Energy-Driven Partitioning of Signal Processing Algorithms in Sensor Networks

Dong-Ik Ko, Chung-Ching Shen, Shuvra S. Bhattacharyya, and Neil Goldsman

Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies,University of Maryland, College Park MD 20742, USA
{dik, ccshen, ssb, neil}@eng.umd.edu

Abstract. In a sensor network, as we increase the number of nodes, the requirements on network lifetime, and the volume of data traffic across the network, it is often efficient to move towards hierarchical network architectures (e.g., see [5]). In such hierarchical networks, sensor nodes are clustered into groups, and their roles are divided into master and slave nodes for more efficient structuring of network traffic. The operational complexity of each sensor node and the amount of data to be transmitted across sensor nodes strongly influence the energy consumption of the nodes, which ultimately determines the network lifetime. This paper provides a new way of reducing data traffic across nodes by determining and exploiting the lowest data token delivery points within an application graph that is distributed across a network. The technique divides an application graph into two sub-graphs and then distributes each divided subgraph over a master node and its associated slave nodes. The buffer costs of the graph edges over the cutting line corresponds to the amount of data to be transmitted between nodes after allocating the two partial subgraphs such that one subgraph executes on a master node, and the other subgraph is distributed across the associated slave nodes. Since the energy consumption on each node is dominated by the transceiver, the reduced data traffic allows for reducing the turn-on time of the transceivers, and thereby leads to high energy savings. This technique also distributes the workload of sensor nodes in a systematic manner. The more balanced workload also contributes to efficient battery usage, and also improves the latency for processing the data frames captured by the sensor nodes.

1 Introduction and Related Work

The energy consumption of the nodes in a wireless sensor network must be carefully optimized to increase network lifetime. This paper develops an overall minimization of an energy consumption of a sensor network, and provides an efficient trade-off between latency and network lifetime by balancing the workload of the sensor nodes, and carefully determining the points in the application that must communicate across nodes so that the turn-on time of transceivers is minimized.

Many useful approaches have been suggested previously to reduce the energy consumption of sensor nodes. Shih et al. have distributed the FFT function over a master node and slave nodes to reduce energy consumption by moving the function

S. Vassiliadis et al. (Eds.): SAMOS 2006, LNCS 4017, pp. 142 – 154, 2006.

from a cluster head node to slave nodes [11]. Kumar, Tsiatsis, and Srivastava [8] explore energy and latency trade-offs by considering different computational capabilities for master and slave nodes. Other researchers have suggested a hierarchical, physical layer driven sensor network design to reduce data traffic and energy consumption of a sensor node in connection with the physical-layer network functions [10, 12]. In these latter approaches, the node optimization needs to be performed carefully in conjunction with the underlying protocol characteristics.

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, while also taking into account changes in latency due to distributed mapping, and application of a hierarchically clustered sensor network organization. The approach is especially suited for multirate signal processing applications, which exhibit complex and nonuniform patterns of data exchange across functional modules of the application.

Many sensor network applications or important application subsystems can be modeled efficiently with dataflow semantics. By analyzing a well-designed dataflow graph model of an application, operational efficiency can be effectively estimated and optimized at a coarse grain level for various kinds of target architectures (e.g., see [2, 3, 6]). Parameterized dataflow [1] is a form of dataflow that is especially well-suited to sensor network signal processing applications due to its integrated support for adaptation and reconfiguration at various layers of abstraction. Parameterized dataflow allows for dynamic change of variables and configuration settings that can be mapped to module- or subsystem-level parameters of an application.

This paper employs the DGT (dynamic graph topology) [7] method for modeling applications. DGT is a form of parameterized dataflow that emphasizes support for run-time flexibility by allowing for efficient, dynamic changes in application graph topologies based on run-time requests. In DGT semantics, the connections (dataflow edges) between actors (functional modules), as well as the amount of data produced and consumed by the actors can be changed, with the changes expressed in terms of dynamic parameters of the application. In the context of sensor network optimization, this feature can be used to integrate modeling of master/slave node relationships in a clustered network, and also modeling of dynamically changing application graph topologies that execute on sensor nodes.

