In Proceedings of the IEEE Real-Time Systems Symposium, Tucson, Arizona, December 2007.

An Energy-driven Design Methodology for Distributing DSP Applications across Wireless Sensor Networks

Chung-Ching Shen, William Plishker, Shuvra S. Bhattacharyya, and Neil Goldsman
Dept. of Electrical and Computer Engineering, and Institute for Advanced Computer Studies
University of Maryland at College Park, USA
{*ccshen, plishker, ssb, neil*}*@umd.edu*

Abstract

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 embedded sensor nodes becomes critical.

Digital signal processing (DSP) applications typically require intensive data processing operations. They are difficult to apply directly in resource-limited WSNs because their operational complexity can strongly influence the network lifetime.

In this paper, we present a design methodology for modeling and implementing DSP applications applied to wireless sensor networks. This methodology explores efficient modeling techniques for DSP applications, including acoustic sensing and data processing; derives formulations of energy-driven partitioning for distributing such applications across wireless sensor networks; and develops efficient heuristic algorithms for finding partitioning results that maximize the network lifetime. A case study involving a speech recognition system demonstrates the capabilities of our proposed methodology.

1. Introduction and related work

In a hierarchical wireless sensor network [11], sensor nodes are clustered into groups, and their roles are divided into 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, the targeted wireless sensor network structures are such single-hop, master-slave topologies. Such static configurations of network structure have simple routing characteristics, and protocol demands, and allow designers to optimize effectively for network lifetime in terms of energy consumption, as well as for scalability through hierarchical organization.

Digital signal processing (DSP) applications are often relevant to processing sensor data, and usually require intensive computation. The behavior of many DSP applications can be characterized with regular computation patterns and modeled efficiently through dataflow graphs. By analyzing a well-designed dataflow model of an application, operational efficiency can be estimated and optimized accordingly (e.g., see [2, 12]).

In a typical WSN configuration, resource-limited slave nodes are unable to handle computationally-intensive tasks due to lack of hardware and software support. In such a case, microcontrollers in slave nodes can perform relatively light-weight computations. With advances in integrated circuit technology, slave nodes can be equipped with increasing amounts of computational resources, such as digital signal processor subsystems. Then the microcontrollers can perform protocol and control tasks, while the DSP processors perform more intensive computational tasks.

The energy consumption of the nodes in a wireless sensor network must be carefully optimized to increase network lifetime. Computation and communication tasks that sensor nodes execute affect major energy consumed. Especially, communication tasks on the transceiver dominate overall power consumption on a sensor node [5]. Thus, the amount of data to be trans-

mitted across the wireless channel should be minimized due to the reduction of the turn-on time of the transceivers. For distributing DSP computations to the WSN, this reduction is significant since DSP applications usually process a large amount of data for their tasks. In this paper, we present an algorithm that addresses this objective for DSP applications that are modeled as dataflow graphs. Specifically, our algorithm finds an efficient trade-off between the workloads of computation and communication tasks in both master and slave nodes.

Many useful approaches have been studied previously to reduce the energy consumption of sensor nodes. In [15], Singh discusses system-level trade-offs related to energy costs of WSN technologies. Shih [14] distributes the FFT function over a master node and associated slave nodes to reduce energy consumption. Kumar [11] explores energy and latency trade-offs by considering different computational capabilities for master and slave nodes.

Wang [16] develops an approach that partitions applications between master and slave nodes, and also applies dynamic voltage scaling to further reduce power consumption. In contrast to Wang's approach, 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 [8].

Ko [10] introduces the general approach of reducing data traffic across WSN nodes by determining and exploiting the lowest data token delivery points within an application dataflow graph. This paper builds on the approach and problem formulations introduced by Ko, and develops an efficient heuristic method that is more efficient and scalable compared to the exhaustive search approach employed by Ko. We also go beyond the developments of [10] in this paper by addressing both homogeneous and heterogeneous network organizations; taking into account energy consumption in more detail as well as real time latency constraints, based on measured and simulated task profiles; and as mentioned above, by integrating our partitioning methods into the DIF package.

