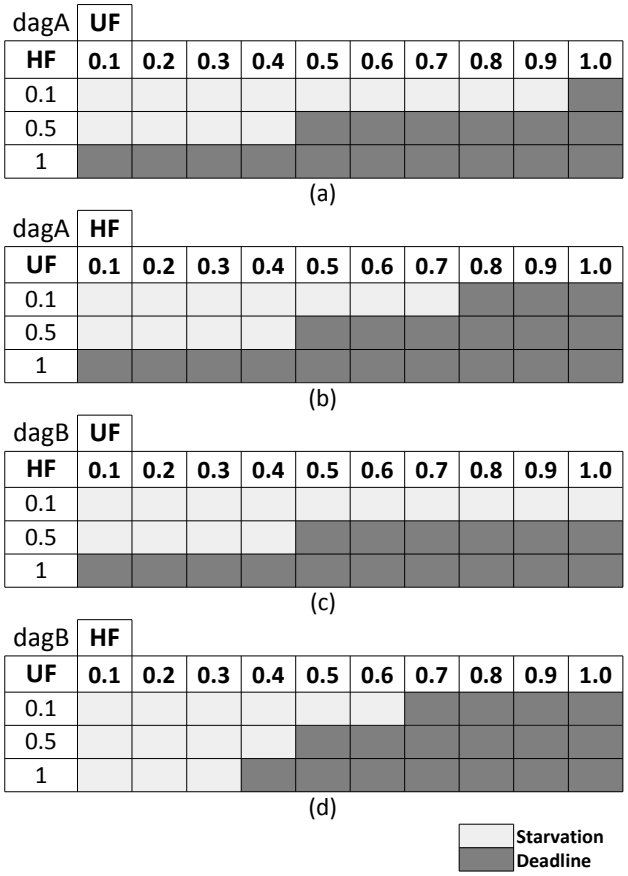

Figure 9.5: Failure Modes Across UF and HF

deadline failures. Note that when $UF = 1.0$ task deadlines are equal to their respective longest path from the source node(s). Lower values of UF relax deadlines thereby reducing deadline failures.

9.5 Uniform Battery Wear-off

Uniform wear-off of the batteries in the BSN is one of the objectives of our algorithm. α_T, representing the total charge inside a battery, is computed at time intervals equaling the time-period of a DAG and is used as a measure of the amount of wear-off a battery has undergone.

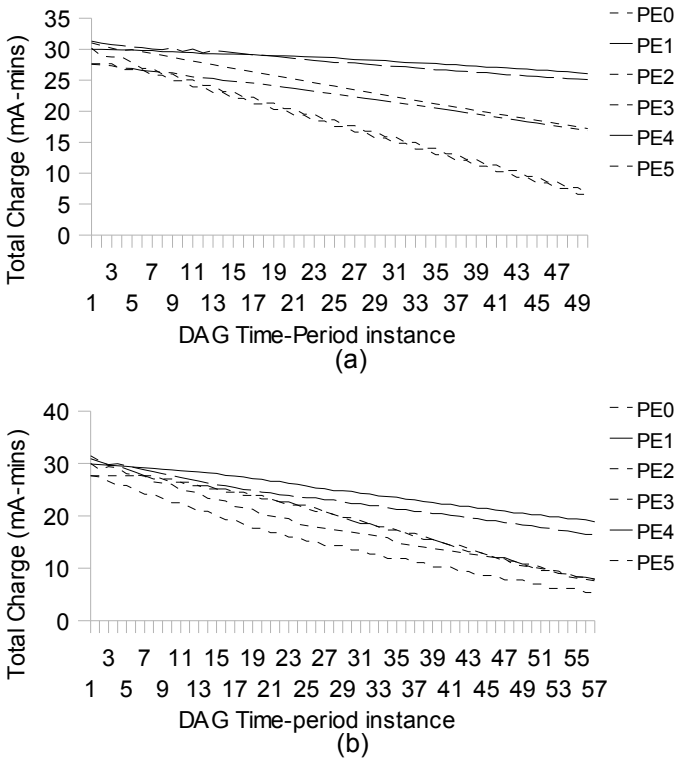

Figure 9.6: α_T showing uniform wear-off

Naturally α_T tends to decrease as the scheduling process progresses. Fig. 9.6a and Fig. 9.6b show the plot of α_T of all 6 PEs for *dagA* and *dagB* respectively. Its evident from the plots that the algorithm does try to achieve a uniform loading of all the batteries in the BSN. However, a perfect uniform wear-off might not be possible since the batteries are heterogeneous.

We will end this section by showing the recovery effect that our algorithm utilizes. Fig. 9.7 shows a snapshot of α_A of three PEs for time-period instances 1 to 10 during scheduling of *dagA*. The plot clearly shows recovery effect in action wherein the batteries get discharged in a 'pulsed' manner.

