To appear in the ACM Transactions on Sensor Networks

Energy-driven Distribution of Signal Processing Applications across Wireless Sensor Networks

CHUNG-CHING SHEN and WILLIAM L. PLISHKER
University of Maryland, College Park
DONG-IK KO
Texas Instruments
and
SHUVRA S. BHATTACHARYYA and NEIL GOLDSMAN
University of Maryland, College Park

Wireless sensor network (WSN) applications have been studied extensively in recent years. Such applications involve resource-limited embedded sensor nodes that have small size and low power requirements. Based on the need for extended network lifetimes in WSNs in terms of energy use, the energy efficiency of computation and communication operations in the sensor nodes becomes critical. Digital signal processing (DSP) applications typically require intensive data processing operations and as a result are difficult to implement directly in resource-limited WSNs. In this paper, we present a novel design methodology for modeling and implementing computationally-intensive DSP applications applied to wireless sensor networks. This methodology explores efficient modeling techniques for DSP applications, including data sensing and processing; derives formulations of energy-driven partitioning (EDP) for distributing such applications across wireless sensor networks; and develops efficient heuristic algorithms for finding partitioning results that maximize the network lifetime. To address such an energy-driven partitioning problem, this paper provides a new way of aggregating data and reducing communication traffic among nodes based on application analysis. By considering low data token delivery points and the distribution of computation in the application, our approach finds energy-efficient trade-offs between data communication and computation.

Categories and Subject Descriptors: C.2 [**Computer-Communication Networks**]: Network Architecture and Design; C.3 [**Special-purpose and Application-based Systems**]: Signal Processing systems

General Terms: Algorithms, Design, and Experimentation

Additional Key Words and Phrases: Wireless sensor networks, DSP, energy efficiency, network lifetime, and speech recognition

This work was supported in part by the Laboratory for Physical Sciences.
Author's address: C. Shen, W. Plishker, S. S. Bhattacharyya, N. Goldsman, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742, USA; e-mail: ccshen@umd.edu, plishker@umd.edu, ssb@umd.edu, neil@umd.edu; D. Ko, Texas Instruments, Dallas, USA; e-mail: dik777@gmail.com.

1. INTRODUCTION

A wireless sensor network (WSN) system is composed of a collection of sensor nodes, where each node contains components for sensing, data processing, and communication. Traditionally, sensor nodes are limited in size, power, and memory, and perform relatively light-weight computational tasks. Also, sensor nodes are often deployed very densely, or in remote, inaccessible, or dangerous areas, which makes replacement of their batteries costly or infeasible. Therefore, WSN systems are typically designed with low power consumption, and energy-constrained lifetime maximization as primary objectives. Energy-constrained WSN applications include habitat monitoring, environmental observation, and battlefield surveillance (e.g., see [Akyildiz et al. 2002], [Kuorilehto et al. 2005]).

In a sensor network, as we increase the number of sensor nodes, requirements on network lifetime, and volume of data traffic across the network, it is often efficient to move towards hierarchical network architectures (e.g., see [Kumar et al. 2003]). In a hierarchical WSN, sensor nodes are clustered into groups, and the nodes within each clustered group are divided into the "master" (e.g., the cluster head) and "slave" nodes for more efficient structuring of network traffic. The master node is typically fully-featured — that is, the platform is equipped with a relatively high-performance processor and transceiver, larger size memory storage, and sizable energy source. Conversely, slave nodes are lean in terms of features — the platforms are equipped with simple processors (e.g., small microcontrollers), simple transceivers, sensors, limited memory storage, and relatively small energy resources. Thus, slave nodes are typically equipped to carry out simple computations and transmit only the required processed data to the associated master nodes for more computationally-intensive tasks.

In this paper, we target this kind of hierarchical WSN organization. We assume that after a self-organizing, cluster/network startup period, every slave node is able to communicate directly with its associated master node, and the size of each clustered group is in the range up to tens of nodes. Here, we define the network size as the number of active nodes in a system. This type of network structure is similar to so-called infrastructure-based networks [Romer and Mattern 2004]. As a practical example, a tire pressure monitoring system was designed and demonstrated in [Zhang et al. 2008] that is based on such a master-slave topology with the use of an energy-related TDMA technology.

We assume that a fixed amount of processing must be performed within each cluster so that minimal information needs to be communicated globally (across clusters) within the overall network, and our objective is to maximize the energy efficiency of this pre-defined amount of cluster-level processing. Such static configurations of network structure have simple routing characteristics, and protocol demands, and provide useful opportunities to optimize the network lifetime in terms of energy consumption, as well as network scalability through hierarchical organization. Whenever the network size changes (e.g., nodes enter or exit the system), the workload distribution corresponding to the amount of processing within each cluster should be efficiently adapted in terms of the new network structure. This workload redistribution scheme, which will be demonstrated in Section 5.4, can provide further improvements in network lifetime under dynamic network topologies.

Data collection and management play an important role in WSNs, especially for the applications, such as digital signal processing (DSP) applications, that need to sense and process large amounts of data. For example, a distributed automatic speech recognition (DASR) system [Shen et al. 2008] which is a DSP-based WSN application senses and processes large amounts of data, and communicates required information across WSN. The DASR system can be applied to application scenarios. The first scenario involves using a DASR system as a speech-based speaker-dependent command and control system in a battlefield environment. When the system is applied in a battlefield for recognizing command words, the speaker-dependent property provides a benefit by rejecting command words that are spoken by unauthorized people. The other application scenario is to use a DASR system as a surveillance system for collecting large amounts of speech data with similar patterns from arbitrary speakers. Since sensor nodes are usually designed to be small, they can be hidden in a battlefield environment that is being monitoring. Therefore, acoustic signals from the enemy can be secretly sensed, collected, and translated into useful data on the sensor nodes. Through a well-designed communication protocol, this data can then be transmitted to a central node for further processing and backend analysis. The recognized words, for example, can be used to distinguish among a diversity of languages or to survey specific words in a crowd for special monitoring and detection purposes.

Based on a master-slave network topology, with advances in integrated circuit technology, slave nodes can be equipped with increasing amounts of computational resources, such as digital signal processor subsystems, and other kinds of subsystems for performing some amount of pre-processing on sensed data before transmitting it. For example, in [Calhoun et al. 1995], the authors propose a sensor node architecture that contains a digital signal processor for executing signal processing programs. In such kinds of platforms, microcontrollers can perform protocol and control tasks, while the digital signal processor performs more intensive computational tasks. For many applications, doing some processing on the slave nodes reduces the amount of data that needs to be communicated across the network. Our work in this paper is motivated by this important potential benefit of pre-processing, and develops a systematic methodology apply and optimize slave node pre-processing in master/slave WSNs.

The energy consumption of the sensor nodes in a wireless sensor network, including the energy consumption for computation- and communication-related tasks, must be carefully optimized to increase network lifetime. In general, the communication tasks carried out by the transceiver dominate the overall power consumption on a sensor node (e.g., see [Ganesan et al. 2004], [Tang and Xu 2008]). For example, Table I shows a comparison of power consumption among microcontrollers and radio modules in various off-the-shelf platforms. The column labeled "MCU" represents the power consumed by the microcontroller module when it is running at full speed. TX (RX) denotes the power consumed by the radio module in transmit (receive) mode.

To reduce the energy consumed by the transceivers on the sensor nodes, the amount of data to be communicated across the wireless channel should be minimized. When distributing DSP-oriented computations across a WSN, this consider-

Table I. Power consumption comparison among various off-the-shelf platforms.

Product name	MCU	TX (0 dBm)	RX
Texas Instruments CC2430	20.2mW	74.1mW	81mW
Ember Em250	25.5mW	72mW	84 mW
Atmel ATmega64RZAV	14.4mW	49.5mW	46.5mW
Crossbow MICAz	24mW	52.2mW	59.1mW

ation is significant since DSP applications usually process large amounts of data in an iterative fashion. We carefully consider this problem of workload distribution, including the costs associated with computation and communication, so that we can maximize energy efficiency for WSN applications. We define this problem as the energy-driven partitioning (EDP) problem.

In this paper, we develop a precise formulation of the EDP problem for signal processing applications that are modeled as dataflow graphs, and we present an effective algorithm to address the problem. Specifically, our algorithm finds an efficient trade-off between the workload distribution of computation and communication tasks. The technique that we develop in this paper is novel in that it analyzes the pattern of internal data exchange rates within an application to minimize the overall energy consumption of a sensor network system, while also taking into account latency and/or real-time constraints based on application requirements. The approach is especially well-suited for multirate signal processing applications, which exhibit complex and nonuniform patterns of data exchange across functional modules and subsystems.

The rest of paper is organized as follows. Section 2 discusses the related work. Section 3 provides relevant background on dataflow modeling for DSP applications. Section 4 introduces the general idea for modeling workload distributions based on the given dataflow graphs. Section 5 shows our novel methodology of energy-driven partitioning, which includes energy formulations, cost and constraints, and dynamic topology management. Section 6 analyzes the NP-completeness of the addressed problem for energy-driven partitioning and proposes a heuristic algorithm to solve the EDP problem. Experiments including case studies are discussed in Section 7. Experimental setup and simulation results are shown in Section 8. Finally, Section 9 concludes the paper and discusses useful directions for future work.

2. RELATED WORK

Many researchers have suggested a hierarchical, physical-layer driven sensor network design to reduce data traffic and energy consumption of sensor nodes in connection with the physical-layer network functions (e.g., see [Lindsey et al. 2002]). To manage network lifetime, design of distributed medium access control protocols based on integrating physical layer parameters has been discussed in [Chen and Zhao 2007]. At the network layer, different energy-efficient routing schemes have also been discussed. For example, in [Park and Sahni 2006], the authors present an on-line heuristic routing scheme to maximize network lifetime. This scheme is based on the number of messages successfully routed before the first failed message. In these kinds of approaches, supporting significant levels of computational complexity for sensor node processing tasks is not a major concern. Instead, em-

phasis is placed on node optimization for conserving energy in conjunction with underlying protocol characteristics. Compared to these more protocol-oriented approaches, our approach for energy-driven workload distribution is geared towards optimizing network lifetime when sensor nodes possess enhanced processing capabilities, and processing tasks have higher computational complexity. However, for efficient operation, our approach also needs to be configured appropriately with regards to relevant protocol characteristics. We will demonstrate analysis later in the paper where a TDMA-based protocol is selected and carefully integrated with our partitioning methods to provide a practical network configuration for our proposed methods.