2. Design flow overview

Figure 1 presents an overall design flow for our proposed methodology. Our approach in this paper involves distributing dataflow representations of DSP applications across WSNs. To specify and analyze these dataflow representations we use the dataflow interchange format (DIF) and the associated DIF package [8]. DIF is a unified textual language for expressing different kinds of dataflow semantics. The DIF package is a software package that provides representations and utilities for analyzing dataflow graphs that are captured using DIF.

In the semi-automated design flow demonstrated in this paper, we use DIF to specify the behavior of DSP applications. Moreover, we assign power and timing estimation results to components of each dataflow graph. These estimation results 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. These algorithms form the core contribution of this paper.

In the current form in which our optimization approaches are implemented, we map the automatically-generated application partitioning results onto the targeted WSN by hand (i.e., by accordingly hand-partitioning C code for the application subsystems onto the different sensor nodes), and execute the resulting WSN configurations to measure the results.

3. Dataflow modeling approach

Dataflow models of computation are widely used for modeling DSP applications (e.g., see [2]). Synchronous dataflow (SDF) [12] is a restricted form of dataflow that is useful for an important class of applications, and is used in a variety of commercial design tools. In SDF, the number of data values (*tokens*) produced and consumed by each graph vertex (*actor*) is fixed and known at compile time. As a result of this restriction, graphs can be scheduled statically based on the so-called *repetition vector:* q, which is a vector that is indexed by the actors in the graph, and gives the

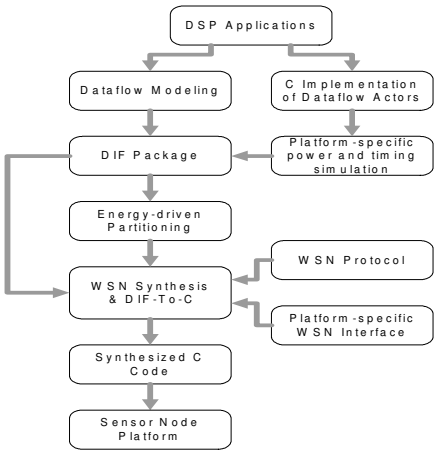


Figure 1. Overall design flow.

number of times that each actor needs to be invoked in a static (periodic) schedule for the graph. For more details on the repetition vector and other fundamental SDF features, we refer readers to [2] and [12].

In this paper, DSP applications that involve purely static dataflow behavior are modeled using SDF. 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 real time sensing activities. For example, in some applications, the analog data input must be controlled so that the following digital processing sections are executed only when needed. Therefore, 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 [1]. PSDF expresses a wide range of dynamic dataflow behaviors while preserving much of the useful analysis and synthesis capability of SDF [1]. 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.

4. Energy-driven partitioning

4.1. Definitions

Communication between WSN nodes significantly impacts system performance and must be carefully evaluated in terms of energy consumption and latency. In our approach, we adopt a light-weight time synchronization scheme for communication. This synchronization scheme is described in [7]. Because the overhead associated with this scheme is relatively small, as demonstrated in [7], we ignore synchronization for our analysis of energy consumption in this paper, and we focus only on the energy consumption for processing and communicating data tokens. Our experiments demonstrate that even though we make this simplification, our results lead to significant improvements in energy savings.

To formalize the problem, an application dataflow graph, $G = (V, E)$, is partitioned into two non-empty subgraphs $G_m = (V_m, E_m)$ and $G_s = (V_s, E_s)$, which respectively represent the distribution of the partitioned application graph onto master and slave nodes. The corresponding partition cut, E_c, is then defined to be the set of edges that "cross the partition":

$$E_c = \{(u, v) \in E | ((u \in V_s) \text{ and } (v \in V_m))\}.$$

Given a partition of the dataflow graph into two parts, the total number of tokens that must be transferred across the partition can be obtained by summing up the token transfer volumes of the edges that cross the partition cut.

In our targeted network structure, the network topology is assumed to be static — that is, nodes are not movable and the network size is known beforehand. 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 must complete a predefined amount of computation and transmit/receive processed data as needed based on the computation.

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.

