Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks

Yang Yu and Viktor K. Prasanna
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-2562
{yangyu, prasanna}@usc.edu

Abstract

We propose an energy-balanced allocation of a real-time application onto a single-hop cluster of homogeneous sensor nodes connected with multiple wireless channels. An epoch-based application consisting of a set of communicating tasks is considered. Each sensor node is equipped with discrete dynamic voltage scaling (DVS). The time and energy costs of both computation and communication activities are considered. We propose both an Integer Linear Programming (ILP) formulation and a polynomial time 3-phase heuristic. Our simulation results show that for small scale problems (with ≤ 10 tasks), up to 5x lifetime improvement is achieved by the ILP-based approach, compared with the baseline where no DVS is used. Also, the 3-phase heuristic achieves up to 63% of the system lifetime obtained by the ILP-based approach. For large scale problems (with 60 - 100 tasks), up to 3.5x lifetime improvement can be achieved by the 3-phase heuristic. We also incorporate techniques for exploring the energy-latency tradeoffs of communication activities (such as modulation scaling), which leads to 10x lifetime improvement in our simulations. Simulations were further conducted for two real world problems – LU factorization and Fast Fourier Transformation (FFT). Compared with the baseline where neither DVS nor modulation scaling is used, we observed up to 8x lifetime improvement for the LU factorization algorithm and up to 9x improvement for FFT.

I. Introduction

Wireless sensor networks (WSNs) are being developed for a wide range of civil and military applications, such as target tracking, infrastructure monitoring, habitat sensing, and battlefield surveillance [6], [10]. WSNs usually contain a number of networked sensor nodes with each sensor node consisting of computation, communication, and sensing devices. These sensor nodes collaborate with each other to realize certain applications.

This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-C-00-1633.
A preliminary version of this paper appears in ACM LCTES 2003.

For instance, in a target tracking application, up to thousands of sensor nodes are dispersed over a specific area of interest. The sensor nodes are usually organized into clusters [13], [32] with each cluster consisting of tens of sensor nodes. Distributed signal detection and collaborative data processing are performed within each cluster for detecting, identifying, and tracking vehicles. Some of the operations involved in such data processing include the LU factorization [5] and the Fast Fourier Transformation (FFT) [7].

Energy efficiency is a key concern in WSNs. The large number of sensor nodes involved in the system and the need to operate over a long period of time require energy-aware design and operation at all levels of abstraction, from the physical layer to the application layer. However, while many hardware techniques [1], [14], network protocols [13], [16], and data processing algorithms [18], [19] have been proposed for energy-aware design, systematic mechanisms for designing energy-aware collaborative processing between sensor nodes still need to be addressed.

The state of the art in WSN design is largely ad-hoc – system planning and resource management are done without a systematic methodology. This can lead to inefficient utilization of the system. The main motivation of our efforts is to develop techniques for systematic and rapid design and deployment of WSN applications [3], [25], [32].

We focus on the development of energy-efficient collaborative algorithms for WSNs based on high-level computation models of WSNs. Such high-level models allow designers to make informed decisions regarding energy and time tradeoffs at the node and network level – creating a modular, layered paradigm for application development. Toward such a goal, we study the following problem in this paper.

Energy-Balanced Task Allocation Problem: We consider a single-hop cluster of homogeneous sensor nodes connected through multiple wireless channels. Each sensor node is equipped with dynamic voltage scaling (DVS) [30] The target application consists of a set of communicating tasks. Throughout the paper, the term *activity* refers to either a computation task or a communication request. We consider an epoch-based scenario [18], where an instance of the application is executed during the beginning of each epoch and must be completed before the end of the epoch. Such a requirement is usually called the *latency constraint*, We use the term *period* to indicate the length of each epoch. Also, we assume that time-synchronization schemes (e.g., [9]) are available within the cluster.

We consider the *exclusive access constraint*. Specifically, a non-preemptive scheduling policy is employed by each sensor node and each wireless channel. Also, at any time, a sensor node can receive or send data by using at most one channel. The underlying network protocol is assumed to be capable of scheduling a communication activity over a specified channel according to the start and finish time of the activity. Such a scheduling policy requires coarse-level bandwidth reservation mechanisms, which can be provided by, for example, a time-division multiple-access (TDMA) protocol. Moreover, we consider the *task placement constraint*, which is typically required when certain tasks for sensing the raw data must be allocated onto different sensor nodes.

A task allocation is defined as (1) the *assignment* of tasks onto sensor nodes, (2) the *voltage settings* of tasks, (3) the *assignment* of communication activities onto channels, and (4) the *scheduling* of computation and communication activities. Our general goal is to find an allocation in order to maximize the lifetime of the cluster. Toward such a goal, we propose an energy-balanced task allocation such that the maximal energy dissipation among all sensor nodes during each period is minimized, subject to the latency, exclusive access, and task placement constraints.

Our Contributions: The idea of energy-balanced task allocation to a single-hop cluster in WSNs is proposed. As we shall see in Section II, most of the previous efforts in energy-aware task allocation or resource management try to minimize the overall energy dissipation of the system. This strategy may not be suitable in the context of WSNs, since each sensor node is equipped with its own energy source. Moreover, for event-driven systems, applications often need to be executed after the system has been working for sometime. In such a case, an energy-balanced task allocation should also consider the fact that the remaining energy can vary among sensor nodes.

To the best of the authors' knowledge, this is the first work for task allocation in WSNs that considers the time and energy costs of both the computation and communication activities. We first present an integer linear programming (ILP) formulation of our problem. The optimal solution of the problem can be obtained by using a commercial software package such as [28], though the running time of such a software can be large. Next, we propose a polynomial time 3-phase heuristic. Finally, we incorporate techniques that explore the latency-energy tradeoffs of communication activities, such as modulation scaling [23].

Our simulation results show that for small scale problems, up to 5x lifetime improvement is achieved by the ILP-based approach, compared with the case where no DVS is used. Also, the 3-phase heuristic achieves up to 63% of the system lifetime obtained by the ILP-based approach. For large scale problems, the 3-phase heuristic achieves up to 3.5x lifetime improvement when only DVS is used. By incorporating modulation scaling, up to 10x lifetime improvement was observed. Simulations were also conducted for application graphs from two real world problems – LU factorization and FFT. We observed a lifetime improvement of up to 8x for the LU factorization algorithm and up to 9x for FFT.

Paper Organization: We discuss the related work in Section II. The energy-balanced task allocation problem is defined in Section III. The ILP formulation of the problem is given in Section IV. The 3-phase heuristic is described in Section V. Techniques, such as modulation scaling, are incorporated into our approaches in Section VI. Simulation results are demonstrated in Section VII. Finally, we give concluding remarks in Section VIII.