In [Li et al. 2006], the authors present several localized topology control strategies for heterogeneous network environments. In this approach, each wireless sensor node locally adjusts its transmission power and selects which neighbor nodes to communicate with according to information about sensor nodes within its local neighborhood. In [Chatterjea et al. 2008], the authors present a distributed scheduling algorithm for managing data aggregation according to correlated information among network nodes so that the transmission of redundant data can be minimized. The algorithm is also able to adapt to network topology changes (i.e., events in which nodes are added to or removed from the network). In [Hoang and Motani 2008], the authors present a collaborative scheme to allow network nodes to compress and then transmit data based on correlated information obtained in a broadcasting, cluster-based network. These methods are proposed to conserve energy due to transmission overhead and improve network lifetime. Our work is different from and complementary to these previous bodies of work in that we systematically analyze the functional behavior for the targeted application to derive an efficient distribution of application tasks, including tasks required for computation and inter-node communication, across a network.

Various useful approaches have been suggested previously to reduce energy consumption in sensor nodes that are capable of executing more computationally-intensive tasks. In [Shih et al. 2001], the authors have distributed the fast Fourier transform (FFT) function over a master node and slave nodes to reduce energy consumption by moving the function from a cluster head node to slave nodes. In [Kumar et al. 2003], energy and latency trade-offs are considered for different computational capabilities between master and slave nodes. In [Wang and Chandrakasan 2001], the authors develop an approach that partitions applications between master and slave nodes, and also applies dynamic voltage scaling to further reduce power consumption. In contrast to the above approaches, our proposed energy-driven analysis and partitioning for an application graph are targeted at the application level. Moreover, the partitioning method presented in this paper applies coarse-grain analysis of dataflow graphs, as well as integration within a dataflow-based DSP design tool. This tool, called the DIF (dataflow interchange format) package, is introduced in [Hsu et al. 2005].

In [Tang and Xu 2008], Tang and Xu develop adaptive data collection strategies to maximize the accuracy of data collected by a base station from sensor nodes under constraints on network lifetime. In contrast to Tang's approach, we consider options for pre-processing data on sensor nodes before information is transmitted

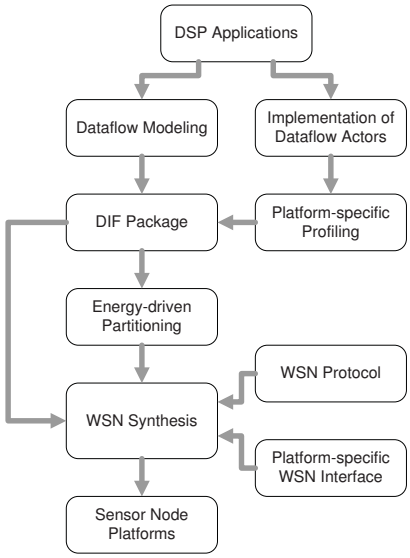


Fig. 1. Overall design flow.

to the "base station" (the master node, in our context). Therefore, our objective is to maximize network lifetime by finding the most appropriate resource distribution for data processing, while considering the computation and communication requirements and their underlying trade-offs for a given application.

Fig. 1 presents the overall design flow associated with our proposed methodology for energy-efficient master/slave signal processing in sensor networks. This methodology involves distributing dataflow representations of DSP applications across WSNs. To specify and analyze these dataflow representations we use DIF and the associated DIF package [Hsu et al. 2005]. In the design flow demonstrated in this paper, we use DIF to specify the behavior of DSP applications. Moreover, we assign power and timing information to components of each dataflow graph. These information are based on measurements and simulations carried out for those components on the targeted hardware platforms. The DIF package then parses the DIF specification of the application and builds an intermediate representation on which we apply algorithms for analysis and optimization of WSN energy usage. Preliminary summaries of this methodology have been presented in [Ko et al. 2006] and [Shen et al. 2007], and our application of the methodology to distributed speech recognition has been demonstrated in [Shen et al. 2008]. This paper goes beyond the developments of [Ko et al. 2006], [Shen et al. 2007], and [Shen et al. 2008] by analyzing the computational complexity in more depth, and developing a dynamic workload redistribution scheme for adaptively optimizing energy efficiency in response to failed network nodes (e.g., due to exhausted battery capacity or loss of functionality due to an attack).

3. BACKGROUND: SYNCHRONOUS DATAFLOW MODELING

Digital signal processing algorithms are widely used in the processing of sensor data, and often require intensive computation. The behavior of many DSP applications can be characterized by regular patterns of computation and modeled efficiently through dataflow graphs. By analyzing a well-designed dataflow model of an application, operational efficiency can be estimated and optimized accordingly, such as with the use of dataflow-based algorithms for scheduling, memory management, consistency analysis among different sample rates, and hardware/software partitioning (e.g., see [Bhattacharyya et al. 1995], [Kalavade and Suhrahmanyam 1997], and [Lee and Messerschmitt 1987]).

Dataflow models of computation are widely used for modeling DSP applications (e.g., see [Bhattacharyya et al. 1995]). In such models, applications are dataflow graphs. A dataflow graph is a directed graph $G = (V, E)$ in which application nodes (or so-called actors) $v \in V$ represent computational modules for executing tasks, and directed edges $e \in E$ represent buffers for storing data values, encapsulated as *tokens*, as they pass between computations. Actors can be fired only if sufficient numbers of tokens are available in their input buffers to perform meaningful computations. Whenever an actor is fired, it consumes certain numbers of tokens from its input edges, executes its associated computational task, and produces certain numbers of tokens on its output edges. Also, there is a non-negative integer *delay* associated with each dataflow graph edges, where each unit of delay corresponds to an initial token that is placed in the corresponding buffer. DSP applications typically execute continuously on large, often-unbounded input streams, and thus each dataflow actor executes in an iterative fashion, through a sequence of "invocations" of the corresponding task as it is applied to successive tokens or blocks of tokens from the associated input edges.

When a DSP applications is mapped into a clustered sensor network, each slave node generally captures data from its set of one or more sensors, and this captured data can then be sent to the associated master node immediately, or the data can be pre-processed to some degree within the slave node before it is sent to the master node.

When analyzing data processing functionality within a dataflow graph in terms of energy-efficient workload distribution, it is useful to consider carefully the rates at which actors produce and consume data from their incident output and input edges, respectively. For this purpose, we use a specialized form of dataflow called synchronous dataflow (SDF) [Lee and Messerschmitt 1987], which is widely used in the design and implementation of DSP applications. SDF is a restricted form of dataflow in which the number of tokens produced by each actor onto each of its output edges is constant (i.e., identical for all invocations of the actor), and similarly, the number of tokens consumed from each actor from each input edge is constant.

An SDF graph $G = (V, E)$ can be scheduled statically by constructing a *periodic schedule*, which is a schedule that executes each actor at least once, does not deadlock, and produces no net change in the number of tokens that reside in any of the graph edges. Such a schedule is usually derived by first solving the system of *balance equations* for the SDF graph. The balance equations model the requirement that

for each edge e, $x(src(e))prd(e) = x(snk(e))cns(e)$, where $x(A)$ denotes the number of invocations of actor A in the targeted periodic schedule; $src(e)$ represents the source actor of edge e ; $snk(e)$ represents the sink actor of edge e; $prd(e)$ represents the number of tokens produced onto e by each invocation of $src(e)$; and $cns(e)$ represents the number of tokens consumed from e by each invocation of $snk(e)$.

Given a properly constructed SDF graph, solving the system of balance equations for the unknown invocation counts $\{x(A)|(A \in V)\}$ yields a unique minimal, positive integer solution, which is called the *repetitions vector* of G, and is often denoted by the symbol q [Lee and Messerschmitt 1987]. A periodic schedule for G can then be implemented by executing each actor A exactly $q(A)$ times according to the dataflow requirements (token consumption) requirements of the actors. Because the repetitions vector q represents a minimal solution to the balance equations, a periodic schedule constructed in this way from q is referred to as a *minimal* periodic schedule.

An iterative execution of G can be achieved with bounded memory by simply encapsulating such a periodic schedule within a loop. The iteration count of this "top-level loop" would be based on the total number of samples that the application needs to process (if known in advance). Alternatively, the top-level loop can be implemented as an infinite loop if there is no pre-defined bound on the "extent" of the input signals. Software-based implementations of SDF graphs are usually achieved through infinite-loop-encapsulations of periodic schedules.

4. MODELING WORKLOAD DISTRIBUTIONS

4.1 Partitions of SDF graphs

Given a targeted WSN system, we start by modeling its signal processing functionality as an acyclic SDF graph. A broad class of useful signal processing applications can be modeled in this way (e.g., see [Bhattacharyya et al. 1999], [Lee and Messerschmitt 1987]). Extensions of our techniques to arbitrary SDF topologies (i.e., those containing cycles) are omitted in this paper due to space limitations.

We model workload distributions between a master node and a slave node as *feedforward partitions* (*FFPs*) of the dataflow graph that represents the overall signal processing computation to be performed. Here, by an FFP of a directed graph, we mean a decomposition of the given graph into two disjoint subgraphs G_1 and G_2 such that all edges in G that cross the partition are directed from vertices in G_1 to vertices in G_2 (i.e., the "crossing edges" are oriented in the same direction with respect to the two subgraphs). Formally, a subgraph of a directed graph $G = (V, E)$ is a directed graph $G' = (V', E')$, where $V' \subseteq V$, and E' consists of all edges in G whose source and sink vertices are both contained in V' — in other words,

$$E' = \{e \in E|(src(e) \in V') \text{ and } (snk(e) \in V')\}.$$

Thus, an FFP of G is an ordered pair (G_1, G_2) of subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, and for every edge $e \in E$ that is neither contained in E_1 nor E_2, we have that $src(e) \in V_1$ and $snk(e) \in V_2$.

4.2 Parameterized dataflow and adaptive distributions

In this paper, DSP applications that involve purely static dataflow behavior are modeled using SDF, as described in Section 4.1. However, modern WSN applications do not always conform to the restricted semantics of SDF. Instead, they may require more general semantics to model application-specific sensing activities. For example, in some applications, the sensed data input must be controlled so that the following digital processing sections are executed only when needed. Thus, in this paper we also use the more general *parameterized synchronous dataflow* (PSDF) model of computation. PSDF results from the integration of SDF with the meta-modeling framework of parameterized dataflow [Bhattacharya and Bhattacharyya 2001]. PSDF expresses a wide range of dynamic dataflow behaviors while preserving much of the useful analysis and synthesis capability of SDF [Bhattacharya and Bhattacharyya 2001]. Furthermore, PSDF provides a systematic framework for integrating various forms of SDF analysis techniques, such as the ones we develop in this paper, into a more general, dynamic dataflow setting.

Each hierarchical subsystem in a PSDF specification consists of three distinct graphs: the init graph (Φ_i), subinit graph (Φ_s), and body graph (Φ_b). In our use of PSDF, we employ the body graph to model the main functional behavior of an application and the init and subinit graphs to describe any input-dependent changes to the body graph functionality. According to PSDF semantics, the init graph is invoked prior to each invocation of the associated (hierarchical) parent subsystem, while the subinit graph is invoked prior to each invocation of the associated body subsystem. Such interaction allows for two distinct levels of reconfiguration control. Based on this modeling approach, and the scheduling features provided by PSDF, low-overhead, "quasi-static" schedules can be generated for hand-coded implementation or software synthesis [Bhattacharya and Bhattacharyya 2001]. Here, by a quasi-static schedule, we mean an ordering of execution for the functional modules (dataflow actors) whose structure is largely fixed at compile time, with a small number of decision points or symbolic adjustments made at run-time based on the values of relevant input data.