To estimate the computational energy that each actor in G consumes, we distinguish between the cases of having *homogeneous* and *heterogeneous* processing platforms across the master-slave hierarchy network. Here, by "heterogeneous," we mean that the master and slave nodes, respectively, are equipped with different data processing components (having, in general, different speeds, supply voltages, etc.) in addition to the microcontrollers that are used for protocol control. In the heterogeneous case, 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 whenever actor v executes on the master node (on a slave node). For the homogeneous case, we use a single pair $(P(v), t(v))$ of corresponding attributes for each actor v. This pair gives the power

consumption and execution time of v regardless of whether v is assigned to the master node or to a slave node.

We also characterize edges in G with power consumption and execution time values. Specifically (using a minor abuse of notation), each edge e is characterized by two pairs of attributes $(P_m(e), t_m(e))$ and $(P_s(e), t_s(e))$, which represent the local (intra-node) power consumptions values and latencies for the master and slave nodes, respectively, 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 on our application partitioning model, edges in E_c corresponds to points in the dataflow graph where tokens are transmitted and received through the wireless channel.

4.2. Energy formulation on data processing and communicating

The total number of data values transferred across an SDF edge e in a minimal scheduled iteration of an SDF graph G can be expressed [12] as

$$\tau(e) = q(src(e)) \cdot prd(e) = q(snk(e)) \cdot cns(e), \quad (1)$$

where $src(x)$ represents the source actor of a dataflow edge x; $prd(y)$ represents the number of tokens produced onto SDF edge y by each execution of $src(y)$; $snk(x)$ represents the sink actor of a dataflow edge x; and $cns(y)$ represents the number of tokens consumed from SDF edge y by each execution of $snk(y)$. Based on this, the total number of tokens X_c that cross the partition within each scheduled iteration of an SDF application graph can be expressed as

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

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

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

and

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

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

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

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

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

The total energy consumption on a single node for one scheduled iteration (i.e., the energy per scheduled iteration period) is defined as 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

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

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

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

Note that $E_{idle}(m)$ can be small since 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

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

To compare the network lifetime according to the selection of different partition cuts, we define the network lifetime as the time that the first node in the network runs out of energy. Thus, our network lifetime estimation is formulated as

$$min\{T_{lifetime}(master), T_{lifetime}(slaves)\}, \text{ where} \quad (10)$$

$$E_{source} = T_{lifetime} \cdot E_{iter} \cdot R_{avg};$$

R_{avg} is the average rate at which scheduled iterations are executed; and E_{source} is the total energy stored in a 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.

4.3. Constraints

Local and global latency constraints are also considered in the partitioning problem formulated in this paper. Here, 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

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

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,

$$L_g = L_l(slaves) + N_s \cdot L_l(master) + N_s \cdot X_c \cdot D \,. \qquad (11)$$

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 Eq. 11, 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

$$N_s \cdot L_{sync} + L_g \le L_p \,. \qquad (12)$$

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 Eq. 12, 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, Eq. 12 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 Eq. 12.

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 Eq. 11 and 12, we can find the minimum requirement of t_p as $2 \cdot L_l(slaves)/N_s$ if $N_s = N_p$. Generally, $N_p \ge N_s$, 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

$$\forall l \in E_c, src(l) \in G_s \Rightarrow preds(src(l)) \in G_s \wedge snk(l) \in G_m \Rightarrow succs(snk(l)) \in G_m \quad , \qquad (13)$$

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 finding 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

$$max\{min[T_{lifetime}(master), T_{lifetime}(slaves)]\} \,, \qquad (14)$$

subject to 1) $G_m \neq \varnothing$; 2) $G_s \neq \varnothing$; 3) $N_s \cdot L_{sync} + L_g \le L_p$; and 4) E_c is a valid partition cut, based on Eq. 13.

4.4. Proposed partitioning algorithm

Classical graph partitioning problems are similar to the energy-driven partitioning problem (EDP) devel-

oped in this paper. These classical graph partitioning problems, which are discussed in [6], include "PARTITION INTO ISOMORPHIC SUBGRAPHS", "BALANCED COMPLETE BIPARITE SUBGRAPH", "GRAPH PARTITIONING", and "ACYCLIC PARTITION". EDP is somewhat different, however, from the related classical problems due to the energy and latency considerations in EDP, and due to the additional restrictions associated with Eq. 13. The graph partitioning problem is NP-complete, and although EDP is somewhat different from graph partitioning, EDP is also NP-complete. The NP-hardness of EDP can be established with a reduction from the well-known subset sum problem [6]. We omit the details here due to space limitations.