II. Related Work

Extensive research efforts have studied the problem of energy-efficient task allocation and scheduling with DVS in uni-processor real-time systems, including [2], [15], [24], [30]. Recently, research interests have been shifted to multi-processor systems. A list-scheduling based heuristic is proposed in [12], to dynamically recalculate the priority of communicating tasks. In [17], static and dynamic variable voltage scheduling heuristics for real-time heterogeneous embedded systems are proposed. An approach based on critical-path is used for selecting the voltage settings of tasks. However, both [12] and [17] assume that the task assignment is given. A similar problem to the one studied in this paper is investigated in [33]. A two-phase framework is presented to first determine the allocation of tasks onto processors and then the voltage settings of tasks using convex programming. In [34], a dynamic processor voltage adjustment mechanism for a homogeneous multi-processor environment is discussed. However, the time and energy costs for communication activities are not addressed in any of [12], [33], and [34].

The goal of all the above works is to minimize the overall energy dissipation of the system. While such a goal is reasonable for tightly coupled systems, it does not capture the nature of WSNs. The reason is that to minimize the overall energy dissipation can lead to heavy use of energy-effective sensor nodes, regardless of their remaining energy. The consequent short lifetime of such sensor nodes will very likely hinder the system from delivering required performance. This weakness is a major motivation of the proposed energy-balanced task allocation.

Our work considers the energy and time costs of both computation and communication activities. As indicated by several researches, wireless communication is a major source of energy dissipation in WSNs. By incorporating techniques such as modulation scaling, we can greatly improve the energy-efficiency of the system.

Energy-balanced task allocation bears some resemblance to load-balance in distributed computing. However, the communication activities over the same wireless channel need to be serialized such that run-time contentions can be avoided. The serialization imposes new challenges that distinguish our problem from most of the existing works for load-balance or real-time scheduling in distributed systems.

III. Problem Definition

A. System Model

We consider a set of m homogeneous sensor nodes, $PE = \{PE_i : i = 1, \ldots, m\}$, connected by a single-hop wireless network with K communication channels. The homogeneity refers to the same processor and radio capabilities. Each sensor node is equipped with D discrete voltage levels, listed as $V = \{V_i : i = 1, \ldots, D\}$ in decreasing order. Each voltage level in V corresponds to a specific computation speed (given in cycles per second) of the processor. Let SP_j denote the speed of V_j. Let R_i denote the remaining energy of PE_i. For ease of analysis, we assume that the processors consume zero power during idle state.

Regarding the *exclusive access constraint*, we assume that a non-preemptive scheduling policy is employed by each sensor node and each wireless channel. In other words, the time duration scheduled for different computation (communication) activities over the same sensor node (wireless channel) cannot overlap with each other. Moreover, the underlying communication protocols are assumed to be capable of scheduling communication activities according to the start time of each activity in order to avoid run-time contentions. We assume all channels have the same bandwidth. Let τ denote the time for transmitting one data unit between two sensor nodes over any channel. For ease of analysis, we assume that such a transmission costs the same amount of energy at both the sender and the receiver, denoted by ε. Let τ_s and ε_s denote the startup time and energy costs for communication. The data transmission between two tasks on the same sensor node is performed through the local memory with zero time and energy costs.

For ease of analysis, we assume that the radios are completely shutdown in idle state. The energy cost for shutting down and restarting the radio is assumed to be included in ε_s. Low power paging or signaling channel mechanisms can be used for synchronization between sensor nodes when the radios are shutdown. However, the modeling of the power consumption for such mechanisms is beyond the scope of this paper. We also assume that computation and communication activities can be parallelly executed on any sensor node.

B. Application Model

An epoch-based application [18] consisting of a set of communicating tasks is considered. Let P denote the period of the application, which is the length of each epoch. An instance of the application is activated at time kP, and must be completed by the relative deadline, $(k+1)P$, where $k = 0, 1, 2, \ldots$.

The structure of the application is represented by a directed acyclic graph (DAG), $\mathcal{G} = (T, E)$, where node set T denotes the set of n tasks, $\{T_i : i = 1, \ldots, n\}$, and edge set E denotes the set of e directed communication activities between tasks, $\{E_i : i = 1, \ldots, e\}$. Every edge in E pointing from node T_i to T_j, denoted as (i, j), means that the output of task T_i needs to be transmitted to T_j before T_j can start computation. There is a precedence constraint on two tasks T_i and T_j, if there is a path of alternate nodes and edges from T_i to T_j in the DAG. Similarly, there is a precedence constraint on two communication activities, (i, j) and (i', j'), if there is a path from T_j to $T_{i'}$. A task with no incoming edges is called a *source* task. A task with no outgoing edges is called a *sink* task.

For most applications in WSNs, the source tasks are used for sensing or gathering raw data. For ease of analysis, the *task placement constraint* is defined as that no two source tasks can be assigned to the same sensor node. Nevertheless, our model and approach can be extended to handle the general case that any pair of tasks must be or must not be assigned to the same sensor node.

For any task $T_i \in T$, let C_i denote its workload in terms of the worst-case number of required computation cycles. The execution time of T_i on any voltage level $V_j \in V$, t_{ij}, can be calculated as $t_{ij} = \frac{C_i}{SP_j}$. The voltage level of a sensor node is assumed to be dynamically switched, if necessary, upon the arrival of a task instance. Because at most one switch is needed for executing a task instance, the associated time overhead is assumed to be included in the workload of the task. From [4], the power consumption for executing a task follows a monotonically increasing and strictly convex function of the computation speed, $g_i(\cdot)$, which can be represented as a polynomial function of at least second degree. Hence, the energy dissipation for executing T_i on V_j, e_{ij}, can be calculated as $e_{ij} = g_i(SP_j)t_{ij}$. The exact forms of $g_i(\cdot)$ can vary for different tasks based on their instruction components.

The communication load of any edge $E_i \in E$ is represented by its weight, w_i, as the number of data units to be transmitted. We assume that all the data of an edge is transmitted in one data packet with variable size. For an edge $E_i = (j, k)$, let t'_i and e'_i denote the time and energy costs of the corresponding communication activity, if tasks T_j and T_k are not assigned to the same sensor node. We have $t'_i = \tau_s + \tau w_i$ and $e'_i = \varepsilon_s + \varepsilon w_i$.

C. Task Allocation

Based on the above system and application models, a *task allocation* is defined as (1) the *assignment* of tasks onto sensor nodes, (2) the *voltage settings* of tasks, (3) the *assignment* of communication activities onto channels, and (4) the *scheduling* of computation and communication activities. Each task can be assigned to exactly one sensor node with a fixed voltage setting. Also, each communication activity can be assigned to exactly one channel. An allocation is feasible if it satisfies the latency, exclusive access, and task placement constraints.