2 DGT (Dynamic Graph Topology) Specifications

The DGT model allows for dynamic change of graph topologies through schedules that are pre-computed at a compile time. DGT is based on PSDF semantics [1], but is significantly more flexible than PSDF in that it allows graph actors and edges to be treated as dynamic parameters as well as the more standard types of parameters supported in the dynamic reconfiguration capabilities of PSDF. In DGT, as in PSDF, the data transfer rate of a port of an actor (i.e., the number of data values produced or consumed with respect to the incident edge) can be determined by a special subgraph, called the *init* graph. In this way, the consumption rate and production rate of selected ports can be determined dynamically, just before the invocation of the associated DGT graph. Additionally, in DGT, the *subinit* graph Φs can control the behavior of

the associated body graph by dynamically changing the topology (interconnections between actors) of the associated body graph before each invocation (graph iteration) of the body graph. The set of possible graph topologies is predicted at compile time.

Figure 1 shows how a subinit graph can extract appropriate header information and set up parameters (*X*:param) with the required information for the associated body graph. An appropriate graph is selected from a set of possible graphs(G_1, G_2, G_3}by the subinit graph with (*X*:param). This mechanism is effective because many data streams for modern DSP applications are delivered in the form of frames, where each frame has a header part and a payload part, and the header part can be parsed to determine the appropriate graph topology.

Here, we classify actors and ports into two categories based on whether or not their behavior changes dynamically. Actors and ports that are not changed in the graph topology are called fixed actors (a_f) and fixed ports (p_f), respectively, while actors and ports having potential dynamic changes are named as varying actors (a_v) and varying ports (p_v). Here, one point that requires careful consideration is that a fixed actor(a_f) can have a varying port (p_v) since a fixed actor (a_f) can appear with different types of ports.

The *subinit* graph Φs dynamically sets up varying actors and varying ports based on data being delivered and produces an appropriate graph topology for the associated body graph. Deadlock-free operation and bounded memory requirements for each possible set of graph topologies are verified at compile time. At runtime, the *subinit* graph Φs sets up an appropriate graph topology for the associated body graph and selects an appropriate pre-computed schedule that also contains code and buffer size that is minimized for the configured graph. Code and buffer size minimization is obtained by a scheduling technique appropriately chosen depending on graph characteristics. In DGT, verification of validity of schedules can be performed at compile time and valid schedules can be guaranteed and can be ready to use at runtime without the overhead of fully dynamic scheduling. At runtime, *the subinit* graph Φs looks up pre-computed schedules in a table with the active set of parameter values.

3 Energy Consumption Optimization by Distribution of an Application

3.1 Application Cutting in a Sensor Network

In a clustered sensor network, each sensor node captures data from its set of one or more sensors. The captured data can be sent to the associated master node immediately, or the data can be processed to some degree within the slave node before it is sent to the master node. For the data processing functionality, each edge within the application dataflow graph may have different data transfer characteristics. It is useful to consider these characteristics carefully when dividing a dataflow graph for processing across a master- and slave-node pair.

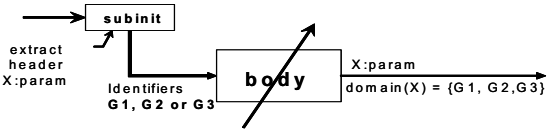


Fig. 1. Illustration of DGT semantics

Dividing an application graph in this manner generally allows us to reduce the amount of data that must be transmitted between the nodes, and it also allows us to balancing the workloads of sensor nodes. The amount of data that must be transmitted directly influences the turn-on time of the sensor node transceivers, which are major sources of energy consumption. Similarly, distributing the workload of an application for balanced processing increases network lifetime through balanced battery usage across the sensor nodes. Therefore, it is useful to partition dataflow graphs across sensor nodes with joint consideration of data transfer volume and workload balance.

Synchronous dataflow (SDF) is an especially useful model, due to its predictability and formal properties, for representing many signal processing applications [2, 9]. In SDF, the number of data values (tokens) produced and consumed by each actor is constant. As a result of this restriction, graphs can be scheduled statically based on the so-called repetition vector $\vec{R}$), which is a vector that is indexed by the actors in the graph, and gives the number of times that each actor needs to be invoked in a static schedule for the graph. Such a schedule can be repeated indefinitely with bounded memory requirements to process the indefinite-length data streams that are characteristic in the signal processing domain.