A variety of heuristic algorithms for graph partitioning have been developed. The *Kernighan-Lin* (K-L) [9] and *Fiduccia-Mattheyses* (F-M) [4] algorithms are both popular heuristic algorithms for graph partitioning. These algorithms involve incrementally exchanging vertices across the partition cut if the exchanges improve 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 CT of a partitioning result as

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

We seek to minimize CT so that the corresponding maximized network lifetime can be obtained. Based on this, we formulate the $Gain$ function for moving actor v from one side of a given cut to the other. Intuitively, $Gain(v)$ 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. $Gain$ values can be computed and updated efficiently based on the formulations developed earlier in this 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. 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 $Gain(v)$, 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. $D(v)$ is expressed as

$$D(v) = \{gain(G_s), gain(G_m)\} =$$
$$\{E(v) + gain(cut, t), N_s \cdot (-E(v) + gain(cut, r))\}, v \in G_s$$
or
$$\{-E(v) + gain(cut, t), N_s \cdot (E(v) + gain(cut, r))\}, v \in G_m$$

where

$$gain(cut, t) = X_c(v) \cdot P_t \cdot t_c - \overline{X_c}(v) \cdot P_t \cdot t_c ,$$

$$gain(cut, r) = X_c(v) \cdot P_r \cdot t_c - \overline{X_c}(v) \cdot P_r \cdot t_c ,$$

and

$$X_c(v) = \sum_{e \in cutedges(v)} prd(e) \cdot q(src(e)) ,$$

$$\overline{X_c}(v) = \sum_{e \in noncutedges(v)} prd(e) \cdot q(src(e)) .$$

Here, $cutedges(v)$ is the set of edges of vertex v that cross the partition cut (thus, $cutedges(v) \subseteq E_c$), and $noncutedges(v)$ is the set of edges of vertex v that do not cross the cut. Therefore, $X_c(v)$ denotes the total number of data tokens to be transmitted and received by v due to the existing partition cut, and $\overline{X_c}(v)$ denotes the 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, $gain(cut, t)$ $(gain(cut, r))$ of v denotes the gain of potential communication energy variation for transmitting (receiving) data tokens respectively across the partition cut if vertex v is moved across the existing cut.

Based on this gain pair function for each candidate vertex v, we derive $Gain(v)$ by:

$$Gain(v) = CT -$$
$$max\{\{E(s) + E(t), N_s(E(m) + E(r))\} - D(v)\}$$
$$= CT -$$
$$max\{E(s) + E(t) - gain(G_s), N_s(E(m) + E(r)) - gain(G_m)\}$$

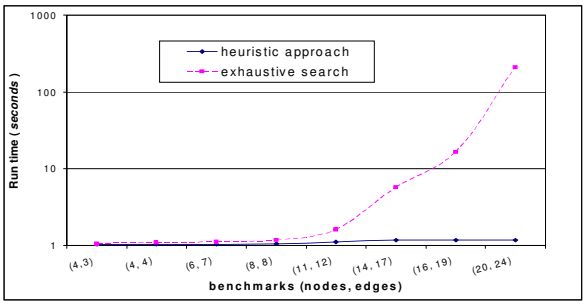
Note that $Gain(v)$ can be positive- or negative-valued. A positive value for $Gain(v)$ means that there is an improvement in the cost function CT if v is moved across the existing cut. On the other hand, a negative value for $Gain(v)$ represents a degradation of CT if v is moved.