Another motivation that we have for using PSDF in this work — in addition to modeling dynamic behavior in signal processing functionality — is to help develop an advanced workload redistribution scheme for dynamic topology management. Based on our redistribution approach, whenever the network size changes, a the overall workload distribution result is efficiently adapted in terms of the new network structure. This is useful for scenarios in which sensor nodes can enter or exit the system at run time. Such dynamics may arise, for example, due to incremental node deployments and exhausted batteries, respectively.

5. ENERGY-DRIVEN PARTITIONING

In our proposed concept of EDP, an application dataflow graph (e.g., an SDF graph) $G = (V, E)$ is taken as input, and an FFP (G_s, G_m) is computed for G, where $G_m = (V_m, E_m)$ and $G_s = (V_s, E_s)$ respectively represent the distribution of the partitioned application graph onto master and slave nodes. The *partition cut* E_c associated with such an FFP is defined to be the set of edges that "cross the

partition":

$$E_c = \{e \in E | (src(e) \in V_s) \text{ and } (snk(e) \in V_m)\}.$$

The network size is initialized at design time, and can be changed at run time. The master node performs the function of collecting data from all other slave nodes; therefore, the communication is one-way. In order to maintain predictability and fairness across the slave nodes, we define a *scheduled iteration* (S) of a partitioned application graph as one functional iteration of the application such that all master and slave nodes complete a predefined amount of computation and transmit/receive processed data as needed based on the computation. When the application graph is an SDF graph, such an iteration in our methodology corresponds to a minimal periodic schedule for the graph.

In the particular solution demonstrated in this paper, we define such a scheduled iteration based on the periodic behavior of the TDMA-based communication pattern in the network. Our energy consumption analysis for partitioning an application graph is targeted at the application level, and we assume that the underlying protocol design can handle problems of packet loss and collision without changing the higher level application behavior. Therefore, we formulate the system energy consumption model at a high level, based on the schedule iteration concept, and independent of implementation details for the underlying communication protocols. We have validated the correctness and utility of this multi-level design model in our experiments.

Each actor v of the application graph is characterized by two pairs of attributes $(P_m(v), t_m(v))$ and $(P_s(v), t_s(v))$, where $P_m(v)$ ($P_s(v)$) denotes the power consumed and $t_m(v)$ ($t_s(v)$) denotes the execution time when actor v executes on the master node (on a slave node). Similarly, each edge e of the application graph is characterized by pairs of attributes $(P_m(e), t_m(e))$ and $(P_s(e), t_s(e))$ that represent the power consumption values and latencies for local (intra-node) communication on the master and slave nodes, respectively. These values are associated with the transfer of one token of data across the edge.

For modeling the energy consumption of communication, we distinguish between edges that cross the partition cut (edges in E_c), and edges that do not (edges in $E - E_c$). Note that, based in our application partitioning model, edges in E_c correspond to points in the dataflow graph where tokens are transmitted and received through the wireless communication channel.

5.1 Energy formulations for data processing and communication

The total number of data values transferred across an SDF edge e in a minimal schedule period of an SDF graph G can be expressed [Lee and Messerschmitt 1987] as

$$\tau(e) = q(src(e)) \cdot prd(e) = q(snk(e)) \cdot cns(e). \tag{1}$$

Thus, given an FFP for G, the number of tokens X_c that crosses an FFP partition within each scheduled iteration of an SDF application graph is expressed as

$$X_c = \sum_{l \in E_c} \tau(l). \tag{2}$$

The computational energy consumed in each scheduled iteration by the slave and master nodes, respectively, can then be formulated as

$$E(s) = \sum_{\forall v \in V_s} q(v) \cdot P_s(v) \cdot t_s(v) + \sum_{\forall e \in E_s} 2\tau(e) \cdot P_s(e) \cdot t_s(e), \tag{3}$$

and

$$E(m) = \sum_{\forall v \in V_m} q(v) \cdot P_m(v) \cdot t_m(v) + \sum_{\forall e \in E_m} 2\tau(e) \cdot P_m(e) \cdot t_m(e). \tag{4}$$

Furthermore, the energy consumption of communication for transmitting and receiving data tokens that cross the partition cut can be expressed as

$$E(t) = \sum_{\forall l \in E_c} \tau(l) \cdot P_t \cdot t_c = X_c \cdot P_t \cdot t_c, \tag{5}$$

and

$$E(r) = \sum_{\forall l \in E_c} \tau(l) \cdot P_r \cdot t_c = X_c \cdot P_r \cdot t_c, \tag{6}$$

where P_r represents receive-mode transceiver power consumption; P_t represents transmit-mode transceiver power consumption; and t_c models the time required for inter-node communication, including transceiver turn-on time and time for executing modulation/demodulation tasks. The value of t_c is dependent on transceiver configuration settings, such as the data rate, that are associated with communication tasks.

5.2 Energy cost per scheduled iteration for EDPs

The total energy consumption on a single sensor node for one scheduled iteration is denoted by E_{iter}. In a network cluster that consists of a single master node and N_s slave nodes, the master node iterates N_s times to process data frames from all of its slave nodes. Therefore, the energy consumption on each slave node for one scheduled iteration is

$$E_{iter}(s) = E(s) + E(t) + E_{idle}(s), \tag{7}$$

where, $E_{idle}(s)$ denotes the energy consumption when a slave node stays in the power-saving "idle" mode after processing and communicating data tokens. The total energy consumption on the master node for one scheduled iteration is formulated as

$$E_{iter}(m) = N_s \cdot (E(m) + E(r)) + E_{idle}(m). \tag{8}$$

Note that $E_{idle}(m)$ could be small because the workload on the master node is significantly heavy compared to the slave nodes in each scheduled iteration. Therefore, *the system energy cost per scheduled iteration*, which we refer to more concisely as *the system energy cost* is defined as

$$E_{sys} = E_{iter}(s) + E_{iter}(m). \tag{9}$$

To compare the network lifetime according to the selection of different FFPs, we define the worse case network lifetime as the total time that elapses before the

first sensor node in the network runs out of energy. Thus, our network lifetime estimation is formulated as

$$min\{T_{lifetime}(master), T_{lifetime}(slaves)\}, \tag{10}$$

where

$$E_{source} = T_{lifetime} \cdot E_{iter} \cdot R_{avg}. \tag{11}$$

Here, R_{avg} is the average rate at which scheduled iterations are executed; and E_{source} is the total energy stored in the given energy source (e.g., battery). Based on a given partitioning result, the network lifetime is determined in terms of the minimum lifetime between the resulting master and slave node configurations.

5.3 Latency and data dependency constraints

To apply our energy-aware optimization techniques in conjunction with a specific sensor network communication protocol, we derive protocol-dependent local and global latency constraints in our targeted partitioning problem. We also develop a data dependency constraint for preventing cyclic dependencies across an FFP (to avoid bi-directional communication complexity).

The *local* latency indicates the execution time period from the input of the master (slave) subgraph to the output of the master (slave) subgraph. This is expressed as

$$L_l = \sum_{\forall v \in V_s \text{ or } V_m} q(v) \cdot t(v) + \sum_{\forall e \in E_s \text{ or } E_m} 2\tau(e) \cdot t(e) + \sum_{\forall l \in E_c} \tau(l) \cdot t(l). \tag{12}$$

The *global* latency, L_g, is defined as the latency of one scheduled iteration. This represents the overall computation and communication time period from the input to the output of a partitioned application graph. That is,

$$L_g = L_l(slaves) + N_s \cdot L_l(master) + N_s \cdot X_c \cdot D. \tag{13}$$

Here, D is the propagation delay for transmitting each data token. The term, $N_s \cdot X_c \cdot D$, is added to incorporate transmission delays in the network. In general, D is very small compared to the local latency, and therefore the whole term $N_s \cdot X_c \cdot D$ is negligible in our latency model. From (13), we notice that the master node should process the data from all of the slave nodes, and therefore the term $N_s \cdot L_l(master)$ is induced for presenting the total local latency that the master node requires in each scheduled iteration. However, the slave nodes can operate in parallel within each scheduled iteration, and thus, the latency required for slave node processing is independent of N_s.

We incorporate the global latency across the network into the formulation of protocol latency. For this purpose, we first define L_{sync} as the synchronization latency added during each synchronization period. Therefore, $N_s \cdot L_{sync}$ denotes the total synchronization latency within a scheduled iteration during each synchronization period. In our predefined topology, the routing delay is omitted because of the one-hop distance between master and slaves nodes. In our experiments, the MAC protocol design is based on TDMA with N_p time slots of uniform time period t_p. We define $L_p = N_p \cdot t_p$ (i.e., the length of a complete TDMA time frame) as the protocol-dependent communication latency. Therefore, the latency constraint

across the network for one scheduled iteration can be bounded as

$$N_s \cdot L_{sync} + L_g \leq L_p. \tag{14}$$

Here, both N_p and t_p are parameters determined by the TDMA configuration that is being used. Recall that the scheduled iteration is defined to maintain predictability and fairness for processing and communicating data tokens across the network. From (14), we observe that this property is satisfied when the TDMA-based protocol is applied.

In TDMA, the nodes in a network communicates with one another at pre-defined time slots to prevent collisions when accessing the wireless channel. Therefore, there is a consistent frame-by-frame communication pattern for TDMA-based protocols. When the TDMA frame length is fixed by setting N_p and t_p, (14) captures the property that the latency of scheduled iteration is also constrained. That is, the operations within one scheduled iteration across the network are predictable in terms of latency when the TDMA-based protocol is applied and satisfies (14).

Note that if $L_{sync} \ll L_p$ (i.e., L_{sync} is small enough compared to L_p), $D \approx 0$, and there exists a partition cut such that $L_l(slaves) = N_s \cdot L_l(master)$, then a fully-balanced workload is achieved between the master node and the slave nodes. In such a case, from (13) and (14), we can find the minimum requirement of t_p as $2 \cdot L_l(slaves)/N_s$ if $N_s = N_p$. Generally, $N_p \geq N_s$, and it is difficult to find a fully balanced partition in terms of given actor and edge attributes for an application graph.

To prevent cyclic dependencies (potential deadlock), we consider another constraint in our partitioning formulation. In particular, a partition cut must satisfy the following validity (dependency) constraint:

Given an application graph, $G = (V, E)$, a partition cut E_c is *valid* if

$$\begin{aligned} \forall l \in E_c, src(l) \in G_s \Rightarrow preds(src(l)) \in G_s \text{ or } snk(l) \in G_s \\ \Rightarrow succs(snk(l)) \in G_m, \end{aligned} \tag{15}$$

where $preds(v)$ ($succs(v)$) represents the set of immediate graph predecessors (successors) of actor v.