The number of tokens that are transferred across an edge in the dataflow graph in each schedule iteration can be obtained from the repetitions vector $\vec{R}$ and the number of tokens produced by the source actor of the edge. Given a partition of the dataflow graph into two parts, the total number of tokens that must be transferred (buf_{tr}) across the partition can be obtained by summing up the token transfer volumes of the edges that cross the partition.

The repetitions vector can be obtained through (1) and (2) [9]:

$$T(e,v) = \begin{cases} prd(e) & \text{if v = src(e)} \\ -cns(e) & \text{if v = snk(e)} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$T \bullet \vec{R} = 0 \quad (2)$$

In (1), *prd(e)* is the number of tokens produced onto edge *e* by each execution of *src(e)*, which denotes the source actor of *e* . Similarly, *cns(e)* is the number of tokens consumed from *e* by each execution of *snk(e)*, which is the sink actor of *e*.

The total number of tokens buf_{tr} that cross a given partition in a schedule iteration can then be expressed as

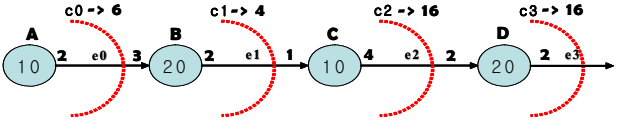
$$buf_{tr} = \sum_{i=1}^{N_c} \sum_{j=1}^{Edge_{n_i}^{\rightarrow}} R(n_i) \cdot prd(e_j(n_i)) \tag{3}$$

where N_c is the number of actors whose outgoing edges cross the partition; $(n_1, n_2, \cdots, n_{N_c})$ is an ordering of the actors whose outgoing edges cross the partition; $Edge_{n_i}$ is the number of outgoing edges of actor n_i that cross the partition; and $e_j(n_i)$ is the j th outgoing edge of n_i that crosses the partition, based on some ordering of the outgoing edges.

Figure 2(a) illustrates how data transmission requirements can change depending the selection of a partition. Figure 2(a) provides four possible candidates for a "cutting line" to determine the partition. The edges that cross the cutting line determine the network data transfer volume that must be incurred on each graph iteration due to the associated application partition. The number shown inside each actor represents the processing complexity in terms of the actor execution time. The number on the left side of an edge represents the number of tokens produced by the source actor, and the number on the right side represents the number of tokens consumed by the sink actor.

In Figure 2, there are four edges, e0,e1,e2 and e3 . Figure 2(b) shows the repetition vector for Figure 2(a), and Figure 2(c) shows buf_{tr} for each cutting line candidate $C0$-$C3$.

After a cutting line is determined for a graph, the graph is effectively divided into "left" and "right" subgraphs, where the left subgraph represents preprocessing of sensor



a) cutting line candidates

$$\begin{matrix} & A & B & C & D \\ e0 \\ e1 \\ e2 \end{matrix} \begin{bmatrix} 2, & -3, & 0, & 0 \\ 0, & 2, & -1, & 0 \\ 0, & 0, & 4, & -2 \end{bmatrix} \bullet \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \vec{R} = [3,\ 2,\ 4,\ 8]$$

b) Repetition vector

$$Buf_{tr,c0} = R_A \cdot buf_p(e_0(A)) = 3 \cdot 2 = 6$$
$$Buf_{tr,c1} = R_B \cdot buf_p(e_1(A)) = 2 \cdot 2 = 4$$
$$Buf_{tr,c2} = R_C \cdot buf_p(e_2(A)) = 4 \cdot 4 = 16$$
$$Buf_{tr,c3} = R_D \cdot buf_p(e_3(A)) = 8 \cdot 2 = 16$$

c) buf_{tr}s for each cutting line

Fig. 2. An illustration of partitioning (cutting line) trade-offs

signals and the right subgraph represents postprocessing. Accordingly, the left subgraph is allocated to the associated slave node, and the right subgraph is allocated to a master node.