Based on this $Gain$ formulation, Figure 2 shows a pseudocode specification of our heuristic approach for EDP. Figure 3 provides performance comparisons between our proposed heuristic and exhaustive search for EDP. In Figure 3, we created several randomly-generated synthetic SDF graphs as the input set and assumed that all graphs are to be applied on a homogeneous network. Figure 3(a) reports the optimal results for both approaches and 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 Eq. 14 are veri-

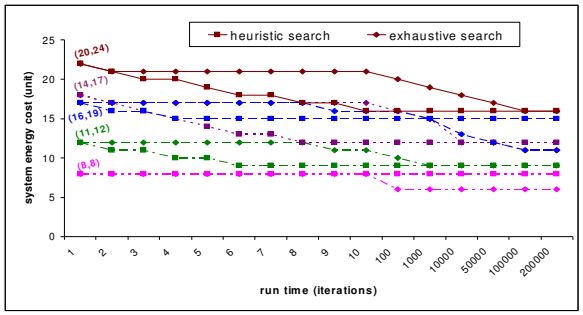
fied for each candidate partitioning to filter out invalid results.

Input: $G = (V, E)$
Output: $G_m = (V_m, E_m)$, $G_s = (V_s, E_s)$, and E_c.
begin
  Create an initial partition.
  Compute $CT(G_m, G_s, E_c)$ for the initial partition.
do
  for (each $v \in V$) **do**
    if v violates constraints in Eq. 14
      Mark v to be locked.
    else
      Mark v to be unlocked, and Compute $Gain(v)$ for v.
  while (unlocked nodes exist)
    Find one unlocked node v with maximum $Gain(v)$.
    Add $(v, Gain(v))$ to an ordered list, $order\_L$.
    Lock v, and Update $Gain$ for all unlocked nodes.
    Select the first k nodes that maximizes $\sum_v Gain(v)$ from $order\_L$.
    if $\sum_v Gain(v) > 0$
      for (each $v \in$ *selected k nodes*) **do**
        Update G_m and G_s based on switching v.
        Update $CT(G_m, G_s, E_c) = CT_{old}(G_m, G_s, E_c) - Gain(v)$.
while $\sum_v Gain(v) > 0$
end

Figure 2. Heuristic approach for the EDP.



(a) Run time comparison based on the complexity of synthetic SDF graphs.



(b) Cost versus run time comparison for selected synthetic SDF graphs.

Figure 3. Performance comparison for EDP schemes.

Figure 3(b) shows a comparison of E_{sys} versus run time for successive algorithm iterations ("run time testing iterations") on several synthetic graphs. Here, the total number of run time testing iterations represents the maximum number of allowable node 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 edges, and we also normalize all energy-related attributes (i.e., the power and time estimates) to 1 for all actors and edges. We observe from Figure 3(b) that our heuristic search algorithm converges significantly faster than exhaustive search for each synthetic graph. Note here that if both search algorithms converge on the same point, then both algorithms 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. Due to space limitations, we omit details on analyzing the algorithm step by step.

5. Experiments

Our experiments are constructed using homogeneous and heterogeneous wireless sensor networks that have master-slave topologies. Each experiment includes one master node with varying amounts of slave nodes. In order to have each experiment satisfying latency constraints appropriately, we set parameters such that

$$N_p = N_s \text{ and } t_p = max\{L_l(slaves), L_m(master)\}$$

for the TDMA-based protocol setup. Thus, all computation and communication operations for each scheduled iteration can be done within a given TDMA time frame.

For our homogeneous WSN target systems, we use the Texas Instruments/Chipcon 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 microcontroller. In addition, we equip each sensing platform with an acoustic sensor.

For the heterogeneous target 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 device 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 simula-

tors from the IAR Embedded Workbench and Texas Instruments Code Composer Studio to derive task-level timing estimates. Due to space limitations, the timing estimation results are not reported here.

5.1. A case study

We have developed a distributed WSN version of Phadke's embedded automatic speech recognition system [13]. We apply this distributed automatic speech recognition (DASR) system here as a case study for EDP. DASR systems have a variety of potential applications including battlefield monitoring, border control, or commercial surveillance.