In summary, solving the energy-driven partitioning problem means to find a partition cut E_c of an application graph G so that a master-slave WSN topology having maximum lifetime results from the partitioned subgraphs G_m and G_s. That is, we wish to find a solution that satisfies

$$max\{min[T_{lifetime}(master), T_{lifetime}(slaves)]\}, \tag{16}$$

subject to 1) $G_m \neq \emptyset$; 2) $G_s \neq \emptyset$; 3) latency constraint is satisfied; and 4) E_c is a valid partition cut if data dependency constraint is satisfied.

5.4 Dynamic topology management

By applying the PSDF model in conjunction with the EDP approach, we provide an advanced form of workload redistribution that can be used for dynamically-changing network sizes — i.e., for scenarios in which sensor nodes can enter or exit the system at run time. Such dynamics may arise, for example, due to incremental node deployments and exhausted batteries, respectively. The number of network nodes is characterized by a dynamic parameter that represents the number of slave

nodes existing in the system. This parameter is maintained and broadcasted by the master node. From the customized protocol described previously, the master node can determine the number of active nodes existing in the system based on the requests it receives from the slave nodes and the usage of time slots for its TDMA schedule. Thus, soon after any change in network size, all active slave nodes are informed about the change by the master node, and furthermore, the associated EDP configuration will generally be adjusted so that the workload distribution is efficiently adapted to the new network structure.

Based on our PSDF modeling approach, at design time, a given application graph is analyzed and the corresponding EDP configurations are determined in terms of different network size settings. These EDP results correspond to different partition cuts on an application graph as the network size varies. The results are stored on the sensor nodes for driving workload redistribution at run time. In other words, the number of slave nodes is a parameter that is maintained on the master node, and broadcast periodically to the slave nodes. On the slave nodes, this parameter drives a *quasi-static schedule* (a dynamic schedule with a relatively large portion of the structure fixed statically, at design time) that allows the schedule to adapt efficiently to changes in network size, and associated changes in energy-driven partitions of network functionality. We refer to this integration of quasi-static scheduling with the EDP approach as *quasi-static energy-driven partitioning* (QS EDP).

For a reasonable number of candidate EDP results (supported network sizes), many practical platforms can easily accommodate the program memory requirements for the scheduling information associated with QS EDP. For example, on the Texas Instruments CC2430F128 platform, a total 128K bytes of on-chip program memory is available, and based on our implementation, the program memory cost of QS EDP is approximately 1K bytes per unit of supported network size.

Fig. 2 gives an example to illustrate this workload redistribution scheme. Fig. 2(a) shows an application graph with four partition cut candidates (C_1 — C_4) that are considered individually. We refer to these candidate EDP results to "static EDP" results. Then, we compare these static EDP results in terms of estimated system energy costs (i.e., (9)) as a function of the number of slave nodes existing in the network. The workload redistribution scheme with respect to QS EDP results is illustrated by the blue solid curve in Fig. 2(b). From Fig. 2(b), we observe that by adding sensor nodes to a system that allows for workload redistribution, a quasi-static EDP always adapts to the minimum energy cost from its available static EDP results. This energy efficient adaptive approach results in overall system lifetime improvement in our experiments, which are be demonstrated in Sections 8.2 and 8.3.

6. ANALYSIS AND SOLUTIONS FOR THE EDP PROBLEM

The EDP problem is NP-complete, where the NP-hardness of EDP can be established with a reduction from the well-known *partition problem* [Garey and Johnson 1979]. Here, we consider a decision version of the EDP problem, where we are given latency and energy constraints, and the objective is to determine whether an EDP solution exists that satisfies the constraints. The energy constraint here refers to the maximum energy consumption among master and slave nodes, which directly influences the lifetime of the network.

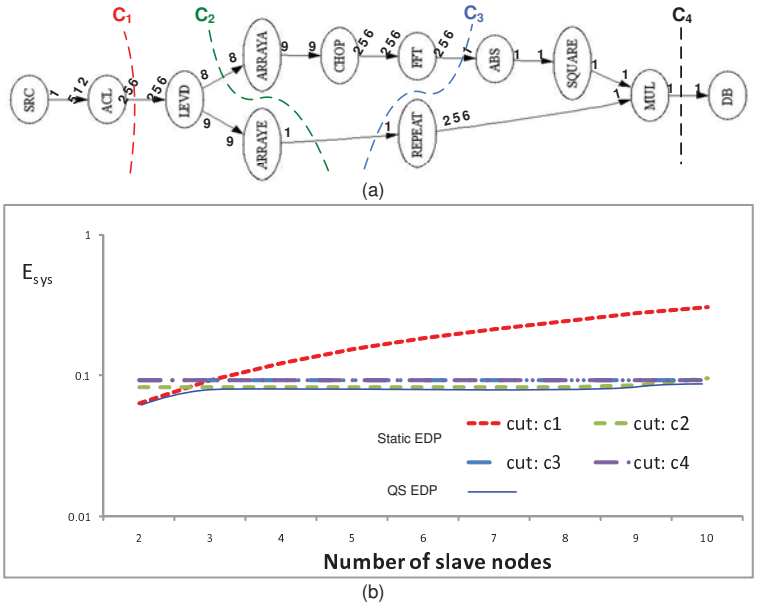


Fig. 2. Example of workload redistribution scheme.

Claim 1: The EDP problem is NP-complete.

Proof. We show in this section that the energy-driven partitioning (EDP) problem is NP-complete in the size of the input graph. We first show that the EDP problem is in NP (i.e., $EDP \in$ NP). The certificate for the EDP problem consists of a slave graph G_s, a master graph G_m, and a partition cut E_c. Given pairs of power consumption and time attributes as the weights of actors and edges, we can compute $E(s)+E(t)$ and $E(m)+E(r)$ with respect to various partition cuts based on (3), (4), (5), and (6). Each such computation process can be performed in polynomial time by stepping through each actor and edge; summing up the actor and edge weights; and summing the cut-edge weights.

Next, we show that the partition problem is polynomially reducible to the EDP problem. The partition problem is a well-known NP-complete problem (e.g., see [Cormen et al. 2001], [Garey and Johnson 1979]). Intuitively, in the partition problem, we are given a multiset S of positive integers, and we must determine whether or not there is a way to partition S into two subsets S_1 and S_2 such that the sum of the elements in S_1 equals the sum of the elements in S_2.

To show that the partition problem is polynomially reducible to EDP, we first suppose that we are given an instance I of the partition problem. From I, we can derive a corresponding instance I' of EDP by constructing a *homogeneous* SDF (HSDF) graph with a single source actor v_{src}, and a single sink actor v_{snk}, and in I which every item in I' is instantiated as a corresponding actor in . An HSDF graph is an SDF graph in which $prd(e) = cns(e) = 1$ for every edge e, and hence, $q(A) = 1$ for all every actor A [Lee and Messerschmitt 1987]. Here, both v_{src} and v_{snk} have 0-

valued weights (for both power consumption and time). The construction continues by having each "corresponding actor" in I' (i.e., each actor in I' that corresponds to an element of I) connected with an edge *from* the source actor and with another edge *to* the sink actor. The weight of each actors in I' is assigned as

$$(power, time) = (x, 1), \tag{17}$$

where x is the positive integer value of the corresponding element in I . The weight of each edge in I' is assigned as

$$(power, time) = (0, 0). \tag{18}$$

The latency constraint associated with the derived EDP instance I' is taken to be infinite (equivalently, the latency constraint can be taken to be so large compared to the actor execution time weights that it will always be satisfied), and the energy constraint is taken to be

$$E_{constraint} = \frac{S}{2}, \tag{19}$$

where S is the sum of all elements of I. By our construction, S can also be expressed as

$$S = \sum_{v \in V} p(v), \tag{20}$$

where V is the set of all actors in I', and $p(v)$ represents the power consumption weight associated with actor v in I'.

Fig. 3 illustrates this graph construction process with an instance I of the partition problem and the corresponding instance I' of the EDP problem. Now for a graph $G = (V, E)$ with zero-valued edge weights, no constraint on latency, and energy constraint $E_{constraint}$ as defined in (19), our decision version of EDP involves finding a FFP $G_s = (V_s, E_s)$ and $G_m = (V_m, E_m)$ such that the maximum master/slave energy consumption associated with the partition is less than or equal to $E_{constraint}$. By construction (since each actor execution requires unit time, and edges have zero cost), the energy consumption associated with an FFP $((V_s, E_s), (V_m, E_m))$ for I' can be expressed as

$$max\{(\sum_{v \in V_s} p(v)), (\sum_{v \in V_m} p(v))\}. \tag{21}$$

From (19), (20), and (21), it follows that an FFP with the given energy constraint (i.e., a solution that demonstrates feasibility of I') exists if and only if the vertices can be partitioned into two subsets V_1 and V_2 such that

$$\sum_{v \in V_1} p(v) = \sum_{v \in V_2} p(v) = E_{constraint} = \frac{S}{2}. \tag{22}$$

This would in turn mean that

$$\sum_{v \in V_1} I(v) = \sum_{v \in V_2} I(v) = \frac{S}{2}, \tag{23}$$

which demonstrates the feasibility of I.

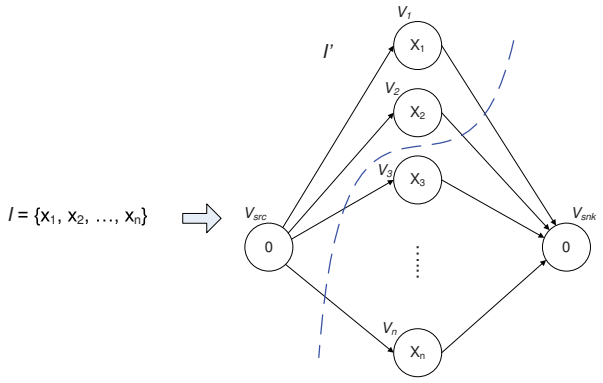


Fig. 3. Graph construction for deriving NP-complete proof for the EDP problem.

In summary, we have shown that given an instance I of the partition problem, we can derive a corresponding instance I' of the EDP problem such that I is feasible (has a solution) if and only if I' is feasible.

In the example of Fig. 3,

$$\begin{aligned} V_s = \{v_{src}, v_1, v_2\}, V_m = \{v_3, \ldots, v_n, v_{snk}\}, \\ E_s = \{e_{s1}, e_{s2}\}, e_m = \{e_{3s}, \ldots, e_{ns}\}, \\ \text{and } E_c = E - E_s - E_m = \{e_{1s}, e_{2s}, e_{s3}, \ldots, e_{sn}\}. \end{aligned} \tag{24}$$

Furthermore, $\sum_{i=1}^{2} x_i = \sum_{j=3}^{n} x_j = E_{constraint}$, and the weight of the cut E_c is 0. It is easily verified that all steps involved in the transformation between I and I' can be performed in polynomial time. Therefore, from the known NP-hardness of the partition problem, we can conclude that EDP is NP-hard, and since we have already shown that $EDP \in$ NP, it follows that EDP is NP-complete.