The *system lifetime* is defined as the time duration from the time when the application starts execution to the time when any sensor node in the cluster fails due to depleted energy. A general solution to maximize the system lifetime is to allow variable task allocations in different periods. Consequently, the energy cost for each sensor node may vary in different periods. However, due to the high complexity raised by such a solution, we assume that the task allocation remains the same for all application periods. That is, the behavior of the system repeats for each period and every sensor node spends the same energy duration each period. Let $\mathcal{E}_i$ denote the energy dissipation of $PE_i \in PE$ during each application period. Given an allocation, the system lifetime (in number of periods) can be calculated as $\min_i\{\lfloor \frac{R_i}{\mathcal{E}_i} \rfloor\}$. A feasible allocation is optimal if the corresponding system lifetime is maximized among all the feasible allocations.

Note that a more complex definition of the system lifetime would be the time period from the beginning of the application execution to the time when not enough sensor nodes are alive to deliver required performance. However, such a definition is quite application-specific. Thus, a simple but general definition of the system lifetime is adopted in this paper. Now, our task allocation problem can be informally stated as:

Find an allocation of a set of communicating tasks onto a single-hop cluster that minimizes the maximal energy dissipation among all sensor nodes during each application period, normalized by their remaining energy.

IV. Integer Linear Programming Formulation

In this section, we present an ILP formulation of our task allocation problem that captures the behavior of the system during one application period. We first list the notations used in the formulation as follows:

P:	period of the application
t_{ij}, e_{ij}:	time and energy costs of executing task T_i using voltage level V_j
t'_i, e'_i:	time and energy costs of edge $E_i = (j,k)$, if T_j and T_k are not assigned to the same sensor node
$a\|\|b$:	no precedence constraint exists for computation (or communication) activities a and b
$\{x_{ij}\}$:	a set of 0-1 variables such that x_{ij} equals one iff T_i is assigned to PE_j
$\{y_{ij}\}$:	a set of 0-1 variables such that y_{ij} equals one iff the voltage level of T_i is set to V_j
$\{z_{ij}\}$:	a set of 0-1 variables such that z_{ij} equals one iff E_i is assigned to the j-th channel
$\{r_{ij}\}$:	a set of 0-1 variables such that r_{ij} equals one iff T_i and T_j are assigned to the same sensor node
$\{s_{ij}\}$:	a set of 0-1 variables such that s_{ij} equals one iff E_i and E_j are assigned to the same channel
$\{\alpha(i)\}$:	a set of real variables indicating the time when T_i starts execution
$\{\beta(i)\}$:	a set of real variables indicating the time when T_i completes execution
$\{\gamma(i)\}$:	a set of real variables indicating the time when E_i starts transmission
$\{\delta(i)\}$:	a set of real variables indicating the time when E_i completes transmission
$\{p_{ij}\}$:	a set of 0-1 variables such that z_{ij} equals one iff the execution of T_i finishes before T_j starts
$\{q_{ij}\}$:	a set of 0-1 variables such that q_{ij} equals one iff the transmission of E_i finishes before E_j starts

To capture the relative order imposed by the precedence constraints among activities, we define the Constraint set 1 shown in Figure 1. It is easy to verify that the exclusive access constraint for activities with precedence constraints is also enforced by Constraint set 1. However, for activities that do not have precedence constraints between them, an extra set of constraints are needed (Constraint set 2 in Figure 2) to enforce the exclusive access constraint. In addition, the task placement constraint is captured by the Constraint set 3 in Figure 2.

The complete ILP formulation is given in Figure 3, where $\mathcal{E}$ is an auxiliary variable. In the figure, the factor $|x_{ik} - x_{jk}|$ means that the energy cost for (i, j) is counted if exactly one of T_i or T_j is assigned to PE_k, but not both. Clearly, the presented formulation is non-linear. It can be transformed into an ILP formulation by standard linearization techniques [27]. Due to the space limitation, we omit the details of linearization in this paper.

V. Heuristic Approach

In this section, we describe an efficient 3-phase heuristic for solving the task allocation problem. Initially, we assume that the voltage levels for all tasks are set to the highest option (V_1). In the first phase, the tasks are grouped into clusters with the goal to minimize the overall execution time of the application. In the second phase, task clusters are assigned to sensor nodes such that the highest energy dissipation among all sensor nodes, normalized by their remaining energy, is minimized. In the last phase, the system lifetime is maximized by lowering the voltage levels of tasks. The details of the heuristic are as follows.

Phase 1: A task cluster is defined as a set of tasks assigned to the same sensor node with a specific execution order. Communication between tasks within a cluster costs zero time and energy. In this phase, we assume an unlimited number of sensor nodes, implying that the number of clusters is also unlimited. The main purpose of this phase is to eliminate communication activities in order to reduce the overall execution time of the application.

The idea of Phase 1 is similar to the algorithm proposed in [22] (pp. 123 - 131). However, traditional approaches for task clustering usually assume a full connection among processors such that all communication can be parallelized, whereas in our problem, communication activities over the same channel must be serialized. Thus, a new challenge is to select a policy for the serialization that facilitates the reduction of the execution time of the application. We use a simple first-come-first-serve policy to order the communication activities ready at different times. Activities ready at the same time (such as those initiated by the same task) are executed in a non-decreasing order of their communication loads. Nevertheless, more sophisticated policies are also applicable.

The pseudo code for Phase 1 is shown in Figure 4. In the code, L denotes the overall execution time of the application and $C(i)$ denotes the cluster that contains task T_i. Initially, every task is assumed to constitute a cluster by itself. We then examine all the edges in a non-increasing order of their weights. For each edge, (i, j), if the execution time of the application can be reduced by merging $C(i)$ with $C(j)$ without violating the task placement constraint, we perform the merge. Otherwise, T_i and T_j remain in two different clusters. In lines 3 and 6, the function Traverse() is called to traverse the DAG in order to determine the schedule of the tasks and hence L.

The pseudo code for Traverse() is shown in Figure 5. In the code, we maintain a queue of activities, Q_{act}, that stores all the ready computation or communication activities in their expected execution order. We also maintain a timestamp for each task cluster that indicates the finish time for all scheduled tasks within the cluster. Similarly, we maintain a timestamp for each channel that indicates its nearest available time. The timestamps are used to schedule the computation and communication activities in lines 7, 13, and 14. In lines 9 and 14, the timestamps are updated based on the execution time of the scheduled activities. The actions in lines 17 and 18 are important to ensure that the radio can be tuned to at most one channel at any time.

Phase 2: In this phase, we assign the task clusters from Phase 1 onto the actual sensor nodes in PE. Note that multiple clusters can be assigned to the same sensor node. Based on the contained tasks and the corresponding communication activities, we first calculate the energy dissipation of each cluster. Let $\Pi = [\pi_1, \pi_2, \ldots, \pi_c]$ denote the list of all tasks clusters and ξ_i denote the energy dissipation of π_i. The normalized energy dissipation (*norm-energy* for short) of a sensor node is given as the sum of the energy dissipation of the clusters assigned to the sensor node, normalized by the remaining energy of the sensor node.