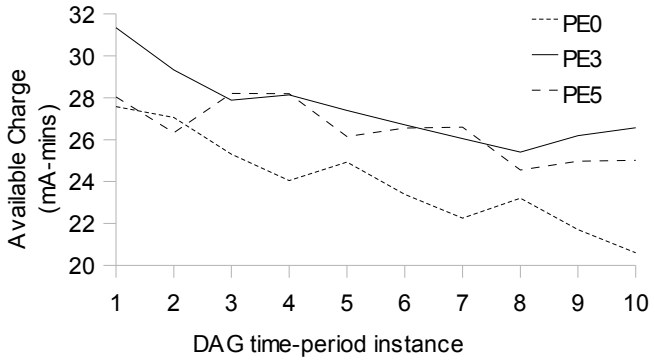

Figure 9.7: α_A showing recovery effect

Chapter 10

User Interface Development for LMBD

LMBD was simulated under DOS environment where all user interactions were via file input/output and console input/output. A need was thus felt to impart greater interactivity between the user and the simulation environment as well as greater intuition to the simulation process. A graphical front-end was then decided to be developed via which a user would be able to interact with the simulation environment. The graphical user interface (GUI) was envisioned to be feature rich; flexible enough to incorporate multiple schedulers, able to display schedule information to the user in a graphical manner, able to display application DAGs in a graphical manner, incorporate various tools for creation of application DAGs, accept all simulation parameters from within the graphical interface and also able to provide timely guidance to a novice user.

10.1 GST: Graphical Simulator Tool

A graphical tool called (unimaginatively) Graphical Simulator Tool (GST) was developed to address the shortcoming of the console based simulation environment. GST is a graphical front-end for simulation of multiple energy and QoS aware multi-processor scheduling algorithms. The tool serves as a single platform from which the user can launch, view and evaluate the results of the various scheduling algorithms. System input such as the number of processors in the hardware architecture as well algorithmic inputs can be entered via the tools graphical interface. Schedules produced by the algorithms can be viewed graphically from within the tool itself, thus enabling easier comparison of the various algorithms.

The tool is flexible enough to allow integration of algorithms that might be developed in the future. Various graphical widgets (for display purposes) are provided in form of function calls and can be used by the algorithm developer. Two types of DAG builders have been incorporated into the tool. A TGFF based builder and a wizard based builder. The former version accepts a DAG configuration file and utilizes the random graph generator TGFF to generate random application DAGs. The latter version allows building of a DAG from scratch, with the user specifying the exact structure and other characteristics of the application DAG. The resultant DAG of both versions can be viewed graphically and saved from within the tool itself.

The tool can be run natively under Linux or in Windows under Cygwin [29].

Figures 10.1-10.5 show some screen shots of GST in action.