The functional goal of our DASR system is to recognize isolated spoken words through a WSN that is based on a master-slave topology. Thus, voice inputs arrive at the slave nodes and are recognized as spoken words at the master node. The recognition is performed using pre-defined word templates that are stored at the master node. Each slave node senses acoustic data continuously and runs start/end detection schemes 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 the real-time sensing behavior on slave nodes using PSDF. The sensing inputs and start detection schemes are modeled in *init* and *subinit* graphs, and thereby, low-overhead, "quasi-static" schedules can be generated for hand-coded implementation or software synthesis [1]. Figure 4 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 *ASR.subinit* reads sample inputs and maintains such a window during real time. If a valid speech token is detected from the acoustic input stream, a dynamic parameter N is configured to enable speech recognition processing in the body graph *ASR.body* based on the number of

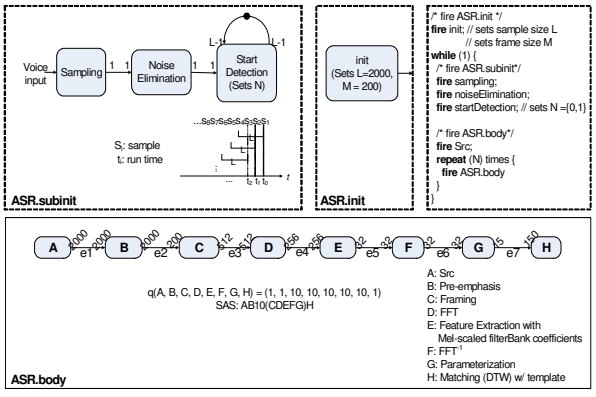


Figure 4. PSDF modeling and quasi-static scheduling for the experimental DASR system.

speech tokens that should be processed in this recognition step. If $N = 0$, then a speech segment involving L samples is not detected, and further processing of the current window of samples will not be undertaken.

Figure 5 lists major parameters for configuring our experimental DASR system as well as relevant specifications for evaluating power consumption on our targeted hardware platforms. In this system, the sampling frequency is 8 KHz, and each raw sample is stored as 8 bits. In our current design of the system, our experimental platforms have limited memory size, and therefore we shorten the word duration to 0.25 seconds so that the number of samples at 8KHz is 2000. That is, once the start detection actor is executed and the beginning of an utterance is detected, 2000 consecutive samples will be sensed and stored for further processing.

Figure 6 shows energy-driven partitioning results from our DASR system using both our proposed heuristic and our implemented exhaustive search method. In Figure 6, E_{sys} values are compared among the initial partitioning (C_0) (i.e., the conventional configuration of having maximal data processing performed on the master node), the heuristic EDP approach (C_{heu}), and the exhaustive EDP approach (C_{opt}) for different network sizes. To obtain the network lifetime, the system energy cost can be applied to Eq. 10 along with the appropriate battery capacity values. As shown in Figure 6, the system energy cost is increased as the network size is increased. An appropriate partitioning result is helpful for balancing the workload between the master node and the slave nodes and reducing the system energy cost. Our approach obtains at least 50% improvement of the energy cost compared to the initial partitions.

In order to achieve the goal of real time sensing and processing on this application, the separate duration T_d of each spoken word is assumed to satisfy the minimum latency required on each slave node. That is, T_d must be larger than the time needed to execute the *ASR.sub-*

CC2430	Value
clock freq.	32 MHz
radio freq.	2.4 GHz
supply voltage	3 V
computation power	36.9 mW
transmit power	80.7 mW
receive power	80.1 mW
data rate	250 kbps

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

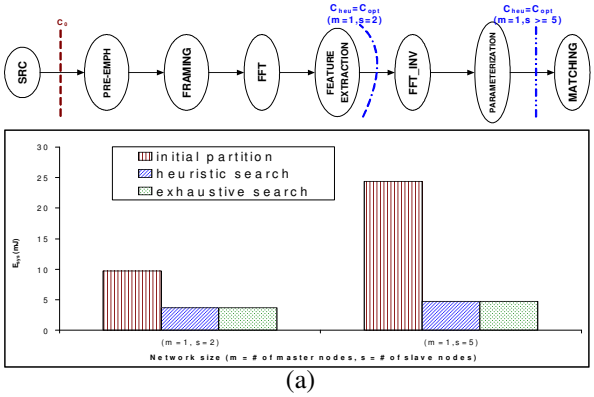
TMS320C5509A	Value
Clock frequency	200MHz
Core voltage	1.6V
I/O voltage	3.6V
Core supply current (CPU + internal memory access)	120mA

Figure 5. System parameters and hardware specification for the experimental DASR system and the experimental DSP platform.