The pseudo code of Phase 2 is shown in Figure 6. Initially, Π is sorted into a non-increasing order of energy dissipation of clusters. Then, for each cluster in Π, we calculated the norm-energy of every sensor node as if the cluster is assigned to the sensor node (called *expected norm-energy*). We then assign the cluster to the sensor node that gives the minimal expected norm-energy. In the code, function TraverseAssigned() is used to find the execution time of the application based on the resulting assignment. Compared with Traverse(), the modification in TraverseAssigned() is that in line 7 of Figure 5, each computation activity is scheduled on the sensor node that it is assigned to. Thus, timestamps are maintained for all sensor nodes, instead of clusters.

Phase 3: The voltage levels of tasks are adjusted in this phase with the goal to maximize the system lifetime. An iterative greedy heuristic is used (shown in Figure 7). Let $\mathcal{E}$ denote the maximum of the norm-energy among all sensor nodes. The sensor node that determines $\mathcal{E}$ is called the *critical node*. In each iteration, we find the task such that by lowering its current voltage level to the next level, $\mathcal{E}$ can be decreased the most. The increased latency caused by lowering the voltage is added to L. Since the schedule of activities can be changed by the latency increment, L is re-computed by traversing the DAG every time it reaches P (in line 15).

In Figure 7, ed_i denotes the energy gain by lowering the current voltage of T_i to the next level, while td_i denotes the incurred increment in latency. The array composed by ed_i's of all tasks assigned to PE_j is denoted as ED_j.

Time Complexity Analysis: In Phase 1 (Figure 4), the **While** iteration is executed e times. Function Traverse() in line 6 takes $O(n+e)$ time. Thus, Phase 1 needs $O(e(n+e))$ time. In Phase 2 (Figure 6), the ordering in line 1 takes $O(c \log c)$ time. The outer iteration is executed c times. The results of m possible assignments are compared in line 5. The traverse in line 8 takes $O(n+e)$ time. Hence, Phase 2 takes $O(c \log c + mc + n + e)$ time. In Phase 3 (Figure 7), the sorting in line 1 takes $O(n \log n)$ time. The number of voltage switching in line 9 is bounded by dn. To update ED_r in line 10 needs $O(\log n)$ time. Let p denote the number of times for calling TraverseAssigned() in line 12. The time complexity of Phase 3 is $O(dn \log n + p(n+e))$. Although p equals dn in the worst case, it was observed in our simulations that p usually equals 1 or 2. Thus, the overall time complexity of the heuristic is $O((e+p)(n+e) + mc + dn \log n + c \log c)$, which is $O(dn(n+e+\log n) + e^2 + mn)$ in the worst case.

An Illustrative Example: We illustrate the execution of the above heuristic through a simple example. We assume a cluster of 3 sensor nodes connected by 2 channels. Each sensor node have two voltage levels, V_h and V_l, with $SP_h = 1$ and $SP_l = 0.3$. We assume that it costs one time and energy unit for transmitting one data unit over any channel. The application graph is shown in Figure 9(a), with each circle representing a task. The number close to each circle is the required workload, while the number on each edge is the weight of the edge. The time and energy costs for executing tasks at the two voltage levels are given in Figure 9(b). We assume that $P = 250$ time units.

The clustering steps in Phase 1 is shown in Figure 10. In this phase, the voltage levels of all tasks are set to V_h. The sorted edge list with respect to edge weights is $\{(T_4, T_6), (T_1, T_2), (T_3, T_6), (T_6, T_7), (T_2, T_7), (T_5, T_6), (T_1, T_3)\}$. The table in Figure 10 traces the execution of the algorithm, where L_i is the execution time of the application at the completion of step i. The sub-figures (a) through (e) correspond to the application graph at the completion of steps 0, 1, 2, 3, and 5, respectively. The clusters are marked with polygons in dash line. Note that in steps 6 and 7, the clustering is not performed due to the task placement constraint.

During Phase 2, we first calculate the energy dissipation for each cluster – 190 energy units for cluster $\pi_1 = \{T_1, T_2, T_7\}$, 100 for the cluster $\pi_2 = \{T_3, T_4, T_6\}$, and 50 for cluster $\pi_3 = \{T_5\}$. Since the remaining energy for the three sensor nodes are the same, we simply assign π_1 to PE_1, π_2 to PE_2, and π_3 to PE_3.

Finally, we adjust the voltage levels of tasks. Since PE_1 is the critical node, we first set the voltage level of T_2 to V_l, which reduces $\mathcal{E}_1$ to 106 and increases L from 80 to 219. Next, we set the voltage level of T_1 to V_l, which further decreases $\mathcal{E}_1$ to 92 and increases L to 242. After this step, the critical node becomes PE_2 with $\mathcal{E}_2 = 0.1$. Since the latency constraint is 250, our heuristic terminates.

In the above example, we decreases the norm-energy of the critical sensor node from 0.19 to 0.1, implying a system lifetime improvement by a factor around 2.

VI. Incorporating Energy-Latency Tradeoffs for Communication Activities

While DVS has been widely applied into various applications for energy saving in computation activities, techniques for exploring the energy-latency tradeoffs of communication activities are gaining interest. An important observation [11] is that in many channel coding schemes, the transmission energy can be significantly reduced by lowering the transmission power and increasing the duration of the transmission. Techniques such as modulation scaling [23] have been proposed for implementing such tradeoffs. Recently, algorithms for applying such techniques in the context of packet transmissions or data gathering in wireless networks have been studied in [11], [23], [31].

Our approaches can be extended to incorporate the above tradeoffs. In the following, we discuss through the example of modulation scaling that explores the tradeoffs by adapting the modulation level to match the traffic load.

For ease of analysis, we focus on the Quadrature Ampitude Modulation (QAM) scheme [26]. The techniques presented in this paper are extendible to other modulation schemes as well. Given a communication activity E_i with a packet of s_i bits, assuming a fixed symbol rate R_i, the transmission time can be calculated as [23]:

$$\tau_i = \frac{s_i}{b_i \cdot R_i} \ , \tag{1}$$

where b_i is the modulation level in terms of the constellation size (number of bits per symbol). The corresponding energy dissipation can be modeled as a function of τ_i, denoted as $f_i(\tau_i)$. We have [23]:

$$f_i(\tau_i) = [C_i \cdot (2^{\frac{s_i}{\tau_i \cdot R_i}} - 1) + D_i] \cdot \tau_i \cdot R_i \ , \tag{2}$$

where C_i is determined by the quality of transmission (in terms of Bit Error Rate) and the noise power, and D_i is a device-dependent parameter that determines the power consumption of the electronic circuitry of the sensor nodes. The energy-latency tradeoffs for transmitting 1 bit is plotted in Figure 11. The settings for C_i, D_i, and R_i are extracted from [23]. Also, we may estimate the energy dissipation for receiving the packet as

$$f'_i(\tau_i) = D_i R_i \tau_i \ . \tag{3}$$

In practice, the value of b_i is typically set to positive even integers, resulting in discrete values of τ_i. For any communication activity $E_i \in E$, let t'_{ij} denote the time cost with b_i set to the j-th modulation level. Also, let e^s_{ij} and e^r_{ij} denote the corresponding sending and receiving energy costs. We can calculate the values of t'_{ij}'s, e^s_{ij}'s, and e^r_{ij}'s based on equations 1, 2, and 3.