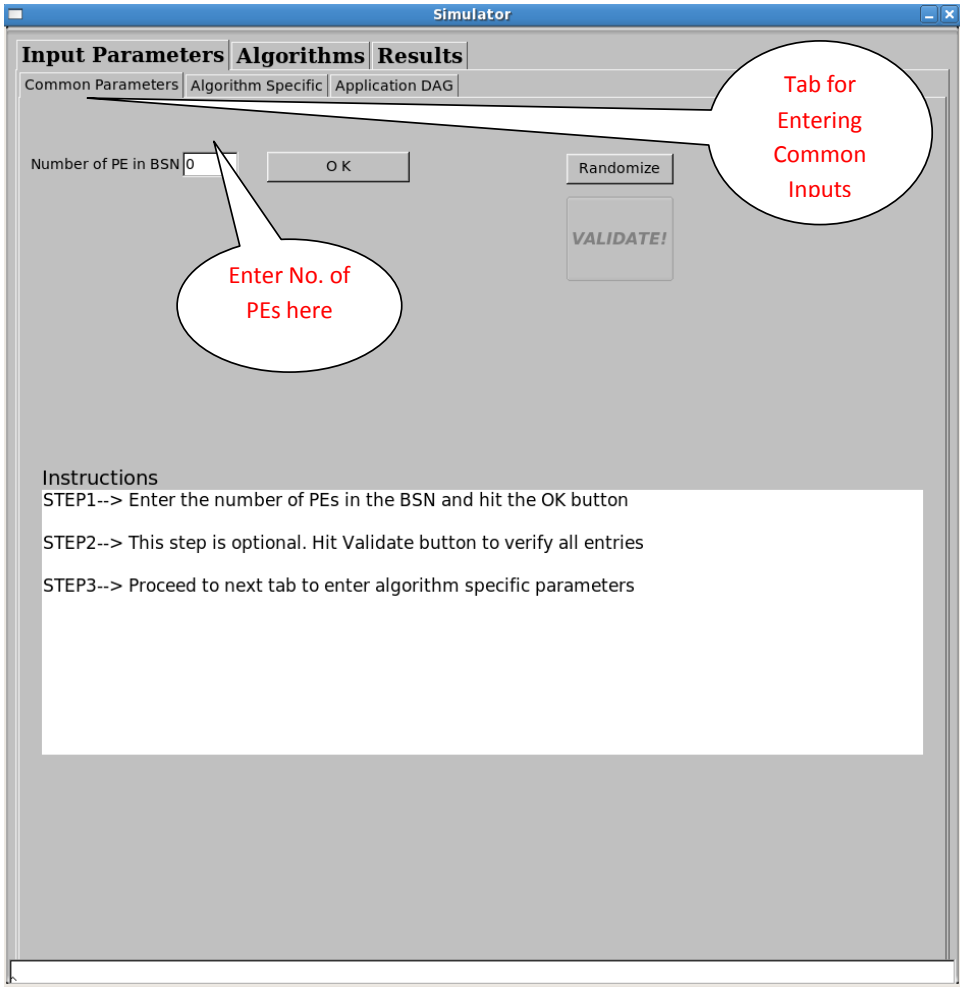

Figure 10.1: Starting screen of GST

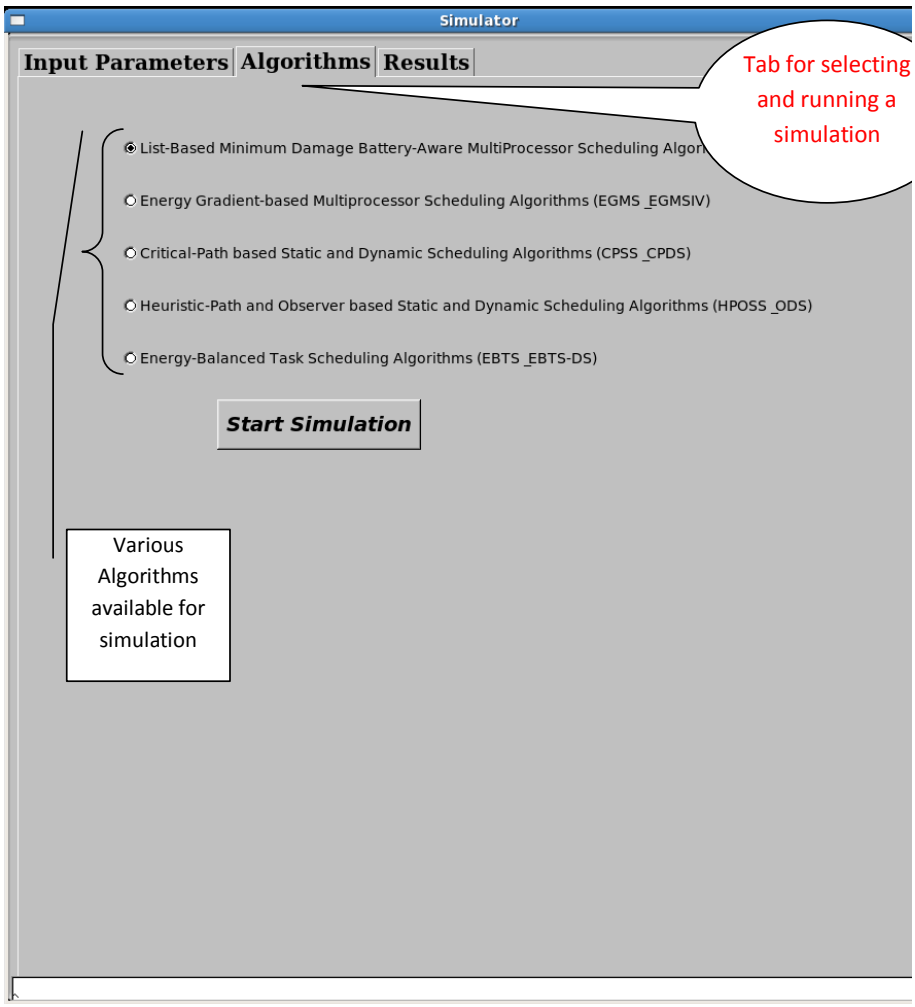

Figure 10.2: Option to run multiple algorithms

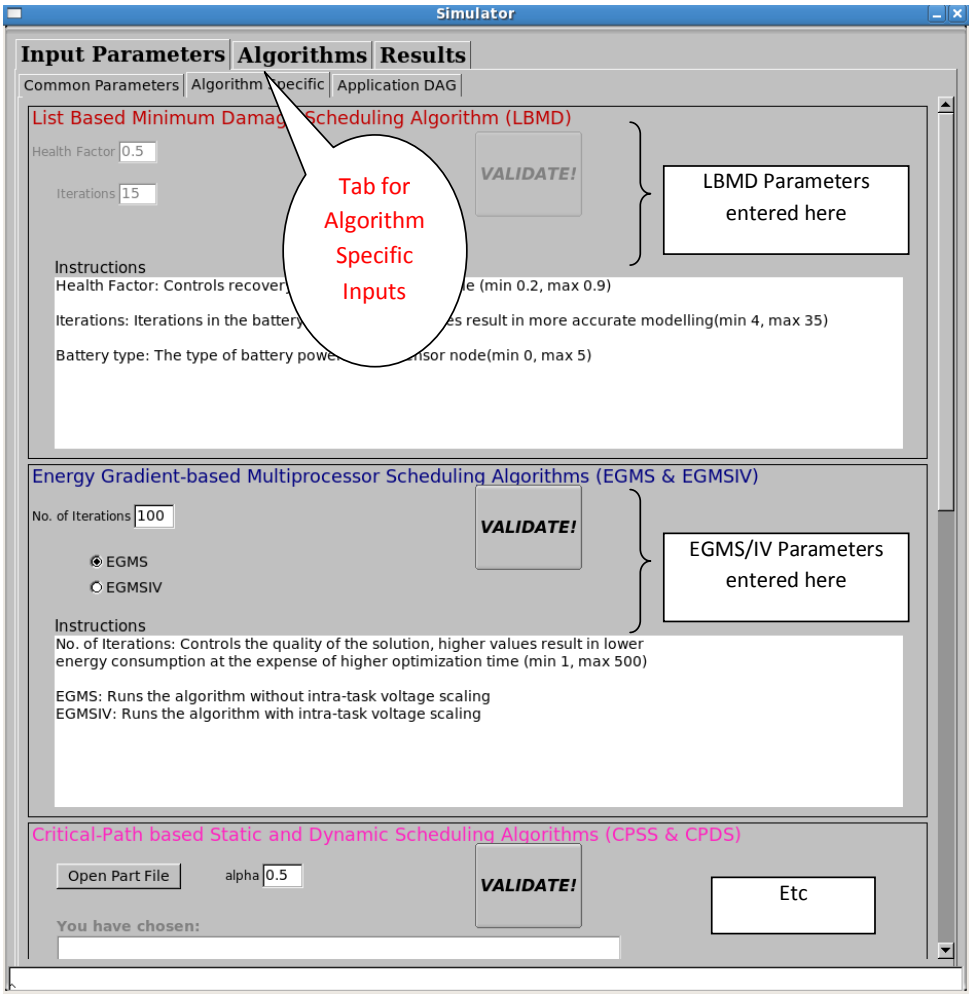

Figure 10.3: Entry page for simulation inputs

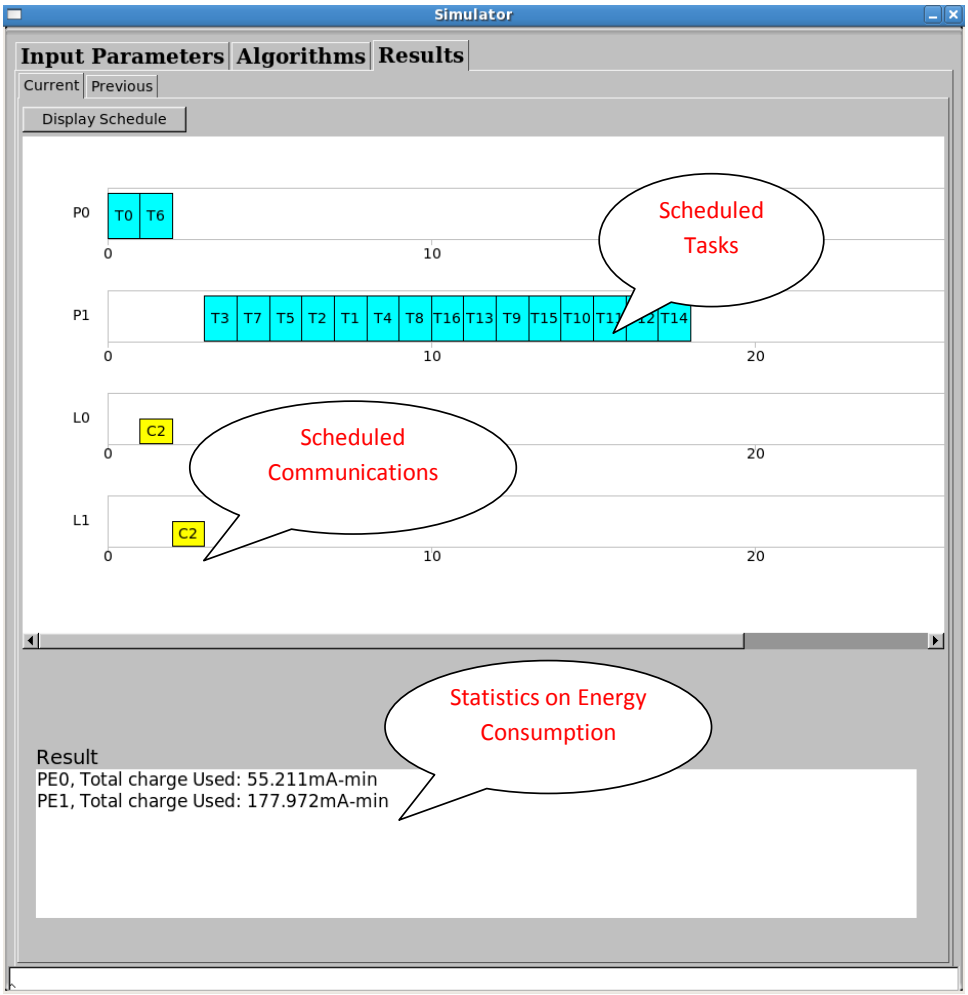

Figure 10.4: Graphical display of an output schedule

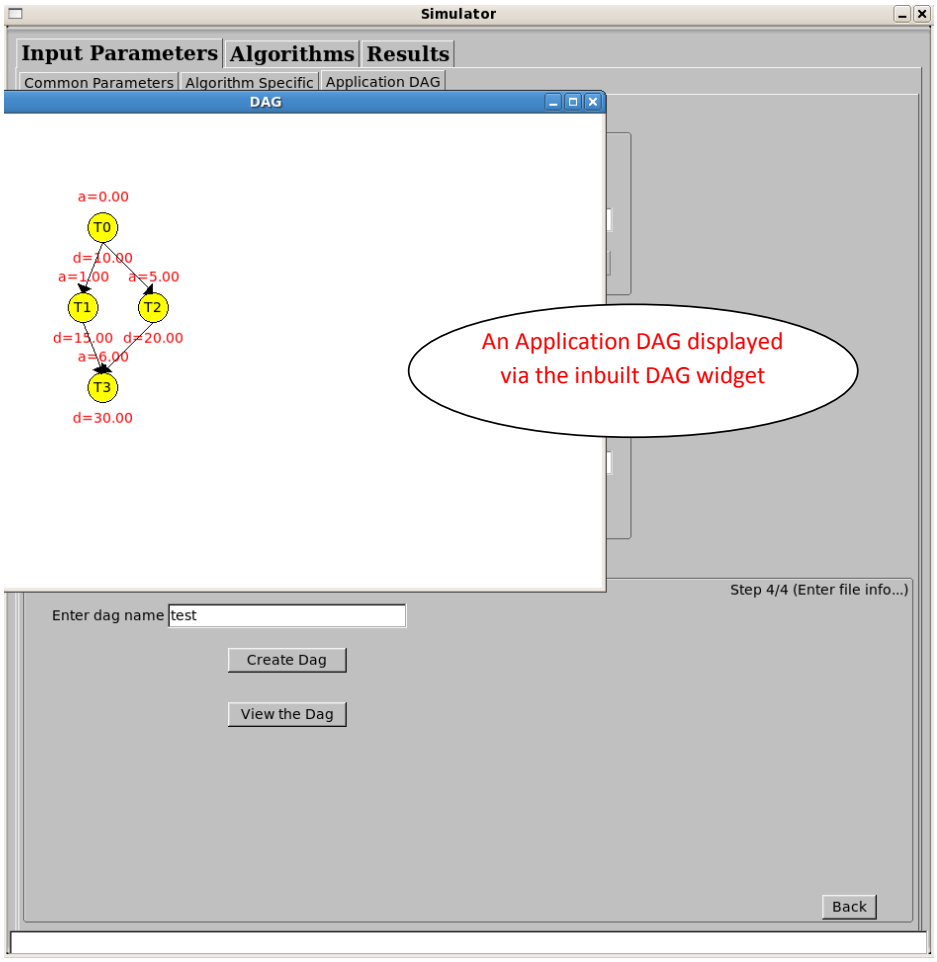

Figure 10.5: An application DAG created via GST

Chapter 11

Conclusion

The predominant energy source for mobile and wearable embedded systems, such as Body Sensor Networks(BSN), is the electro-chemical battery. Energy optimization techniques for such battery powered embedded systems need to devote special attention to the battery since a battery deviates significantly from an ideal energy source and exhibits several non-linear behaviors. Owing to this particular constraint, traditional energy saving techniques which predominantly assume the energy source to be ideal, are not suitable for battery powered systems.

In this dissertation we have developed a battery-aware scheduling algorithm known as *List-Based Minimum Damage(LMBD)* to schedule an application DAG, with deadline and non-negligible communication overheads, on to the processing elements of a BSN. The algorithm incorporates a list scheduler, with a heuristic *Damage* function serving as a decision maker. Non-ideal battery properties such as *rate-capacity* effect and *current-profiling* effect are exploited to maximize the amount of charge that can be extracted out of a battery. The *recovery* phenomenon of the battery is thoroughly exploited, particularly to rescue events

that might otherwise suffer a battery-failure. The proposed algorithm is unique in the context that it considers the processing elements of the BSN be individually powered by dedicated heterogeneous battery types; a first in literature.

This dissertation has also explored the feasibility of adopting unique power supply interfaces to eliminate losses in the power transmission systems. In particular, a previously proposed technique called Passive Voltage Scaling (PVS) is found to be a suitable choice to eliminate losses in DC-DC converters of the BSN. In PVS the DC-DC converter is eliminated by interfacing the processing element directly to the battery and operating the processing element at a lower operational frequency.