Each cutting line in general leads to different workload distributions of an application graph, as well as different values of buf_{tr}. Intuitively,$C0$ leads to increased workload for the master node, since the master node is in charge of most of the data processing functionality. That value of buf_{tr} for $C0$ is 6 tokens. Similarly, $C3$ increases the workload of the slave node, while alleviating the workload of the master node; however, buf_{tr} for $C3$ increases to 16 tokens. As an alternative to $C0$ and $C3$, $C1$ allows for lower data transmission and more balanced workload distribution.

3.2 Cutting Algorithm

Cutting an application dataflow graph is an NP hard problem. However, in many sensor network applications, particularly those involving very simple, ultra-low cost/ power sensor node processing, the application graphs are of limited size, and are manageable by exact techniques. This paper uses an exhaustive search method for finding the best cutting line to target such applications and to demonstrate the potential of high-level, dataflow graph analysis for coordinating the processing across senor nodes.

More precisely, given an application dataflow graph Φ, our objective is to partition Φ into two subgraphs Φ_1 and Φ_2. In this partitioning, we would like to minimize

$$buf_{tr,\, c_i(\Phi)} \tag{4}$$

subject to

$$\text{if } n \in \text{actors}(\Phi_2), \text{ then } \text{successors}(n) \subset \text{actors}(\Phi_2) \text{ and} \tag{5}$$

$$t(\Phi_1) - \delta t(\Phi_2) \leq \Omega \tag{6}$$

Here, $t(X)$ is the execution time of subgraph X , assuming that the subgraph is assigned to the same sensor node, and processing resources across the nodes are homogeneous. The formulation can easily be extended to handle heterogeneous processing resources, but for clarity and conciseness, we focus here on the homogeneous case. The subgraph execution time is obtained by adding the execution time estimates for the individual actors in the subgraph. Also, $\text{actors}(X)$ represents the set of actors in subgraph X, and given an actor n , $\text{successors}(n)$ represents the set of immediate graph successors of n . The constraint in (5) is necessary to avoid cyclic dependencies (potential deadlock) between the master and slave node.

The parameter δ is a coefficient that affects the load balancing aspect of the optimization. An appropriate choice for δ can be estimated by experimentation, or one can run the optimization multiple times for different values of δ and take the most attractive result. As the value of δ is increased, the workload of the master node is decreased, and the latency of the application is also generally decreased since the workload of the application is more distributed over slave nodes. The symbol Ω represents a tolerance for workload imbalance in conjunction with δ.

3.3 Effect on Energy Consumption

The total energy of a sensor node E can be divided into two parts: E_{radio} and E_{mc}, where E_{radio} represents the energy consumed by the transceiver, and E_{mc} represents the energy consumed by the microcontroller and the associated peripherals, such as the memory, UART, and ADC, apart from the transceiver. Thus,

$$E = E_{radio} + E_{mc} \tag{7}$$

The transceiver energy E_{radio} is usually dominant in the total energy consumption of a sensor node, and in the context of dataflow processing, this energy is proportional to the number of tokens that must be communicated. An optimal cutting of an application graph in terms of token transfer minimization across the cutting line therefore results in optimal streamlining of transceiver turn on time. In other words, by reducing buf_{t_r}, E_{radio} can be minimized under the workload balance constraints.

Each partitioned subgraph is mapped to a slave node or a master node. The operations of a subgraph apart from its transceiver-related operations are modeled by E_{mc}. Through a minor abuse of notation, we represent the energy consumption for data processing in an application *appl* as $E_{mc}(appl)$. By distributing the application over a master node and a slave node, $E_{mc}(appl)$ can be divided into two sub energy consumption components: $E_{mc,s}(appl)$ and $E_{mc,m}(appl)$, corresponding respectively to the slave and master nodes. Thus, we have

$$E_{\mathrm{mc}}(appl) = E_{\mathrm{mc,s}}(appl) + E_{mc,m}(appl) \tag{8}$$

In a sensor network cluster that consists of a single master node and η slave nodes, the master node iterates η times to process data frames from all of its slave nodes. Then $E_{\mathrm{mc,m}}$ is the total energy consumption for microcontroller-related functions by the master node during its η iterations of right-side-subgraph processing of data frames received from the slave nodes. The relationships among $E_{\mathrm{mc,m}}$, $E_{\mathrm{mc,s}}(appl)$, and $E_{\mathrm{mc,s}}(appl)$ can be summarized as