To modify our ILP formulation, a set of 0-1 variables, $\{u_{ij}\}$, are needed to indicate the modulation level of the communication activities. Specifically, u_{ij} equals one iff the modulation level of E_i is set to the j-th level. Moreover, we replace the constraint set marked with * in Figure 1 with the following one, which states that the transmission time of $E_i = (a, b)$ depends on the modulation level for E_i and the locations of T_a and T_b:

$$\forall E_i = (a,b) \in E, \quad \delta(i) = \gamma(i) + \sum_j (u_{ij} t'_{ij})(1 - r_{ab}) \ .$$

Moreover, we change the constraint on the auxiliary variable $\mathcal{E}$ in Figure 3 as follows:

$$\forall PE_k \qquad \frac{\sum_{T_i \in T}\{x_{ik} \sum_j (y_{ij} e_{ij})\} + \sum_{E_i=(a,b)\in E}\{x_{ak}(1-x_{bk}) \sum_j (u_{ij} e^s_{ij}) + (1-x_{ak}) x_{bk} \sum_j (u_{ij} e^r_{ij})\}}{R_k} \le \mathcal{E} \ .$$

For the 3-phase heuristic, we assume that both voltage and modulation levels of the system are set to the highest options in Phase 1 and 2. We then slightly modify Phase 3, such that the energy savings achieved by lowering the modulation levels of communication activities are also examined. The modified pseudo code is shown in Figure 8. One concern is that to decrease the transmission energy at the sender, we actually increase the receiving energy at the receiver. Thus, in lines 13 and 14 of Figure 8, we ensure that the modulation scaling is performed only when the increase in the reception energy does not cause the value of $\mathcal{E}$ to increase. By doing so, our heuristic can handle the situation in highly dense WSNs, where the receiving energy is comparable with the sending energy.

VII. Experimental Results

A simulator based on the system and application models presented in Section III was developed to evaluate the performance of our approach using application graphs from both a synthetic approach and real world problems. The goals of our simulations are (1) to measure and compare the performance of the 3-phase heuristic against the ILP-based approach; and (2) to evaluate the impact of the variations in several key system parameters on the performance of the heuristic, including the tightness of the latency constraint, the relative time and energy costs of communication activities compared with computation activities, and the number of voltage levels.

The evaluation metrics are based on the system lifetime obtained by different approaches. Let LT_{ILP} and LT_{heu} denote the system lifetime obtained by the ILP-based approach and the 3-phase heuristic, respectively. In addition, let LT_{raw} denote the system lifetime obtained by assuming that no voltage or modulation scaling is available (i.e., every sensor node runs and transmits data at the highest speed). Since we do not have a stand alone approach to obtain LT_{raw}, LT_{raw} was calculated based on the value of $\mathcal{E}$ obtained after phase 2 of the 3-phase heuristic.

Unless otherwise stated, all the data presented in this section is averaged over more than 100 instances so that a 95% confidence interval with a 10% (or better) precision is achieved.

A. Synthetic Application Graphs

Experimental Procedure: The structure of the application graph was generated using a method similar to the one described in [8]. The only difference is that we enforce multiple source tasks in the generation of the DAG.

According to Rockwell's WINS node [29], the power consumption of an Intel StrongARM 1100 processor with 150 MIPS is around 200 mW. This implies that the time and energy costs per instruction are around 5 nSec and 1 nJ. Also, the power of the radio module used in WINS is 100 mW at 100 Kbps, implying that the time and energy costs for transmitting a bit are around 10 μSec and 1 μJ. In the following, we set the parameters for our simulator such that the time and energy costs for computation and communication activities roughly follow the above data.

We set the maximum computation speed of each sensor node to 10^2 Mcps (million cycles per second) and the minimum speed to 0.3×10^2 Mcps. It is assumed that other levels of computation speed are uniformly distributed between the maximum and minimum speeds. The computation requirements of the tasks followed a gamma distribution with a mean value of 2×10^5 and a standard deviation of 10^5. The power function of task T_i, $g_i(SP)$, was of the form $a_i \cdot (\frac{SP}{10^8})^{b_i}$, where a_i and b_i were random variables with uniform distribution between 2 and 10, and 2 and 3 [20], respectively. For example, suppose $a_i = b_i = 2$. Then, to execute a task of 2×10^5 instructions costs 2 mSec and 4 mJ in the highest speed, and 6.7 mSec and 1 mJ in the lowest speed.

The time and energy costs of communication activities are determined by the number of data units to transmit and the values of τ and ε. Based on the data for WINS, we set $\tau = 10$ μSec and $\varepsilon = 1$ μJ. To focus on the main issues, we set the startup energy dissipation of the radio to be zero. To study the effect of different communication load with respect to the computation load, the number of bits per communication activity follows a uniform distribution between $200CCR(1 \pm 0.2)$, where CCR (communication to computation ratio) is a parameter indicating the ratio of the average execution time of the communication activities to that of the computation activities. Intuitively, a larger value of CCR implies a relatively heavier communication loads compared with the computation loads. Note that by varying CCR, we abstract not only the variations in the amount of transmitted data, but also the variations in the relative speed of computation and communication devices. In our simulations, CCR was varied within $[0, 20]$.

The period of the application, P, was generated in the following way. We first define the *distance* of a node in the application DAG as the number of edges in the longest path from the source to the node. Nodes are then divided into layers, with nodes in each layer having the same value of distance. Since the average time to execute a task in the highest speed is 2 mSec, the computation time required for a layer is estimated as $2\lceil \frac{p}{m} \rceil$ mSec, where p is the number of tasks in the layer. By doing so, we implicitly assume full parallelism in executing the tasks at each layer. In addition, the expected number of communication activities initiated by a task is estimated as its out-degree subtracted by 1. Assuming there are in total q communication activities requested by all the tasks in a specific layer, the corresponding time cost is estimated as $2CCR\lceil \frac{q}{K} \rceil$ mSec. P is then set to the sum of the computation and communication time cost of all layers over u, where $u \in [0, 1]$ is a parameter that approximates the overall utilization of the system. The setting of u is important as it determines the latency laxity for trading against energy. Intuitively, a larger value of u implies a tighter latency constraint and hence less latency laxity.