In addition, this dissertation has also contributed to the development of a graphical application called Graphical Simulator Tool (GST) which serves as a graphical front-end for algorithm simulation environments. Various scheduling algorithms can be incorporated into GST by an user. The user can input algorithmic parameters, launch a simulation and view the results in graphical format from within GST. GST also incorporates various DAG building tools.

11.1 Summary of Research Contributions

To summarize, the following are the contributions of this dissertation:

1. A battery-aware multiprocessor scheduling algorithm called *List-Based Minimum Damage(LMBD)* has been developed to schedule an application DAG on to the distributed sensor nodes of a Body Sensor Network. The algorithm is the first in literature to consider distributed and heterogeneous battery sources for the underlying multiprocessor network.

2. Techniques to eliminate DC-DC converter losses have been explored and a power supply setup called Passive Voltage Scaling is found to be suitable for the sensor nodes under study. LMBD operates under this setup.

3. A graphical application called Graphical Simulator Tool has been developed via which an user can select, launch, view and compare results of scheduling algorithms in an intuitive graphical manner.

11.2 Future Work

The work in this dissertation has plenty of scope for further development. We now briefly introduce possible future research directions.

- LBMD is a static algorithm that generates schedules offline. An immediate and interesting extension of this work would be to design a dynamic version of LBMD.

- The time complexity of LMBD is of order 4. Though being polynomial, this can be regarded as quite high. Effort can be expended in future work to mitigate the high complexity of LMBD.

- Performance of LMBD was evaluated via simulation runs using synthetically generated application DAGs. An interesting extension to this work would be to validate the performance of the algorithm on real-world BSN applications.

- In this work, microcontrollers and the RF modules are considered to be the predominant energy sinks in a sensor network. Though this assumption has been shown to be true, it would nevertheless be interesting to study the impact of other system components such as memory, A-D converters, transducers etc. on battery performance.

Bibliography

[1] M. N. Nyan, F. E. H. Tay, D. Guo, L. Xu, K. L. Yap, L. K. Goh, and B. Veeravalli, "Low-energy Scheduling Algorithms for Wearable Fall Pre-impact Detection System," in *WMSCI 2008: 12TH WORLD MULTI-CONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, VOL II, PROCEEDINGS*, Callaos, N and Lesso, W and Zinn, CD and Baralt, J and Eshraghian, K and Severi, S, Ed., 2008.

[2] Website. [Online]. Available: http://www.itu.int/ITU-D/icteye/Indicators/Indicators.aspx#

[3] T. Starner, "The challenges of wearable computing: Part 1," *IEEE Micro*, vol. 21, no. 4, pp. 44–52, July 2001. [Online]. Available: http://dx.doi.org/10.1109/40.946681

[4] ——, "The challenges of wearable computing: Part 2," *IEEE Micro*, vol. 21, no. 4, pp. 54–67, July 2001. [Online]. Available: http://dx.doi.org/10.1109/40.946683

[5] G.-Z. Yang, Ed., *Body Sensor Networks*. Springer, 2006.

[6] Y. Cho, Y. Kim, and N. Chang, "Pvs: passive voltage scaling for wireless sensor networks," in *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 2007, pp. 135–140. [Online]. Available: http://dx.doi.org/10.1145/1283780.1283810

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. [Online]. Available: http://portal.acm.org/citation.cfm?id=574848

[8] H. E. Rewini, T. G. Lewis, and H. H. Ali, *Task scheduling in parallel and distributed systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. [Online]. Available: http://portal.acm.org/citation.cfm?id=174596

[9] D. Rakhmatov and S. Vrudhula, "Energy management for battery-powered embedded systems," *Trans. on Embedded Computing Sys.*, vol. 2, no. 3, pp. 277–324, 2003. [Online]. Available: http://dx.doi.org/10.1145/860176.860179

[10] L. Benini, G. Castelli, E. Macii, M. Poncino, and R. Scarsi, "A discrete-time battery model for high-level power estimation," in *In Proceedings of Design, Automation and Test in Europe*, 2000, pp. 35–39. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.4155

[11] M. Doyle and J. Newman, "Modeling the performance of rechargeable lithium-based cells : design correlations for limiting cases," *Journal of Power Sources*, pp. 46–51, March 1995. [Online]. Available: http://cat.inist.fr/?aModele=afficheN&cpsidt=3522163

[12] S. Gold, "A pspice macromodel for lithium-ion batteries," in *Battery Conference on Applications and Advances, 1997., Twelfth Annual*, 1997, pp. 215–222. [Online]. Available: http://dx.doi.org/10.1109/BCAA.1997.574106

[13] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, X. Raghunathan, and K. Lahiri, "Battery life estimation of mobile embedded systems," 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.8461

[14] E. H. Callaway, *Wireless Sensor Networks: Architecture and Protocols*. Auerbach Publications, 2004.