Our proof in this section actually demonstrates that a significantly restricted version of the EDP problem (infinite-valued latency constraint, zero-valued edge weights, and HSDF-based dataflow) is NP-complete. It should be noted that the partition problem is readily solvable by approximation schemes, but the additional constraints and dimensions of EDP appear to make a more complex heuristic approach more appropriate.

□

6.1 The heuristic approach for EDP

A variety of heuristic algorithms for graph partitioning have been developed. The Kernighan-Lin (K-L) [Kernighan and Lin 1970] and Fiduccia-Mattheyses (F-M) [Fiduccia and Mattheyses 1982] algorithms are both popular heuristic algorithms for graph partitioning. These algorithms involve incrementally exchanging vertices across the partition cut if the exchange improves the targeted figure of merit.

To formulate our heuristic approach for EDP, we adopt useful ideas from the K-L and F-M algorithms. To help describe our algorithm, we define the cost function

of a partitioning result as

$$CT(G_m, G_s, E_c) = max\{E(s) + E(t), E(m) + E(r)\}.$$

We seek to minimize CT so that the corresponding maximized network lifetime can be obtained. Based on this, we formulate the *Gain* function δ for moving actor v from one side of a given cut to the other. Intuitively, $\delta(v)$ gives the potential energy reduction or increase for both G_m and G_s whenever an actor v is switched from one subgraph to the other. δ values can be computed and updated efficiently based on the formulations developed earlier in the previous section. To discuss the algorithm formulation in more detail, it is useful to define the "cost" of a given partitioning to be the maximum energy consumption for transmitting ($E(t)$) and receiving ($E(r)$) data tokens across the partition cut, E_c, plus the energy consumption of computation ($E(s)$ and $E(m)$) for the two subgraphs, G_s and G_m. That is, we seek to minimize CT for a given SDF graph $G = (V, E)$ so that the corresponding maximum system lifetime can be obtained.

To help derive $\delta(v)$, we define a *gain pair function* $d(v)$ for switching vertex v from one subgraph into the other subgraph based on possible energy variations on master and slave nodes. The value $d(v)$ is expressed as

$$\begin{aligned}
&d(v) = \{\eta(G_s), \eta(G_m)\} = \\
&\{E(v) + \eta(cut, t), N_s \cdot (-E(v) + \eta(cut, r))\}, \text{ if } v \in G_s \\
&\{-E(v) + \eta(cut, t), N_s \cdot (E(v) + \eta(cut, r))\}, \text{ if } v \in G_m
\end{aligned}$$

where

$$\begin{aligned}
&\eta(cut, t) = X_c(v) \cdot P_t \cdot t_c - \bar{X}_c(v) \cdot P_t \cdot t_c, \\
&\eta(cut, r) = X_c(v) \cdot P_r \cdot t_c - \bar{X}_c(v) \cdot P_r \cdot t_c, \\
&X_c(v) = \sum_{e \in cutedges(v)} prd(e) \cdot q(src(e)), \\
&\text{and } \bar{X}_c(v) = \sum_{e \in noncutedges(v)} prd(e) \cdot q(src(e)).
\end{aligned}$$

Here, $cutedges(v)$ is the set of edges of vertex v that cross the partition cut (thus, $cut(v) \subseteq E_c$), and $noncutedges(v)$ is the set of edges of vertex v that do not cross the cut. Therefore, $X_c(v)$ denotes the number of data tokens to be transmitted and received by v due to the existing partition cut, and $\bar{X}_c(v)$ denotes corresponding number of data tokens to be transmitted and received due to the partition cut that would result from moving v across the existing cut. Moreover, for a given vertex v, $\eta(cut, t)$ ($\eta(cut, r)$) denotes the improvement in communication energy for transmitting (receiving) data tokens respectively across the partition cut if vertex v is moved across the existing cut. Here, a negative "gain" means that such a move would cause a net increase in communication energy.

Based on the gain pair function for each candidate vertex v, we derive $\delta(v)$ by:

$$\begin{aligned}
\delta(v) &= CT - max\{[E(s) + E(t), N_s \cdot (E(m) + E(r))] - d(v)\} \\
&= CT - max\{E(s) + E(t) - \eta(G_s), N_s \cdot (E(m) + E(r)) - \eta(G_m)\}
\end{aligned}$$

Note that $\delta(v)$ can be positive- or negative-valued. A positive value of $\delta(v)$ means

```
input: G = (V, E)
output: G_m = (V_m, E_m) , G_s = (V_s, E_s) , and E_c .
begin
1   Create an initial partition.
2    Compute CT(G_m, G_s, E_c) and check constraints in (16)
     for the initial partition.
3   do
4      for (each v ∈ V) do
5         if v violates constraints in (16)
6            Mark v locked.
7         else
8            Mark v unlocked, and Compute δ(v) for v .
9      while (unlocked nodes exist) do
10        Find one unlocked node v with maximum δ(v) .
11        Add (v, δ(v)) to the ordered list, order_L .
12        Lock v , and update δ for all unlocked nodes.
13     Select first k nodes that maximizes ∑_v δ(v) from order_L .
14     if ∑_v δ(v) > 0

15        for (each v ∈ selected k nodes ) do
16           Update G_m and G_s based on switching v .
17           Update
                CT(G_m, G_s, E_c) = CT(G_m, G_s, E_c) − δ(v) .

18     while ∑_v δ(v) > 0
end
```

Fig. 4. Algorithm pseudocode for our heuristic approach to solving the EDP problem.

that there is an improvement in the cost function CT if actor v is moved across the existing cut. On the other hand, a negative value for $\delta(v)$ represents a deterioration of CT if v is moved.

6.2 Performance comparison and analysis

Based on the *Gain* formulation mentioned above, Fig. 4 shows a pseudocode specification of our heuristic for solving the EDP problem, and Fig. 5 provides performance comparisons between our heuristic approach and the exhaustive search for the EDP. In Fig. 5, we examine the performance of our approach on several randomly-generated, synthetic SDF graphs. Fig. 5(a) shows that while the exhaustive search time quickly becomes infeasible for moderate size examples, our heuristic produces comparable results in a fraction of the time. In our implementation of exhaustive search, the constraints in (16) are verified for each candidate partitioning to filter out invalid results.

Fig. 5(b) shows a comparison of E_{sys} versus run time for successive algorithm iterations ("run time testing iterations") on several synthetic SDF graphs. Here, the total number of run time testing iterations represents the maximum number of allowable actor switches. Moreover, to examine the impact of graph complexity in terms of the numbers of actors and edges, we normalize certain graph attributes when constructing the synthetic graphs. Specifically, we normalize the assignments of production and consumption rates to 1 for all the edges, and we also normalize all energy-related attributes (i.e., the power and time estimates) to 1 for all of the actors and edges. We observe from Fig. 5(b) that our heuristic algorithm converges significantly faster than exhaustive search for each synthetic graph. Note here that

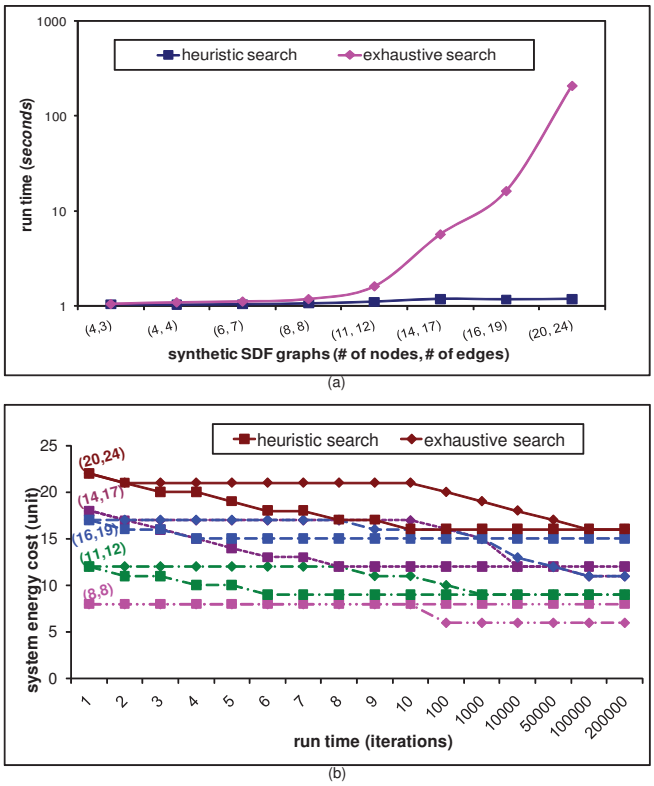


Fig. 5. Performance comparison for EDP schemes: (a) Run time comparison based on the complexity of synthetic SDF graphs. (b) Cost versus run time comparison for selected synthetic SDF graphs.

if both search algorithms converge on the same point, then both algorithm have targeted results that have identical (optimal) quality; in other cases, the exhaustive search algorithm finds an optimal solution, whereas the solution returned by the heuristic algorithm is suboptimal. Our heuristic algorithm for solving the EDP problem has $O(|V|^2|E|)$ time complexity.

7. EXPERIMENTS

7.1 Experimental DSP computations

In this section, we first choose several DSP computations modeled by SDF graphs to illustrate the operation of EDP, and show the corresponding EDP results. Next, we demonstrate the development of a practical application for distributed speech recognition. The first DSP computation, which involves maximum entropy power spectrum (MEPS) computation, is adapted from the Ptolemy II design environment [Eker et al. 2003] . We use this example to illustrate step-by-step the operations involved in EDP based on SDF modeling. Here, we assume that there is 1 master node and there are 5 slave nodes in the targeted network. Fig. 6 shows three different cases of partition cuts for the MEPS computation, where (a) and

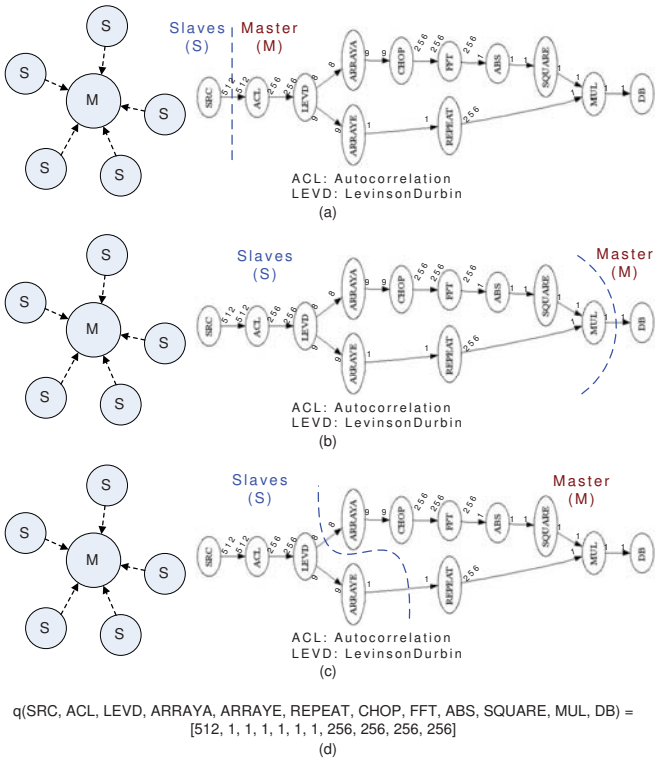


Fig. 6. (a)-(c) show various partitioning cases for a WSN that performs maximum entropy power spectrum computation. (d) represents the repetition vector of the modeled SDF graph.