$$E_{\mathrm{mc,m}} = \eta E_{\mathrm{mc,m}}(appl) = \eta(E_{\mathrm{mc}}(appl) - E_{\mathrm{mc,s}}(appl)) \quad \text{and} \tag{9}$$

$$E_{\mathrm{mc,s}} = E_{\mathrm{mc,s}}(appl) \tag{10}$$

$E_{\mathrm{mc,s}}$, which is the total energy consumption for microcontroller-related functions of a single slave node, is equal to $E_{\mathrm{mc,s}}(appl)$ since data frames for an application graph are transmitted from a slave node to a master node, and for a single data frame, one iteration of a left-side (slave node) sub-graph is activated. Here, $E_{\mathrm{radio,m}}$ is proportional to η since the transceiver of the master node should be turned on during the entire reception of η data frames from the η slave nodes.

The total energy consumed by the master node can be expressed as

$$E_m = E_{\mathrm{mc,m}} + E_{\mathrm{radio,m}} = E_{\mathrm{mc,m}} + \lambda\eta E_{\mathrm{radio,s}} \tag{11}$$

where λ is a coefficient that relates $\eta E_{\mathrm{radio,s}}$ and $E_{\mathrm{radio,m}}$. Since typically $\lambda\ \eta \gg 1$, the master node has significantly more energy consumption compared to the slave nodes. To reduce the overall energy consumption, the number of tokens that must be

transmitted across the nodes should be minimized under the given workload distribution constraints.

3.4 Effect on Latency

The latency for processing a single data frame of a given application depends on the number of slaves in the network cluster, the network topology, and the volume of data contained in each data frame. For a cluster that consists of a single master node and η slave nodes, the latency $L(app)$ for processing a single application data frame can be expressed by (12), independent of the underlying transmission protocol.

$$L(app) = \eta L_{m,\,\mathrm{frame}}(app) + L_{s,\,\mathrm{frame}}(app) + \eta L_{tr,\mathrm{frame}}(app) \tag{12}$$

where $L_{m,\,\mathrm{frame}}(app)$ is the latency of master node (right-side subgraph) processing for a single data frame, and $L_{s,\,\mathrm{frame}}(app)$ is the corresponding latency of slave node processing. In total, a latency of $\eta L_{m,\,\mathrm{frame}}(app)$ is induced on the master node to process the data from all of the slave nodes. The slave nodes, however, can operate in parallel, and thus, the latency required for slave node processing is independent of the number of slave nodes within the network cluster.

$L(app)$ also depends on the network delay for transmitting data frames across nodes. $L_{tr,\mathrm{frame}}(app)$ thus denotes the latency for transmitting a single data frame from a slave node to the master node. The total transmission latency for delivering η data frames from the slave nodes becomes $\eta L_{tr,\mathrm{frame}}(app)$.

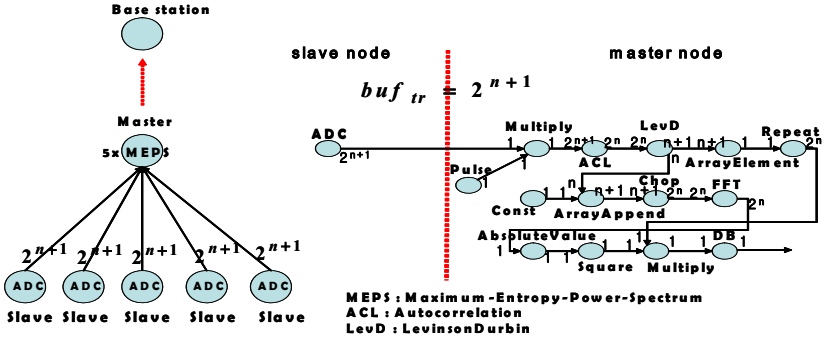
Clearly, $L_{tr,\mathrm{frame}}(app)$depends on the data frame size. In particular,$L_{tr,\mathrm{frame}}(app)$ is proportional to buf_{tr}.

Figure 3 shows three different cases of cutting line selection for an application example that involves maximum entropy spectrum computation. This application is based on an example in the Ptolemy II design environment [4]. The application can be divided into two subgraphs, which are allocated to master and slave nodes as illustrated in the figure. The dotted lines represent cutting line candidates. The application is characterized by a parameter n , called the *order* of the spectrum computation.