[15] Y. Choi, N. Chang, and T. Kim, "Dc-dc converter-aware power management for battery-operated embedded systems," in *DAC '05: Proceedings of the 42nd annual conference on Design automation*. New York, NY, USA: ACM, 2005, pp. 895–900. [Online]. Available: http://dx.doi.org/10.1145/1065579.1065814

[16] P. Chowdhury and C. Chakrabarti, "Static task-scheduling algorithms for battery-powered dvs systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 2, pp. 226–237, 2005. [Online]. Available: http://dx.doi.org/10.1109/TVLSI.2004.840771

[17] Y. Cai, S. M. Reddy, I. Pomeranz, and B. M. Al-Hashimi, "Battery-aware dynamic voltage scaling in multiprocessor embedded system," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, 2005, pp. 616–619 Vol. 1. [Online]. Available: http://dx.doi.org/10.1109/ISCAS.2005.1464663

[18] J. Ahmed and C. Chakrabarti, "A dynamic task scheduling algorithm for battery powered dvs systems," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 2, 2004, pp. II–813–16 Vol.2. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1329396

[19] J. Khan and R. Vemuri, "Battery-efficient task execution on reconfigurable computing platforms with multiple processing units," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and*

Distributed Processing Symposium (IPDPS'05) - Workshop 3. Washington, DC, USA: IEEE Computer Society, 2005. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2005.122

[20] P. Rong and M. Pedram, "Battery-aware power management based on markovian decision processes," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, 2002, pp. 707–713. [Online]. Available: http://dx.doi.org/10.1109/ICCAD.2002.1167609

[21] J. Luo and N. K. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 444–449. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=935550

[22] M. Pedram and Q. Wu, "Design considerations for battery-powered electronics," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM, 1999, pp. 861–866. [Online]. Available: http://dx.doi.org/10.1145/309847.310089

[23] T. L. Martin, "Balancing batteries, power, and performance: system issues in cpu speed-setting for mobile computing," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=930660

[24] *Datasheet of MSP430 Family Ultra Low Power Microcontrollers*, Texas Instruments Incorporated.

[25] C. Park, K. Lahiri, and A. Raghunathan, "Battery discharge characteristics of wireless sensor nodes: an experimental analysis," in *Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005. 2005 Second Annual IEEE Communications Society Conference on*, 2005, pp. 430–440. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1557096

[26] *Datasheet of SmartRF CC2420*, Texas Instruments Incorporated.

[27] D. Rakhmatov, S. Vrudhula, and C. Chakrabarti, "Battery-conscious task sequencing for portable devices including voltage/clock scaling," in *Design Automation Conference, 2002. Proceedings. 39th*, 2002, pp. 189–194. [Online]. Available: http://dx.doi.org/10.1109/DAC.2002.1012618

[28] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, 1998, pp. 97–101. [Online]. Available: http://dx.doi.org/10.1109/HSC.1998.666245

[29] Website. [Online]. Available: http://www.cygwin.com/

[30] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *MobiCom '01: Proceedings of the*

7th annual international conference on Mobile computing and networking. New York, NY, USA: ACM, 2001, pp. 251–259. [Online]. Available: http://dx.doi.org/10.1145/381677.381701

[31] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*. New York, NY, USA: ACM, 1998, pp. 197–202. [Online]. Available: http://dx.doi.org/10.1145/280756.280894

Appendix A

Lemmas

Lemma1: $\mathfrak{C}\mathfrak{1}$ prevents battery-failures. In other words, for a PE, the condition $\alpha_A + \alpha_L \geq 0$ is true at all times during the scheduling process.

Proof: Consider P_j with a partial schedule. For the partial schedule to have been successfully scheduled it must have passed $\mathfrak{C}\mathfrak{1}$. Let E be an event with execution time Δ_E and start time t. Let the charge used by E be α_U and finish time of E be $\acute{t} = t + \Delta_E$. $\mathfrak{C}\mathfrak{1}$ for E is given as:

$$\alpha_A + \alpha_L \geq \alpha_U$$

$$\Rightarrow \alpha_A + \alpha_L \geq \alpha_C + X \tag{A-1}$$

Here $(\alpha_A + \alpha_L)$ gives the net charge available from the battery of P_j at t. α_C is the charge consumed by event E and X is the charge E will lock inside the battery. At $\acute{t}$, the net charge in the battery would be:

$$\alpha_A + \alpha_L - \alpha_C \tag{A-2}$$

From (A-1) and (A-2) it can be seen that $\alpha_A + \alpha_L \geq 0$ at time $\acute{t}$.

Let B be a block with execution time Δ_B, start time t and finish time $\acute{t} = t + \Delta_B$. $\mathfrak{C}\mathfrak{1}$ for block B assigns α_T to α_A at time t. This implies that the start time of B is postponed by a finite amount of idle period. Let this idle period be of duration Δ_I. Now, $\mathfrak{C}\mathfrak{1}$ ensures that $\alpha_A \geq 0$ in the time interval $(t + \Delta_I)$ to $(t' + \Delta_I)$.

$$\Rightarrow \alpha_A + \alpha_L \geq 0 \text{ from } (t + \Delta_I) \text{ to } (t' + \Delta_I) \tag{A-3}$$

That is, assuming insertion of some finite idle period at time t, the net charge in the battery is greater than or equal to zero at all times in B. But, as a property of the battery, the net charge does not depend on any idle periods. This implies that (A-3) is satisfied when start time of B is not postponed by any idle period. Thus (A-3) is true in the time interval t to t' in the partial schedule.