The remaining energy of sensor nodes follows a uniform distribution between $E_{mean}(1 \pm 0.3)$, where E_{mean} is a fairly large number.

Small Scale Problems: We first conducted simulations for small scale problems, with 3 sensor nodes, 3 voltage levels, 2 channel, and 7 - 10 tasks. The number of source tasks in the application graph is set to 2, while the maximal in-degree and out-degree for each node are set to 3. A commercial software package, LINDO [28], was used to solve the ILP problems. Due to the large running time for solving some problem instances, LINDO was interrupted after two hours of execution if the optimal solution was not yet found. Then, the best solution obtained so far was returned. We observed that in most cases, LINDO was able to find the optimal solution within two hours.

The data shown in Figure 12 is averaged over more than 70 instances so that each data point has a 95% confidence interval with a 10% precision. In Figure 12(a), we illustrate the lifetime improvement achieved by the ILP-based approach, which is calculated as $\frac{LT_{ILP}}{LT_{raw}} - 1$. We can see an improvement around 3x - 5x. Figure 12(b) shows the performance ratio of the 3-phase heuristic over the ILP-based approach, i.e., $\frac{LT_{heu}}{LT_{ILP}}$. We can see that the 3-phase heuristic achieved up to 63% of the solution obtained by the ILP-based approach for the conducted simulations.

While the running time of the heuristic is around zero, the average running time of the ILP-based approach ranges from 550 Sec ($n = 7$, $u = 0.5$) to 5900 Sec ($n = 10$, $u = 0.8$) on a Sun Blade1000 machine with a UltraSparc III 750 Mhz CPU.

Large Scale Problems: A set of simulations were conducted for evaluating the performance of the 3-phase heuristic for problems with 10 sensor nodes, 8 voltage levels, 4 channels, 60 - 100 tasks, $CCR \in [0, 20]$, and $u \in [0, 1]$. The number of source tasks in the application graph is set to 6. The maximal in-degree and out-degree for each node are set to 5. Due to the large size of the problems, it is impractical to obtain the optimal solutions by using the ILP-based approach. Thus, we use the lifetime improvement achieved by the 3-phase heuristic as the evaluation metric, which is calculated as $\frac{LT_{heu}}{LT_{raw}} - 1$. The simulation results are shown in Figure 13.

An improvement up to 3.5x in the system lifetime can be observed from Figure 13(a). We can see that the improvement increases when u decreases, as the latency laxity increases accordingly. The lifetime improvement saturates when u approaches 0, i.e., the latency constraint approaches ∞. The curve with $u = 0.0$ gives the upper bound of the improvement that can be achieved by our heuristic with respect to variations in CCR.

The effect of CCR is more complicated. For example, when $u = 0.5$, the lifetime improvement increases when $CCR \leq 6$ and decreases when CCR is beyond 6. This is because when CCR is small, the computation activities dominate the overall energy costs of the application. By increasing CCR, we actually increase the latency constraint without increasing the computation load, which in turn can be traded for lifetime improvement. However, when CCR reaches some threshold value, the communication energy cost becomes more significant than that of the computation activities. Thus, the lifetime improvement achieved by reducing computation energy becomes limited. We shall see later that this shortcoming can be overcome by incorporating modulation scaling into our heuristic.

Figure 13(b) shows the lifetime improvement with number of tasks, n varying from 60 to 100. We can see that the performance of our approach is quite stable with respect to the variation in n.

The miss rate (defined as the ratio of the number of instances that an approach fails to find a feasible solution to the total number of instances) of a heuristic is another key issue. Note that in our simulations, not all instances are guaranteed to have feasible solutions. We observed that the miss rate of the 3-phase heuristic is significant only when CCR is close to zero. Thus, we show the miss rate with $CCR = 0$ in Figure 14. Also, the running time of the heuristic is around 0.5 mSec on a Sun Blade1000 machine with a UltraSparc III 750 Mhz CPU.

Impact of the Number of Voltage Levels: We also studied the impact of the variations in the number of voltage levels. Simulations were conducted with 10 sensor nodes, 60 tasks, 4 channels, $CCR = 2$, $u \in \{0.2, 0.5, 0.8, 1.0\}$ and 1 to 10 voltage levels. The results are demonstrated in Figure 15.

The plots show that when $u > 0.2$, the performance of the heuristic can be significantly improved by increasing the number of voltage levels from 1 to 4. Further increase in the number of voltage levels does not improve the performance much. This is understandable since the energy behaves as a monotonically increasing and strictly convex function of the computation speed. The first derivative of the energy function tends to ∞ when the speed tends to ∞. Thus, the most portion of energy saving is obtained by changing the speed from the highest option to some lower options, which can be efficiently achieved with 4 voltage levels per sensor node.

When $u = 0.2$, the latency laxity is so large that the voltage level of most tasks can be set to the lowest option. Thus, there is almost no improvement by increasing the number of voltage levels beyond 2.

Incorporating Modulation Scaling: We used modulation scaling to illustrate the energy-latency tradeoffs for communication activities. Due to the underlying single-hop connection, we assume that all sensor nodes have the identical settings for parameters C_i, D_i, and R_i. From [23], we set $D_i = 10^{-7}$ [23]. To investigate the impact of different energy/time ratio for data transmission, we set C_i to 10^{-7} and 10^{-6} for different instances. The modulation level, b_i, was set to even numbers between 2 and 6. We set $R_i = 1.7 * 10^4$ so that when $b_i = 6$, it roughly takes 10 μSec and 1 μJ to transmit a bit (as shown in Figure 11).

The simulations were conducted with 10 sensor nodes, 8 voltage levels, 3 modulation levels ($\{2, 4, 6\}$), 60 tasks, $u \in \{0.0, 0.2, 0.5, 0.8, 1.0\}$, and $CCR \in [0, 20]$. Compared with Figure 13, we can observe a significant amount of performance improvement in Figure 16. For example, when $u = 0.5$, the highest lifetime improvement increases from 3x in Figure 13(a) to 6x in Figure 16(a) and even 10x in Figure 16(b). The difference in performance improvement of Figures 16(a) and 16(b) is because that a larger C_i leads to larger energy/time ratio of communication activities, which in turn gives more advantage in reducing the communication energy by utilizing modulation scaling.

Similar to Figure 13, larger improvement is observed when u becomes smaller. In addition, the miss rate of the heuristic exhibits a similar trend as the cases with DVS only.

B. Application Graphs from Real World Problems

In addition to synthetic application graphs, we also considered application graphs of two real world problems: LU factorization algorithm [5] and Fast Fourier Transformation [7]. These two algorithms are widely used as kernel operations for various signal processing, such as beamforming [21].