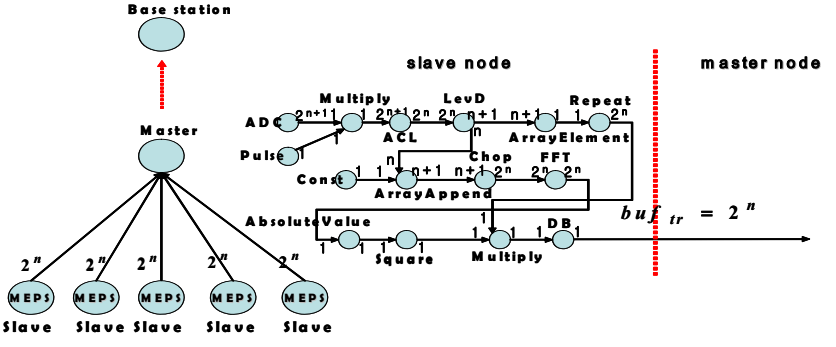
In Figure 3(a), the slave nodes capture raw data frames and send them directly to the master node, where the maximum entropy spectrum processing is performed. Here, buf_{tr} between a single slave node and the master node is 2^{n+1}. Therefore, the total data transmission for each data frame from the 5 slave nodes is $5 \times 2^{n+1}$.

In Figure 3(b), each slave node fully processes a data frame before sending to the master node. This is a fully distributed approach, which minimizes the workload of the master node. In this approach, each slave node sends 2^{n} tokens to the master node. Thus, the total data transmission from the 5 slave nodes is 5×2^{n}.

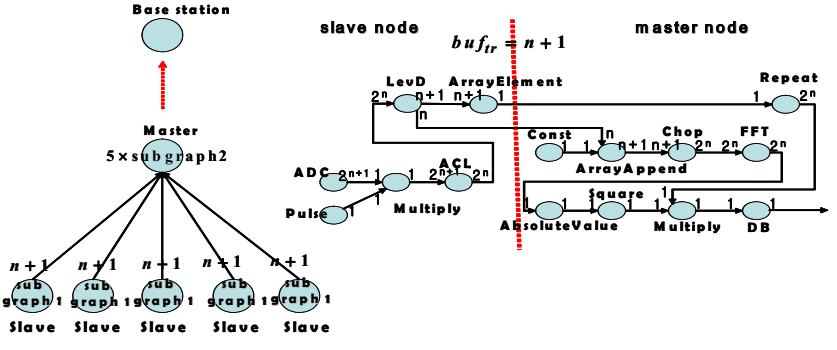
In Figure 3(c), on the other hand, the application graph is divided more evenly into two subgraphs A and B . A copy of subgraph A is assigned to each slave node, and B is allocated to the master node. The carefully-constructed cutting line between A and B reduces buf_{tr} to $(n + 1)$, which results in total slave-to-master data transmission of $5\times(n + 1)$.



a) case 1



b) case 2



c) case 3

Fig. 3. Application mapping over sensor nodes

Without consideration of $L_{tr,\text{frame}}(app)$, the application latencies (L (app)) of the three cases in Figure 3 are related as ($L_{case1} > L_{case3} > L_{case2}$). Case 2 provides the maximal workload distribution by allowing raw data frames to be fully processed in the slave nodes. However, the greatly-reduced$L_{tr,\text{frame}}(app)$ of Case 3 offsets the

increase in $L_{m,\,\mathrm{frame}}(app)$ due to the increased workload of the master, while allowing reduced energy consumption because of reduced transceiver demands.

In summary, the example of Figure 3 illustrates the trade-offs that we can explore among processor workload balancing, latency cost, and transceiver requirements when considering different cutting lines for a multirate signal processing application.

3.5 Cutting Algorithm Under DGT Semantics

Each sensor node in a sensor network can be configured to execute different dataflow graphs depending on any changes in the network's functionality. This requirement leads naturally to a separate dataflow topology for each possible application configuration. As described in Section 2, DGT allows for modeling and software synthesis of a dataflow graph with alternative graph topologies under a single dataflow model. Under DGT semantics, the suggested cutting algorithm can be applied to each graph configuration to generate distributed subgraphs for each possible graph topology.