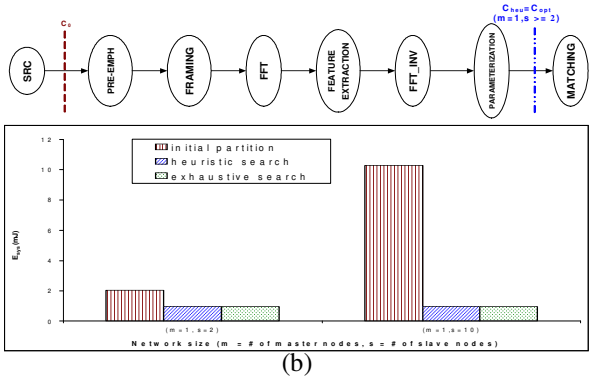
init and *ASR.body* graphs plus the time needed to transmit the required data to the master node. On the CC2430 platform with lower processing speed (e.g. 32 MHz) and 250 kbps transmitter data rate, T_d is approximately 13.67s (13.67s for sensing and processing plus 4.8ms operational time on the transmitter) if all sensing and processing tasks are handled by slave nodes except for the matching task. On the other hand, T_d is approximately 0.33s (0.27s sensing and processing time plus 64ms operational time on the transmitter) if the initial partitioning (C_0) is applied. That is, when an EDP result is applied to the targeted CC2430 platform, T_d is bounded by $0.334s \le T_d \le 13.767s$. This analysis can be used to constrain real-time specifications for the implemented system.

5.2. Other application examples

We also demonstrate our partitioning methodology by distributing two other DSP applications — a *spectrum* computation and a *maximum entropy spectrum* (MES) application, which can be used to derive frequency domain estimates of sensed data signals throughout a WSN (see Figure 7). In experimenting with these applications, we started with model-based reference implementations of the applications that were



(a)



(b)

Figure 6. Energy-driven partitions and E_{sys} comparison for the (a) homogeneous and (b) heterogeneous DASR system.

obtained from the Ptolemy [3] design environment. In Figure 7, we demonstrate the E_{sys} values associated with different partition schemes (i.e., C_0, C_{heu}, and C_{opt}, as defined in Section 5.1) and different network sizes.

As shown in the figures of all experiments, we observe that the system energy cost is increased as the network size is increased; moreover, the E_{sys} values of the initial partitioning result (without using any EDP techniques) is large due to the fact that slave nodes need to transmit all samples to the master node, and the master node has a heavy burden to process all of those "raw" samples. With an appropriate partitioning result for balancing the workload between the master node and the slave nodes, the system energy cost is reduced accordingly. Our approach obtains at least a 50% improvement of the energy cost compared to the initial partitions. Note that the results from our heuristic and the exhaustive search approach may be different (the result of exhaustive search is 5% better on average); however, the heuristic approach still improves energy consumption significantly compared to the initial partitioning solution, and it also achieves near-optimal results.

6. Conclusion

In this paper, we have presented a novel algorithm, and associated design methodology for distributing DSP applications across master/slave WSN systems in an energy-efficient fashion. This methodology integrates high-level application modeling, and task- and network-level energy and latency modeling to comprehensively optimize system performance. Results on synthetic benchmarks and on three practical applications demonstrate the utility of our proposed methods. Useful directions for future work include completing an automatic synthesis flow for mapping energy-driven partitioning results onto targeted WSNs, and extending our partitioning constraint model to include resource-limited hardware subsystems.

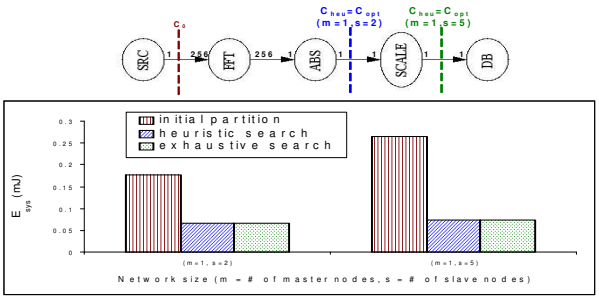
Acknowledgement

This research was sponsored by the Laboratory for Physical Sciences and the Advanced Sensors Collaborative Technology Alliance.