(b) show two extreme cases of workload distribution and (c) shows one that has a more balanced distribution. MEPS processing can be divided into two subgraphs, which are allocated to the master and slave nodes as illustrated in the figure. The dotted lines on graph edges represent partition cut candidates. The *order* (a parameter relating to the complexity and accuracy of the operation) of the MEPS computation in this presented example is 8. Moreover, the repetitions vector of the associated SDF graph model is shown in Fig. 6(d).

In Fig. 6(a), the slave nodes send raw data directly to the master node without any processing, where all of the MEPS processing is performed. This configuration involves a partition cut that assigns the source actor of the graph to G_s and the remaining actors to G_m. Therefore, the total data transmission (i.e., X_c) for each scheduled iteration from the 5 slave nodes is $5 \cdot 512 = 2560$ tokens.

In Fig. 6(b), each slave node executes a complete MEPS computation, thereby fully processing a captured data frame before communicating to the master node. This is a fully distributed scenario, which minimizes the workload of the master node. In this scenario, each slave node sends 256 tokens to the master node. Thus, the X_c from the 5 slave nodes is $5 \cdot 256 = 1280$.

In Fig. 6(c), on the other hand, the application graph is divided more evenly into two subgraphs. The carefully-constructed partition cut between G_s and G_m

reduces X_c to 9, which results in total slave-to-master X_c of $5 \cdot 9 = 45$ tokens per schedule iteration. The example of Fig. 6 illustrates, in terms of the amount of data tokens to be processed and communicated, trade-offs involved in workload balancing among sensor nodes in a network. Our heuristic algorithm can be used to explore such trade-offs effectively in terms of our energy consumption formulation and the underlying dataflow graph modeling approach.

We have also examined three other DSP applications in our EDP experiments. The first is spectrum computation [Eker et al. 2003], which can be used in converting signals from time domain to frequency domain representations. The second is a seven-level, tree-structured filter bank [Vaidyanathan 1990], which is commonly used in sub-band coding with perfect reconstruction for audio coding applications. Then the third is a distributed speech recognition application, which will be examined in depth in Section 7.2. Results on all of the applications are presented in Section 8.

7.2 Case study: distributed automatic speech recognition

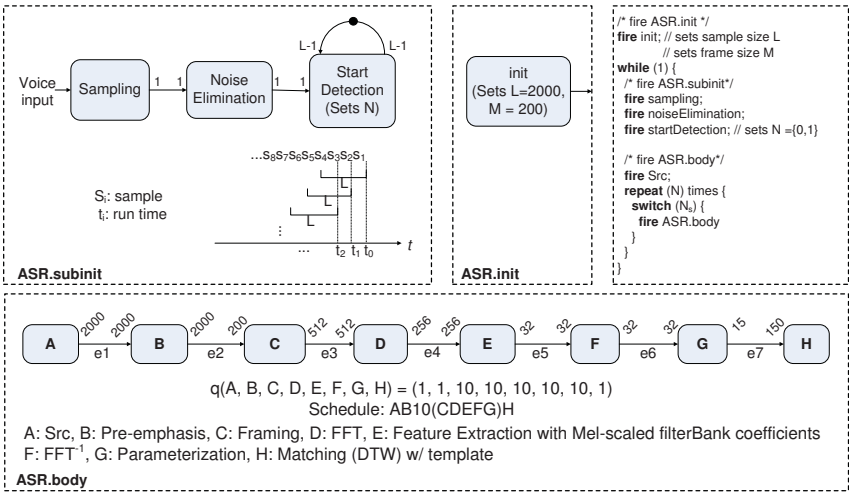
We present a distributed automatic speech recognition (DASR) system as an in-depth case study for our proposed EDP methodology. DASR systems have a variety of potential applications including military surveillance and command and control [Shen et al. 2008].

The signal processing steps that we have employed for the speech data processing and word recognition are based on the approach presented in [Phadke et al. 2004], which is in the context of centralized, single source speech processing. Our development of this application is new in that we apply these speech processing methods in a distributed context, and integrate our EDP methodology into the targeted network to optimize the energy efficiency.

The functional goal of our DASR system is to recognize isolated spoken words through a WSN that is based on a single-cluster, master-slave topology. Thus, voice inputs arrive at the slave nodes, and are recognized as spoken words at the master node, pre-defined word templates are stored for word matching. Each slave node senses acoustic data continuously, and runs an utterance-start-detection scheme for detecting potential speech tokens. Further data processing is performed either on slave nodes or on the master node based on the results of EDP.

We model this DASR application using PSDF so that the workload redistribution scheme can be applied accordingly. The sensing inputs and start detection scheme are modeled in *init* and *subinit* graphs, and thereby, low-overhead, "quasi-static" schedules can be generated for software implementation [Bhattacharya and Bhattacharyya 2001]. Fig. 7 illustrates our PSDF application model and an associated quasi-static schedule. Here, *ASR.init* sets the length of a sliding window for the past $L-1$ samples at each point of time, and the frame size M for each data frame. *ASR.subinit* reads sample inputs and maintains the sliding window during real-time processing.

In our experiments, we set $L = 2000$ and $M = 200$. The corresponding SDF model for speech recognition processing is shown in *ASR.body*. If a valid speech token is detected from an acoustic input stream, a parameter N is configured dynamically to enable speech recognition processing in the body graph, where the body graph is partitioned at compile time in terms of static EDP results. The



Parameter	Value
Num. of frames /word	10
Num. of parameters /word	15
Num. of template words	4
Num. of samples /word	2000
Num. of filter banks	16
Frame length	256 samples
Sample size	8 bits

Fig. 7. PSDF modeling and quasi-static scheduling, and parameters for the experimental DASR system.

partitioned body graph is chosen to be executed in terms of a run-time decision parameter N_s, which represents the number of slave nodes existing in the current system. As we have discussed previously, the value of N_s is determined by the master node and sent as an input parameter to all slave nodes.

Fig. 7 also shows the parameter configurations for our experimental DASR system. We employ a sampling frequency of 8 KHz., and therefore, a 125μs timer is set up for the microcontroller [CC2430 2003] to enable an 8-bit analog-to-digital converter (ADC) for sampling and converting sensed signals from an acoustic sensor. Since sampling and conversion by the selected ADC in [CC2430 2003] takes around 20μs, all sensed samples by the targeted 8 KHz sampling frequency can be captured accurately. Moreover, in order to use limited memory size efficiently, we select words of duration 0.25s or less in our experiments so that the number of samples at 8 KHz is bounded by 2000, where each sample is stored as an 8 bit integer value.

When the DASR system is initialized, slave sensor nodes capture samples from background noise within a certain threshold, and calculate the average. The average noise value is subsequently used to compare signal values that are being monitored and captured. We adopt the start detection scheme described in [Phadke et al. 2004] to detect the start of an utterance in the presence of background noise. Once such an utterance is detected, 2000 consecutive signal samples will be captured and stored in the memory for further processing. That is, speech recognition processing steps are executed as described in the *ASR.body* graph in Fig. 7.

Once the *ASR.body* graph starts to execute, sensed samples are partitioned into several overlapping frames. On each frame, a 256-point fast Fourier transform (FFT) is applied. Each frame-wise FFT result is multiplied by a 16-tap Mel-scaled filter bank to implement the feature extraction function for each frame. The feature extraction function is used to identify the speakers vocal tract in the speech. From this feature extraction step, 15 coefficients are obtained from an inverse discrete cosine transform to represent the parameters a given frame. Then, a dynamic timing warping (DTW) technique [Phadke et al. 2004] is used in the master node to search for a match between the spoken word and one of the template words.

We analyze buffer and latency requirements for this application. Since DTW matching is executed only on the master node, by applying a looped single appearance schedule [Bhattacharyya et al. 1995] to the *ASR.body* graph for implementing the DASR system, the maximum data memory requirement for slave nodes ($buf(G_s)$) from such a schedule is bounded as 2000bytes $\leq buf(G_s) \leq$ 2982bytes. That is, as shown in Fig. 7, if the partition cut is across *e1*, the slave nodes need to store 2000 signal samples and transmit them to the master node. On the other extreme case, if the partition cut is across *e7*, the slave nodes are required to use more memory space to store data — $2000 + 512 + 256 + 32 + 32 + 150 = 2982$ data tokens in this case — but just need to transmit 150 data tokens to the master node.

In order to achieve the goal of real time sensing and processing for this application, we define a minimum duration T_d between consecutive spoken words that the application must be able to handle. In a command-and-control context, for example, such a value would impose a constraint on how fast successive commands could be applied at a given sensor node. The parameter T_d can be translated into a latency constraint on slave node processing and communication. That is, T_d must be larger than the time needed to execute the *ASR.subinit* and *ASR.body* graphs plus the time needed to transmit the required data to the master node.

For example, when the application is implemented on our target platform [CC2430 2003] with a 32 MHz processing speed, and 250 kbps transceiver data rate, and if all sensing and processing tasks except for word matching T_d are handled by the slave nodes, then the minimum allowable "word interval" is approximately 13.675s (0.27s sensing and detection time plus 13.4s processing time and 4.8ms transmission time). On the other hand, if the initial partitioning (C_0) is applied, T_d is constrained below by approximately 0.334s (0.27s sensing and detection time plus 64ms transmission time). Therefore, when an EDP result is applied to the target platform, T_d is bounded by $0.334s \leq T_d \leq 13.675s$. This kind of analysis can be used to constrain real-time specifications for the implemented system.