LU Factorization: Figure 17(a) gives the sequential program for the LU factorization without pivoting, where s denotes the dimension of the matrix. The application graph of the algorithm for the special case of $s = 5$ is given in Figure 17(b). Each $T_{k,k}$ represents a pivot column operation and each $T_{k,j}$ represents an update operation. The total number of tasks in the application graph equals $\frac{s^2+s-2}{2}$. Also, we assume the input matrix is available at the sensor node where task $T_{1,1}$ is assigned.

We performed simulations with 10 sensor nodes, 8 voltage levels, 4 channels, 3 modulation levels, and the matrix dimension, s, varying from 5 to 20. Regarding the energy/time ratio for data transmission, we set $D_i = 10^{-6}$. It is easy to verify that the computation requirement of any task, $T_{k,j}$, is $s - k$ ALU operations. Further, for any task, $T_{k,j}$, the size of data transmitted by any communication activity to the task is $s - k$ units in the matrix. We examined two cases with u set to 0.5 and 0.8. In both cases, CCR was selected from $\{1.0, 3.0, 5.0, 8.0, 10.0\}$.

The lifetime improvement achieved by our 3-phase heuristic for the LU factorization algorithm is shown in Figure 18. It can be observed that the performance of the heuristic improves when CCR increases or u decreases. The lifetime improvement approaches 8x when $CCR = 10.0$. Also, very few improvement was observed during our simulations by setting CCR beyond 10.0. The least amount of lifetime improvement is around 15% when $u = 0.8$, $CCR = 1.0$, and $s = 20$.

Fast Fourier Transformation (FFT): The recursive, one-dimensional FFT Algorithm is given in Figure 19(a). In the figure, A is an array of length l which holds the coefficients of the polynomial and array Y is the output of the algorithm. The algorithm consists of two parts: recursive calls (lines 3-4) and the butterfly operation (lines 6-7). For an input vector of size l, there are $2 \times l - 1$ recursive call tasks and $l \times \log l$ butterfly operation tasks (we shall be assuming $l = 2^k$ for some integer k). For example, the application graph with four data points is given in Figure 19(b) . The 7 tasks above the dashed line are the recursive call tasks, while the 8 tasks below the line are butterfly operation tasks.

We performed simulations used 10 sensor nodes, 8 voltage levels, 4 channels, 3 modulation levels. Regarding the energy/time ratio for data transmission, we set $D_i = 10^{-6}$. The vector size was varied from 4 to 64 incrementing by the power of 2. We also examined two cases with u set to 0.5 and 0.8. In both cases, CCR was selected from $\{1.0, 3.0, 5.0, 8.0\}$.

The lifetime improvement achieved by our 3-phase heuristic for the FFT algorithm is shown in Figure 20. Again, the performance of the heuristic improves when CCR increases or u decreases. The lifetime improvement is close to 10x when $CCR = 8.0$ and $l = 64$. The least amount of lifetime improvement is around 75% when $u = 0.8$, $CCR = 1.0$, and $l = 4$.

Note that the above two example applications have exactly one source task that initially holds the entire data set, implying that data dissemination within the cluster is required. However, our technique is also applicable to applications where data are locally sensed or gathered at each individual sensor node. For example, in Figure 19(b), input data can be generated by tasks T4 to T7 through local sensing. Thus, the recursive calls above the dashed line to disseminate the data become unnecessary.

VIII. Concluding Remarks

In this paper, we have investigated the problem of allocating an epoch-based real-time application to a single-hop cluster of homogeneous sensor nodes with multiple wireless channels. A new performance metric has been proposed to balance the energy dissipation among all the sensor nodes. We have presented both an ILP formulation and a polynomial time heuristic. Also, we have incorporated techniques that explore the energy-latency tradeoffs of communication activities.

We have demonstrated through simulations that for small scale problems, a lifetime improvement up to 5x is achieved by the ILP-based approach, compared with the case where no DVS is used. Also, the performance of the 3-phase heuristic achieves up to 63% of the system lifetime obtained by the ILP-based approach. For large scale problems, a lifetime improvements up to 10x was observed when both voltage and modulation scaling were used. Simulations were also conducted for application graphs from LU factorization and FFT. The 3-phase heuristic achieves a lifetime improvement of up to 8x for the LU factorization algorithm and an improvement of up to 9x for the FFT algorithm.

In the future, we would like to validate our approaches using real systems. We are particularly interested in advanced applications for WSNs, where systematic methodologies for task allocation are mostly required for rapid and automated system design.

References

[1] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, and W. J. Kaiser. Wireless integrated network sensor: Low power systems on a chip. In *ESSCIRC ’98*, 1998.

[2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *13th Euromicro Conference on Real-Time Systems*, June 2001.

[3] A. Bakshi, J. Ou, and V. K. Prasanna. Towards automatic synthesis of a class of application-specific sensor networks. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Oct. 2002.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE J. of Solid-State Circuits*, 35(11), Nov. 2000.

[5] M. Conard, M. Marrakchi, Y. Robert, and D. Trystram. Parallel Gaussian elimination on an MIMD computer. *Parallel Computing*, 6:275–295, 1988.

[6] S. Conner, L. Krishnamurthy, and R. Want. Making everyday life easier using dense sensor networks. In *ACM UbiComp*, 2001.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[8] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task graphs for free. In *International Workshop on Hardware/Software Codesign*, pages 97–101, Mar. 1998.

[9] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[10] D. Estrin, L. Girod, G. Pottie, and M. B. Srivastava. Instrumenting the world with wireless sensor networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2001.

[11] A. E. Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi. Energy-efficient scheduling of packet transmissions over wireless networks. In *IEEE InfoCom*, 2002.

[12] F. Gruian and K. Kuchcinski. LEneS: Task scheduling for low-energy systems using variable supply voltage processors. In *Design Automation Conference (DAC)*, pages 449–455, 2001.

[13] W. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application specific protocol architecture for wireless microsensor networks. *IEEE Trans. on Wireless Networking*, 2002.

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[15] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1998.

[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[17] J. Luo and N. K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *VLSI Design*, Jan. 2002.

[18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks,. In *Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[19] C. Meesookho, S. Narayanan, and C. S. Raghavendra. Collaborative classification applications in sensor networks. In *2nd IEEE Sensor Array and Multichannel Signal Processing Workshop*, Aug. 2002.

[20] P. Mejía-Alvarez, E. Levner, and D. Mossé. An integrated heuristic approach to power-aware real-time scheduling. In *Workshop on Power-Aware Computer Systems*, Feb. 2002.

[21] R. A. Mucci. A comparison of efficient beamforming algorithms. *IEEE Trans. on Acoustic, Speech, Signal processing*, ASSP-22:548–558, June 1984.

[22] V. Sarkar. *Partitioning and Scheduling Programs for Execution on Multiprocessors*. The MIT Press, Cambridge, Massachusetts, 1989.