For example, suppose graph G can be configured into three different cases of graph topologies (G1,G2 or G3) depending on changes in sensor network functionality. Our cutting technique is then iteratively applied at compile time to generate a corresponding set of graph partitions (($G1_{sub1}$, $G1_{sub2}$),($G2_{sub1}$, $G2_{sub2}$) and ($G3_{sub1}$, $G3_{sub2}$)) for (G1,G2 or G3), respectively. Under DGT semantics, GN_{sub1} is configured as a slave node at runtime, while its counterpart subgraph GN_{sub2} is set up for a master node. Thus, the suggested technique is applied to each possible graph topology at compile time to obtain an optimal dataflow graph distribution over the network. The compile time partitions derived by our integrated DGT/ graph-cutting techniqe are used at run time along with other relevant scheduling information to achieve a power efficient, adaptive operation of the network.

4 Experimental Results

We have developed experimental prototype platforms (Figure 4) for master and slave nodes using reconfigurable off-the-shelf components, including the Texas Instruments MSP430 microcontroller, the LINX Technologies 916MHz wireless transceiver, and a microphone sensor. The MSP430 provides a 16-bit processor core, along with a 12-bit ADC, 16-bit hardware timer, UART, 48kB program memory, and 10kB data memory.

Figure 5 and Figure 6 show experimental results where we measured the current consumption from our prototype platforms as they were running different partitionings of the maximum entropy spectrum application. In these experiments, we used TDMA operations for wireless communication. For the TDMA operations, we used 10 time slots per frame, and 250ms per time slot to guarantee that transmission and relevant computations can be completed within each slot.

Figure 5 shows experimental results for current consumption comparison in three different application mapping cases involving a single master node and three slave nodes when $n = 8$ is the application order. The amounts of data (in bytes) that must be

transmitted and received between nodes in each slot under cases 1, 2, and 3 are, respectively, 512(2^{8+1}), 256(2^{8}.) and 9(8+1).

Figure 5 shows that sensor node platforms consume much more current when the nodes are transmitting or receiving data compared to when the nodes are in their idle modes. Also, transceiver operation dominates the overall current consumption when data is being transmitted or received.

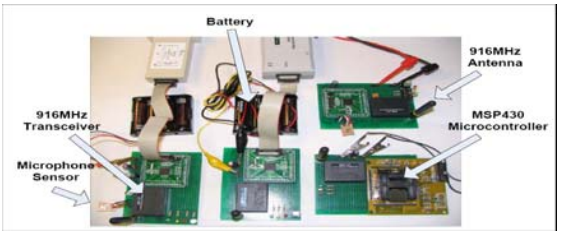
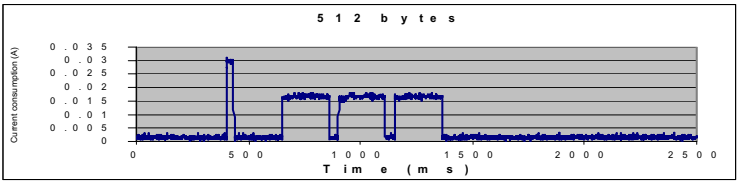
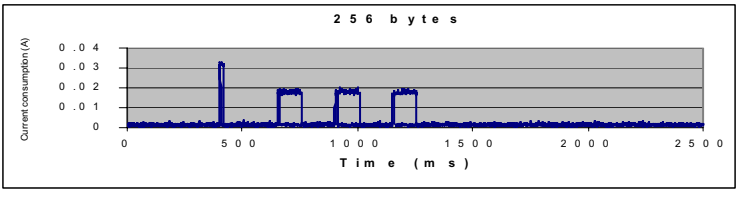


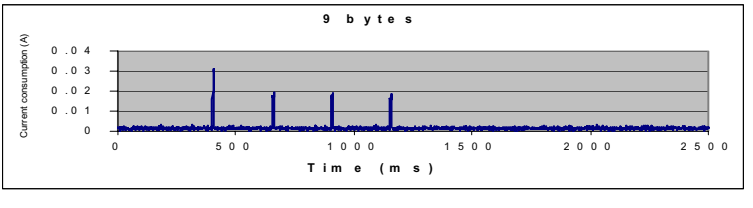
Fig. 4. MSP430-based sensor node platforms



a) case 1(512B)



b) case 2(256B)



c) case 3(9B)