8. EXPERIMENTAL RESULTS

8.1 Experimental setup

Our experiments are constructed using TDMA-based *homogeneous* and *heterogeneous* wireless sensor networks that have master-slave topologies. Here, by "heterogeneous," we mean that the master and slave nodes, respectively, are equipped with different kinds of data processing components (having, in general, different speeds, supply voltages, etc.) in addition to the microcontrollers that are used for protocol control. Conversely, by "homogeneous," we mean that the master and slave nodes

Texas Instruments/Chipcon CC2430

Name	Value
Clock frequency	32MHz
Radio frequency	2.4GHz
Supply voltage	3V
Process power	36.9mW
Transmit power	80.7mW
Receive power	80.1mW
Radio bit rate	250 kbps
Code memory	128KB
Data memory	8KB

Texas Instruments TMS320C5509A DSP

Name	Value
Clock frequency	200MHz
Core voltage	1.6V
I/O voltage	3.6V
Core supply current (CPU + internal memory access)	120mA

Fig. 8. Hardware specifications for the Texas Instruments CC2430 microprocessor and TMS320C5509A DSP processor.

are equipped with identical data processing components.

For our homogeneous WSN target systems, we use the Texas Instruments CC2430 system-on-chip (SoC) device on all master and slave node platforms for executing processing and communication tasks. This device provides a single-chip, integrated transceiver and embedded microprocessor.

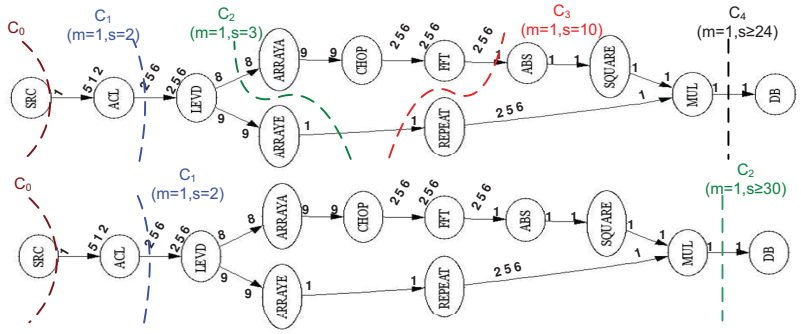
For the heterogeneous systems, we again use the CC2430 device on all slave nodes. However, for the master node, we incorporate, in addition to the CC2430, a Texas Instruments TMS320C5509A [TMS320C5509A 2002] as a dedicated DSP processor. On the master node, we use the transceiver subsystem in the CC2430 for executing communication tasks, and we use the microcontroller in the CC2430 only for protocol control. We use simulators from the IAR Embedded Workbench and Texas Instruments Code Composer Studio to derive task-level timing estimates. Fig. 8 shows hardware specifications for both of the processors that we use in the experiments.

8.2 Simulation results

Fig. 9 and Fig. 10 show experimental results for the targeted DSP applications when distributed across homogeneous and heterogeneous WSNs. The EDP results are simulated to derive partition cuts along with the changes of network size by using the proposed heuristic algorithm on each application graph. In both Fig. 9 and Fig. 10, C_0 denotes the initial partition cut that assigns the source actor to G_s and remaining actors to G_m, and C_i $(i > 0)$ denotes the EDP cut using the proposed heuristic approach in terms of different network sizes (increases in the index i correspond to increases in network size). Note that the initial partitioning corresponds to the conventional configuration of having maximal data processing performed on the master node.

The tables shown in both Fig. 9 and Fig. 10 are profiled task-level timing estimates that represent the execution time of individual actors in each application. Note that in a homogeneous WSN configuration, computational tasks on all nodes are executed by the same kind of processor; therefore, the timing information in columns 2 and 3 of the tables are valid for both master and slave nodes. However, in a heterogeneous WSN configuration, since computational tasks on the master

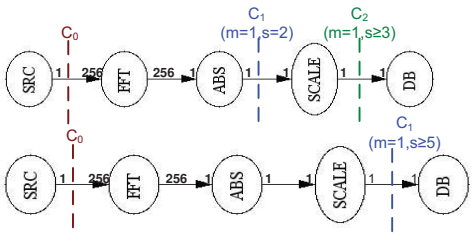
Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	2534.21	7.92E-5	928.75	4.64E-6
ACL	52830479	1.65	24779839	1.24E-1
LEVD	17977524	0.56	1181	5.91E-6
ARRAYA	2186	6.83E-5	323	1.62E-6
ARRAYE	325	1.01E-5	14	7.0E-8
REPEAT	28093	8.77E-4	5899	2.95E-5
CHOP	39672	1.24E-3	9377	4.69E-5
FFT	7479003	0.23	962008	4.81E-5
ABS	1428.24	4.46E-5	5245	2.6E-5
SQUARE	478.85	1.5E-5	181	9.05E-7
MUL	618.29	1.93E-5	580	2.9E-6
DB	722.25	2.26E-5	5310	2.66E-5



q(SRC, ACL, LEVD, ARRAYA, ARRAYE, REPEAT, CHOP, FFT, ABS, SQUARE, MUL, DB) = [512, 1, 1, 1, 1, 1, 1, 256, 256, 256, 256]

(a) EDP results for the MEPS computation.

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	2532.99	7.92E-5	926	9.74E-6
FFT	8163758	0.26	962008	4.81E-3
ABS	1409	4.4E-5	5325	2.66E-5
SCALE	1606.3	5.02E-5	5421	2.7E-5
DB	707	2.21E-5	5252	2.63E-5

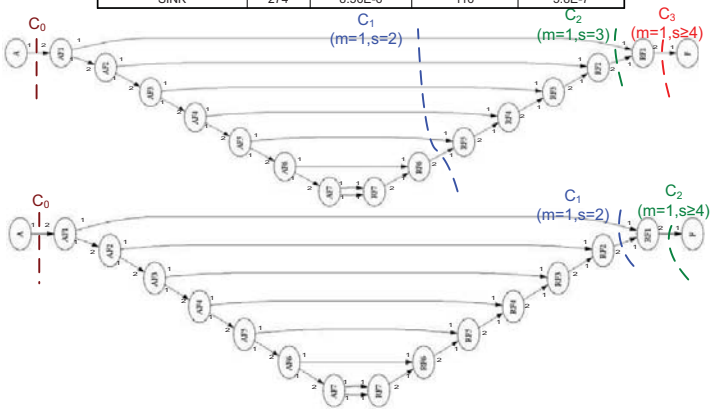


q(SRC, FFT, ABS, SCALE, DB) = [256, 1, 256, 256, 256]

(b) EDP results for the spectrum computation.

Fig. 9. EDP results for experimental DSP applications across homogeneous and a heterogeneous WSNs with various network size settings (m = # of master nodes, s = # of slave nodes).

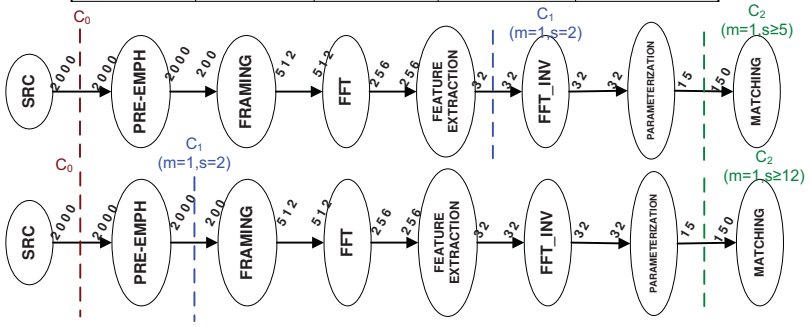
Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
SRC	1398	4.37E-5	804	4.02E-6
FilterAnalyzer (AF)	40868	1.28E-3	8625	4.31E-5
FilterReconstructor (RF)	56317	1.76E-3	11243	5.62E-5
SINK	274	8.56E-6	116	5.8E-7



q(A, AF1, AF2, AF3, AF4, AF5, AF6, AF7, RF7, RF6, RF5, RF4, RF3, RF2, RF1, F) = [128, 64, 32, 16, 8, 4, 2, 1, 1, 2, 4, 8, 16, 32, 64, 128]

(a) EDP results for the seven-level tree-structured filter bank.

Actors	Average execution cycle on CC2430	Average execution time (sec.) on CC2430 with 32MHz CLK	Average execution cycle on TMS320C5509A	Average execution time (sec.) on TMS320C5509A with 200MHz CLK
startDetection	364	1.14E-5	33	1.65E-7
SRC	2496	7.8E-5	1025	5.13E-6
Pre-emphasis	2372682	0.074	100150	5.0E-5
Framing	17285507	0.54	105120	5.26E-4
FFT	87028273	2.72	45053953	0.225
featureExtraction	218199980	6.82	10896790	0.54E-1
invFFT	87028273	2.72	45053953	0.225
Parameterization	20096	6.28E-4	3540	1.77E-5
Matching	126874305	3.96	57580815	0.288



q(SRC, PRE-EMPH, FRAMING, FFT, FEATURE-EXT, FFT_INV, PARAM, MATCHING) = [1, 1, 10, 10, 10, 10, 10, 1]

(b) EDP results for the experimental DASR system.

Fig. 10. EDP results for experimental DSP applications across homogeneous and a heterogeneous WSNs with various network size settings (m = # of master nodes, s = # of slave nodes).

node and the slave nodes are executed by different types of processors, the timing information in columns 4 and 5 of the tables shows each actors execution time on the master node, and the corresponding timing information on the slave nodes is provided from columns 2 and 3.

The EDP simulation results are shown graphically in both Fig. 9 and Fig. 10 based on the derived partition cuts. Here, the top and bottom figures represent the associated EDP results when the target application is applied to a homogeneous and a heterogeneous WSN system, respectively. The EDP results (C_i) are compared with the initial partitioning (C_0) in terms of various network size settings for each WSN configuration. As shown in the figure, partition cuts are shifted gradually from the G_s side to the G_m side of each application graph as the network size is increased. This is because a more balanced workload distribution is found between the master and slave nodes on the chosen applications. As more slave nodes are added, more of the data processing burden should generally be assigned to the slave nodes since the master node, as the central recipient of communication from all slave nodes, needs to take care of more computational requests.

8.3 Energy cost comparison with the workload redistribution scheme

We demonstrate the associated system energy cost, E_{sys} (i.e., (9)), for the chosen applications in Fig. 11 according to the simulated EDP results obtained from Fig. 9 and Fig. 10. For more detailed analysis, E_{sys} can be applied to (10) along with the appropriate battery capacity values so that E_{sys} can be converted to an estimate of network lifetime.

We compare the system energy cost with and without our the workload redistribution scheme for all of the experimental DSP applications. We initialize the targeted WSN systems with 1 master node and 2 slaves. Then additional slave nodes are added into the systems one at a time until a total of 10 slave nodes is reached for each system.