From the above given arguments it is clear that $\alpha_A + \alpha_L \geq 0$ at all times during the scheduling process.

Lemma2: For a PE, the condition $\alpha_{SA} > 0$ is true at all times during the scheduling process.

Proof: Let there exist a partial schedule on P_j till time $\acute{t}$. For the partial schedule to be valid, P_j should have passed $\mathfrak{C}\mathfrak{3}$ at time $\acute{t}$. Which implies at time $\acute{t}$:

$$\alpha_{SA}/\alpha_T \geq HF \tag{A-4}$$

$$\text{Now, } \alpha_{SA} = \alpha_A + SF \cdot \alpha_L \tag{A-5}$$

$$\text{and } \alpha_T = \alpha_A + \alpha_L \tag{A-6}$$

Since, HF is a positive quantity, it follows from (A-4), (A-6) and Lemma1 that

$\alpha_{SA} > 0$ at $\acute{t}$. In the very first iteration of DA $\alpha_{SA} = \alpha_A$. Thus $\alpha_{SA} > 0$ at all times during the scheduling process.

Lemma3: Any rest period Δ to be inserted at the end of a valid partial schedule is always finite.

Proof: The health check at the end of a valid partial schedule is given as $\alpha_{SA}/\alpha_T \geq HF$. Expanding gives:

$$\frac{\alpha_A + SF \cdot \alpha_L}{\alpha_A + \alpha_L} \geq HF \tag{A-7}$$

The locked charge α_L in a battery gets released due to recovery effects and adds to the available charge α_A, thus:

$$\lim_{time \to \infty} \alpha_L = 0 \tag{A-8}$$

From (A-8) it can be seen that (A-7) will be true for some finite value of Δ since $0 < HF < 1.0$.

Appendix B

Impact of HF and SF on Δ

Rewriting and simplifying (A-7) yields:

$$\alpha_A \geq \frac{\alpha_L \left(HF - SF\right)}{1 - HF} \tag{B-1}$$

The denominator of (B-1) and α_L are positive. When,

1. $HF > SF$

    - If $\alpha_A \leq 0$, then a rest period is definitely required at the end of the partial schedule.
    - If $\alpha_A > 0$, then a rest period may or may not be required. Condition (B-1) needs to be explicitly checked to determine whether a rest period is required.

2. $HF < SF$

    - If $\alpha_A \geq 0$, then a rest period is definitely not required at the end of the partial schedule.

- If $\alpha_A < 0$, then a rest period may or may not be required. Condition (B-1) needs to be explicitly checked to determine whether a rest period is required.

3. $HF = SF$

   - If $\alpha_A \geq 0$, then a rest period is definitely not required at the end of the partial schedule.
   - If $\alpha_A < 0$, then a rest period is definitely required at the end of the partial schedule.

Under the scenario that a rest period Δ is required, (B-1) can be used to arrive at the value of α_A at the end of Δ. This α_A can then be used in the battery model to determine the value of Δ.

Appendix C

Dynamic Voltage Scaling (DVS)

Digital CMOS circuits are susceptible to two kinds of energy loss, namely, static and dynamic. Static power losses are negligible, usually in the order of $<< 1mW$ and are ignored while computing the energy efficiency of a CMOS circuit [30]. Dynamic power loss can be represented by means of a simple equation as given below:

$$P_d = C \cdot f \cdot V_{dd}^2 \tag{C-1}$$

where P_d represents the dynamic power, C the total capacitance of the circuit, f the operating frequency and V_{dd} the supply voltage. It is obvious from equation (C-1) that reducing V_{dd} would result in quadratic savings in P_d. However, a reduction in the supply voltage of the CMOS circuit is accompanied by an increase in circuit delay. The circuit delay of a CMOS circuit is given as:

$$T_d = \frac{V_{dd}}{(V_g - V_t)^2} \tag{C-2}$$

where T_d represents the circuit delay, V_{dd} the supply voltage, V_g the input gate voltage and V_t the threshold voltage. The CMOS circuit now needs to be operated at a lower frequency to compensate for the increased delay. Consequently the execution time for a given task increases.

It can be seen from equations (C-1) and (C-2) that there is a trade-off between dynamic power savings and increased execution time. DVS enabled processors are able to dynamically variable their supply voltage and operating frequency during runtime under software control. Algorithmic techniques that rely on DVS vary the operating voltage and consequently the operating frequency of a processor in an intelligent manner to trade execution time for savings in energy consumption. For further insights into DVS the reader is referred to [31].