Fig. 5. Current consumption comparison of three application mapping

According to the results in Figure 5, we observe that case 3 of the suggested application cutting technique consumes 70.5% less energy than case 1 and 56.5% less than case 2. Here, the current and voltage for each sensor node are obtained by a digital storage oscilloscope. The power consumption for a time frame is obtained according to the sampling points for current and voltage values. The energy consumption within a TDMA time frame is calculated by integrating the power consumption over the time frame. Because the TDMA operations provide a periodic way to generate similar modes of operations for consecutive time frames, we calculate energy consumption results for several time frames and compute average values from these results.

Figure 6 shows how energy comparison varies as the application order parameter*n* is changed. For each order number, we measured current consumption and voltage on our prototype platforms, and calculated the average energy consumption based on the TDMA time frames. According to the results in Figure 6, we observe that as the order number is increased, the disparities between different application mapping cases become more prominent.

Table 1. Latency comparison for different values of order

order	3	4	5	6	7	8
case1	180ms	254ms	404ms	721ms	1364ms	2699ms
case2	64ms	92ms	150ms	270ms	515ms	1021ms
case3	146ms	191ms	280ms	474ms	864ms	1683ms

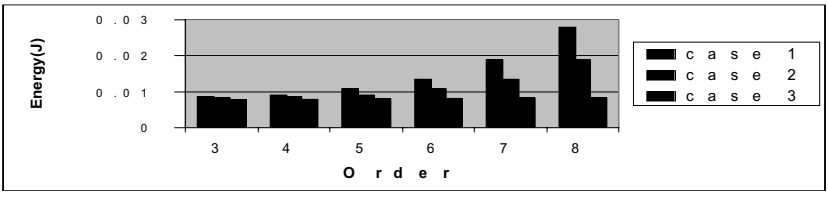


Fig. 6. Energy consumption comparison for different order values

Table 1 shows that as the application order increases, which results in increased data transmission, the relative latency gap between case 2 (best latency) and case 3 (best energy consumption) decreases. For any order, case 1, which is the conventional master-node-centric mapping, generates the worst latency and energy consumption pattern for our benchmark applications.

5 Summary

In this paper, we have developed a technique to partition an application graph into subgraphs to optimize the workload distribution and data transmission when mapping

the application onto a hierarchical sensor network. The technique allows the overall energy consumption of a sensor network to be minimized without considerable loss of latency. In our future work, we will explore the integration of error correction into our partitioning framework to provide further savings in energy consumption.

References

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

[2] S. S. Bhattacharyya, P. K. Murthy, E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Publishers, 1996.

[3] J. T. Buck, E. A. Lee, "Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model", *Proc. ICASSP*, April, 1993.

[4] J. Eker et al., Taming heterogeneity — the Ptolemy approach. *Proceedings of the IEEE*, January 2003.

[5] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences*, 2000

[6] A. Kalavade and P. A. Subrahmanyam, "Hardware / Software Partitioning for Multi-function Systems", *Proc. International Conference on Computer Aided Design*, pp. 516-521, Nov. 1997.

[7] D. Ko and S. S. Bhattacharyya. Dynamic configuration of dataflow graph topology for DSP system design. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pages V-69-V-72, Philadelphia, Pennsylvania, March 2005.

[8] R. Kumar, V. Tsiatsis, and M. B. Srivastava, Computation Hierarchy for In-network Processing, the 2nd ACM international conference on Wireless sensor networks and applications, pp. 68.77, 2003.

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

[10] S. Lindsey, C. Raghavendra, and K. Sivalingam, Data Gathering in Sensor Networks using the Energy Delay Metric. IEEE Transactions on Parallel and Distributive Systems, special issue on Mobile Computing, pp. 924-935, April 2002

[11] 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 Proc. ACM MOBICOM'01, July 2001.

[12] M. Singh and V. K. Prasanna, System-Level Energy Tradeoffs for Collaborative Computation in Wireless Networks Norwell, MA: Kluwer, 2002.