We demonstrate a comparison in terms of system energy cost (E_{sys}) in Fig. 11 for each experimental application. This comparison is based on a homogeneous WSN configuration. Here, "QS EDP" stands for the results from quasi-static EDP, and each partition cut (C_i, where $i \geq 1$) is derived by using our proposed heuristic algorithm with an appropriate network size setting as shown in Fig. 9 and Fig. 10. In Fig. 11, we observe that the system energy cost is reduced consistently with the derived partitioning results for balancing the workload distribution between the master node and the slave nodes. Our approach obtains at least a 50% improvement in energy cost compared to the conventional approach of having maximal data processing performed on the master node. Moreover, as the network size changes, the EDP result is adapted automatically to solutions that are better matched to the new scenarios.

Note that as discussed in Section 6.2, the exhaustive search algorithm always finds an optimal solution, whereas a solution returned by the heuristic algorithm may be suboptimal. We have used the exhaustive search approach to find EDP solutions for all of the applications that we experimented with. In these experiments, we found that the result of exhaustive search was 5% better on average; however, as we have shown in Fig. 11, our heuristic approach still improves energy consumption significantly compared to the conventional master/slave processing approach, and

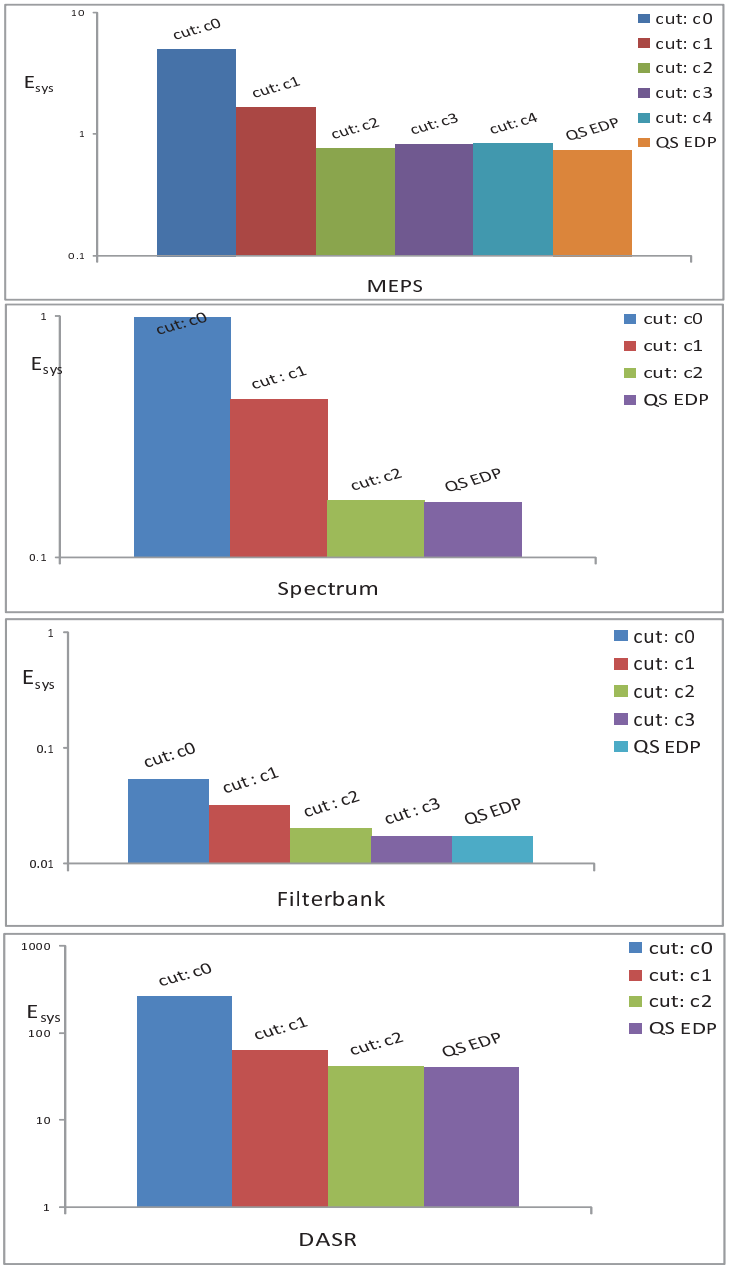


Fig. 11. Simulation results involving energy cost for the experimental applications.

also achieves results that are close in quality to or equivalent to the optimal results obtained from exhaustive search.

9. CONCLUSION

This paper presents a novel algorithm, and associated design methodology for distributing DSP applications across master/slave wireless sensor network (WSN) systems in an energy-efficient fashion. Our methodology integrates high-level application modeling, and task- and network-level energy and latency modeling to comprehensively optimize system performance. We have discussed the use of efficient, integrated techniques for modeling and analysis to represent different DSP applications, formulate WSN topology and protocol requirements, formulate the energy-driven partitioning (EDP) problem, integrate EDP solutions with quasi-static scheduling, and develop an efficient heuristic algorithm for finding EDP results that maximize the network lifetime. Results on synthetic benchmarks and on practical applications demonstrate the utility of our proposed methods. Our experimental results demonstrate that our approach runs efficiently, and improves conventional partitioning results significantly (by least 50% of the energy cost). As applications become more complicated, the proposed methodology becomes even more useful.

REFERENCES

Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. 2002. A survey on sensor networks. *IEEE Comm. 40,* 8, 102–114.

Bhattacharya, B. and Bhattacharyya, S. S. 2001. Parameterized dataflow modeling for dsp systems. *IEEE Trans. Signal Processing 49,* 10, 2408–2421.

Bhattacharyya, S. S., Buck, J. T., Ha, S., and Lee, E. A. 1995. Generating compact code from dataflow specifications of multirate signal processing algorithms. *IEEE Trans Circuits and Systems — I: Fundamental Theory and Applications* 3, 138–150.

Bhattacharyya, S. S., Murthy, P. K., and Lee, E. A. 1999. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology 21,* 2, 151–166.

Calhoun, B. H., Daly, D. C., Verma, N., Finchelstein, D. F., Wentzloff, D. D., Wang, A., Cho, S., and Chandrakasan, A. P. 1995. Design considerations for ultra-low energy wireless microsensor nodes. *IEEE Trans. Computers 54,* 6, 720–740.

CC2430, T. I. 2003. *CC2430 Data Sheet: SWRS036F.* Dallas, TX.

Chatterjea, S., Nieberg, T., Meratnia, N., and Havinga, P. 2008. A distributed and self-organizing scheduling algorithm for energy-efficient data aggregation in wireless sensor networks. *ACM Trans. Sensor Networks 4,* 4 (Aug.), 41.

Chen, Y. and Zhao, Q. 2007. An integrated approach to energy-aware medium access for wireless sensor networks. *IEEE Trans. Signal Processing 55,* 7, 3429–3444.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. 2001. *Introduction to Algorithms.* The MIT Press.

Eker, J., Janneck, J. W., Lee, E. A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. 2003. Taming heterogeneity - the ptolemy approach. In *Proceedings of the IEEE.* 127–114.

Fiduccia, C. M. and Mattheyses, R. M. 1982. A linear-time heuristics for improving network partitions. In *Proceedings of the 19th Design Automation Conference.* 175–181.

Ganesan, D., Cerpa, A., Ye, W., Yu, Y., Zhao, J., and Estrin, D. 2004. Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing 64,* 7, 799–814.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability.* W. H. Freeman and Co., New York.

Hoang, A. T. and Motani, M. 2008. Collaborative broadcasting and compression in cluster-based wireless sensor networks. *ACM Trans. Sensor Networks 3,* 3 (Aug.), 17.

Hsu, C., Ko, M., and Bhattacharyya, S. S. 2005. Software synthesis from the dataflow interchange format. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*. Dallas, TX, 37–49.

Kalavade, A. and Suhrahmanyam, P. A. 1997. Hardware/software partitioning for multi-function systems. In *Proceedings of the International Conference on Computer Aided Design*. San Jose, CA, 516–521.

Kernighan, W. and Lin, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 291–307.

Ko, D., Shen, C., Bhattacharyya, S. S., and Goldsman, N. 2006. Energy-driven partitioning of signal processing algorithms in sensor networks. In *International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, Lecture Notes in Computer Science 4017*. Springer, 142–154.

Kumar, R., Tsiatsis, V., and Srivastava, M. B. 2003. Computation hierarchy for in-network processing. In *Proceedings of the ACM International Conference on Wireless Sensor Networks and Applications*. San Diego, CA, 68–77.

Kuorilehto, M., Hannikainen, M., and Hamalainen, T. D. 2005. A survey of application distribution in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 774–788.

Lee, E. A. and Messerschmitt, D. G. 1987. Synchronous dataflow. *Proceedings of the IEEE 75,* 9, 1235–1245.

Li, X., Song, W., and Wang, Y. 2006. Localized topology control for heterogeneous wireless sensor networks. *ACM Trans. Sensor Networks 2,* 1 (Feb.), 129–153.

Lindsey, S., Raghavendra, C., and Sivalingam, K. 2002. Data gathering in sensor networks using the energy delay metric. *IEEE Trans. Parallel and Distributed Systems 13,* 9, 924–935.

Park, J. and Sahni, S. 2006. An online heuristic for maximum lifetime routing in wireless sensor networks. *IEEE Trans. on Computers 55,* 8, 1048–1056.

Phadke, S., Limaye, R., Verma, S., and Subramanian, K. 2004. On design and implementation of an embedded automatic speech recognition system. In *Proceedings of the 17th International Conference on VLSI Design*. 127–132.

Romer, K. and Mattern, F. 2004. The design space of wireless sensor networks. *IEEE Wireless Communications 11,* 6, 54–61.

Shen, C., Plishker, W., and Bhattacharyya, S. S. 2008. Design and optimization of a distributed, embedded speech recognition system. In *Proceedings of the 16th International Workshop on Parallel and Distributed Real-Time Systems*. 6.

Shen, C., Plishker, W., Bhattacharyya, S. S., and Goldsman, N. 2007. An energy-driven design methodology for distributing dsp applications across wireless sensor networks. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*. 214–223.

Shih, E., Cho, S., Ickes, N., Min, R., Sinha, A., Wang, A., and Chandrakasan, A. 2001. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*. 272–287.

Tang, X. and Xu, J. 2008. Adaptive data collection strategies for lifetime-constrained wireless sensor networks. *IEEE Trans. Parallel and Distributed Systems 19,* 6, 721–734.

TMS320C5509A, T. I. 2002. *TMS320C5509A Data Sheet: SPRS205I*. Dallas, TX.

Vaidyanathan, P. P. 1990. Adaptive data collection strategies for lifetime-constrained wireless sensor networks. *Proceedings of the IEEE 78,* 1, 56–93.

Wang, A. and Chandrakasan, A. 2001. Energy-effiecient dsps for wireless sensor networks. *IEEE Signal Processing Magazine*, 68–78.

Zhang, J., Liu, Q., and Zhong, Y. 2008. A tire pressure monitoring system based on wireless sensor networks technology. In *Proceedings of the International Conference on MultiMedia and Information Technology*. Los Alamitos, CA, 602–605.