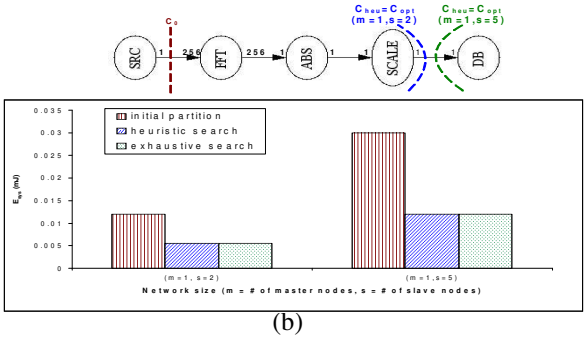
References

[1] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408-2421, October 2001.

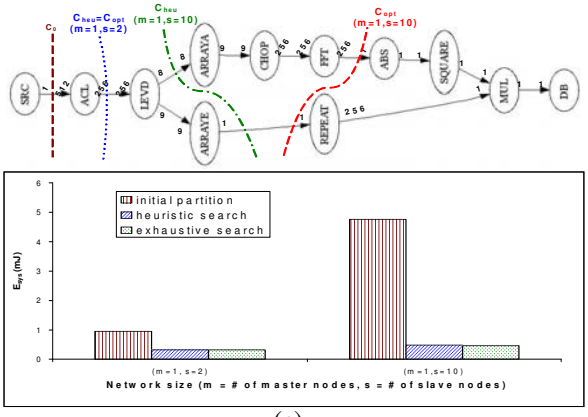
[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. Opti-



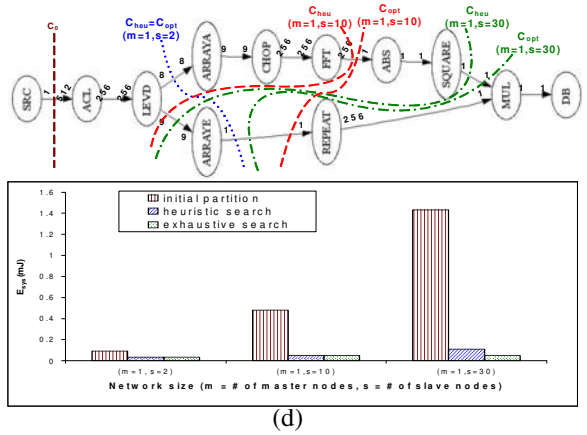
(a)



(b)



(c)



(d)

Figure 7. Energy-driven partitions and E_{sys} comparison for the (a) homogeneous and (b) heterogeneous *spectrum* computation, and the (c) homogeneous and (d) heterogeneous *MES* computation.

mized software synthesis for synchronous dataflow. In *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, July 1997.

[3] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. In *Proceedings of the IEEE*, pages 127-144, January 2003.

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

[5] D. Ganesan, A. Cerpa, W. Ye, Y. Yu, J. Zhao and D. Estrin, Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing*, vol. 64, issue 7, pages 799 - 814, July 2004.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., NY, 1979.

[7] J. Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *Proceedings of the ACM International Conference on Wireless Sensor Networks and Applications*, pages 11-19, September 2003.

[8] C. Hsu, F. Keceli, M. Ko, S. Shahparnia, and S. S. Bhattacharyya. DIF: An interchange format for dataflow-based design tools. In *Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2004.

[9] W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, pages 291-307, 1970.

[10] D. Ko, C. Shen, S. S. Bhattacharyya, and N. Goldsman. Energy-driven partitioning of signal processing algorithms in sensor networks. In *Proceedings of the International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 142-154. Springer, July 2006.

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

[12] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235-1245, September 1987.

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

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

[15] M. Singh and V. K. Prasanna. System-level energy tradeoffs for collaborative computation in wireless networks norwell. In *IEEE International Conference on Communications*, 2002.

[16] A. Wang and A. Chandrakasan. Energy-effiecient DSPs for wireless sensor networks. *IEEE Signal Processing Manazine*, pages 68-78, 2001.