[23] C. Schurgers, O. Aberhorne, and M. B. Srivastava. Modulation scaling for energy-aware communication systems. In *ISLPED*, pages 96–99, 2001.

[24] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *IEEE/ACM International Conference Computer-Aided Design*, pages 365–368, 2000.

[25] M. Singh and V. K. Prasanna. A hierarchical model for distributed collaborative computation in wirelss sensor networks. In *5th Workshop on Advances in Parallel and Distributed Computational Models*, Apr. 2003.

[26] T. Ue, S. Sampei, N. Morinaga, and K. Hamaguchi. Symbol rate and modulation level-controlled adaptive modulation/TDMA/TDD system for high-bit rate wireless data transmission. *IEEE Trans. on Vehicular Technology*, 47(4):1134–1147, Nov. 1998.

[27] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, Ltd, 1999.

[28] The LINDO System Inc. http://www.lindo.com.

[29] The WINS Project, Rockwell Science Center. http://wins.rsc.rockwell.com.

[30] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

[31] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. to appear in IEEE InfoCom 2004.

[32] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE Network Magazine, special issue on Middleware Technologies for Future Communication Networks*, Jan. 2004.

[33] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference (DAC)*, 2002.

[34] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2001.

Constraint set 1:

$\forall T_i \in T$

- $\sum_j x_{ij} = 1$ // every task can be assigned to exactly one sensor node
- $\sum_j y_{ij} = 1$ // every task can be executed using exactly one voltage level
- $\alpha(i) \geq \max_{E_l=(j,i)\in E}\{\delta(l)\}$ // T_i starts execution after receiving all input data
- $\beta(i) = \alpha(i) + \sum_j (y_{ij} t_{ij})$ // execution time of T_i depends on its voltage level

$\forall T_i, T_j \in T$

- $r_{ij} = 1$ iff $\forall k = 1, \ldots, m,\ x_{ik} = x_{jk}$ // r_{ij} equals one if T_i and T_j are assigned to the same sensor node

$\forall E_i = (a,b) \in E$

- $\sum_j z_{ij} = 1$ // E_i can be assigned to exactly one channel
- $\gamma(i) \geq \beta(a)$ // E_i starts transmission after T_a completes execution

(*) $\delta(i) = \gamma(i) + t'_i(1 - r_{ab})$ // the transmission time of E_i depends on the locations of T_a and T_b

for any source tasks T_i

- $\alpha(i) \geq 0$ // all source tasks can start execution at time 0

for any sink task T_i

- $\beta(i) \leq P$ // all sink tasks must complete before the relative deadline

Fig. 1. Constraint sets 1 for the ILP formulation

Constraint set 2:

$\forall T_i, T_j \in T$, such that $i \neq j$ and $T_i || T_j$

$p_{ij} = 1 - p_{ji}$ // p_{ij} is the inverse of p_{ji}

$\alpha(j) \geq p_{ij} r_{ij} \beta(i)$ // if T_i and T_j are assigned to the same sensor node, T_i
// completes before T_j starts execution iff $p_{ij} = 1$

$\alpha(i) \geq p_{ji} r_{ij} \beta(j)$ // if T_i and T_j are assigned to the same sensor node, T_j
// completes before T_i starts execution iff $p_{ji} = 1$

$\forall E_i, E_j \in E$, such that $E_i = (a,b)$, $E_j = (a,c)$, $b \neq c$ // Communiation activities from the same sensor node

$\gamma(j) \geq q_{ij}(1 - r_{ab})(1 - r_{cd})\delta(i)$ // E_i completes before E_j starts transmission iff $q_{ij} = 1$

$\gamma(i) \geq q_{ji}(1 - r_{ab})(1 - r_{cd})\delta(j)$ // E_j completes before E_i starts transmission iff $q_{ji} = 1$

$\forall E_i, E_j \in E$, such that $E_i = (a,b)$, $E_j = (c,b)$, $a \neq c$ // Communiation activities to the same sensor node

$\gamma(j) \geq q_{ij}(1 - r_{ab})(1 - r_{cd})\delta(i)$ // E_i completes before E_j starts transmission iff $q_{ij} = 1$

$\gamma(i) \geq q_{ji}(1 - r_{ab})(1 - r_{cd})\delta(j)$ // E_j completes before E_i starts transmission iff $q_{ji} = 1$

$\forall E_i, E_j \in E$, such that $E_i = (a,b)$, $E_j = (c,d)$, $a \neq c$, $b \neq d$, and $E_i || E_j$

$q_{ij} = 1 - q_{ji}$ // q_{ij} is the inverse of q_{ji}

$s_{ij} = 1$ iff $\forall k = 1 \ldots K$, $z_{ik} = z_{jk}$ //s_{ij} equals one if E_i and E_j are assigned to the same channel

$\gamma(j) \geq q_{ij}(1 - r_{ab})(1 - r_{cd})s_{ij}\delta(i)$ // if E_i and E_j are assigned to the same channel, E_i completes
// before E_j starts transmission iff $q_{ij} = 1$

$\gamma(i) \geq q_{ji}(1 - r_{ab})(1 - r_{cd})s_{ij}\delta(j)$ // if E_i and E_j are assigned to the same channel, E_j completes
// before E_i starts transmission iff $q_{ji} = 1$

Constraint set 3:

$\forall T_i, T_j \in T$, such that T_i and T_j are source tasks and $i \neq j$

$r_{ij} = 0$ // any two source tasks cannot be assigned to the same sensor node

Fig. 2. Constraint sets 2 and 3 for the ILP formulation

Minimize $\mathcal{E}$

Subject to $\forall PE_k$ $\frac{\sum_{T_i \in T}\{x_{ik}\sum_j(y_{ij}e_{ij})\} + \sum_{E_i=(a,b)\in E}\{e'_i|x_{ak} - x_{bk}|\}}{R_k} \leq \mathcal{E}$

and Constraint sets 1, 2, and 3

Fig. 3. ILP formulation for the energy-balanced task allocation problem

```
1.  Each task is assumed to constitute a cluster by itself
2.  Set $E$ as the list of edges in a non-decreasing order of the edge weights
3.  $L \leftarrow$ Travese()
4.  **While** $E$ is not empty **Do**
5.      Remove the first edge from $E$, denoted as $(i, j)$
6.      $L' \leftarrow$ Traverse() as if $C(i)$ and $C(j)$ are merged
7.      **If** $L' < L$ and to merge $C(i)$ and $C(j)$ does not violate the task placement constraint
8.          Merge $C(i)$ and $C(j)$
9.          $L \leftarrow L'$
10. **If** $L > P$, **Return** failure
```

Fig. 4. Pseudo code for Phase 1

```
1.  Initialize $Q_{act}$
2.  Set the timestamps for all task clusters and channels to zero
3.  Append all source tasks to $Q_{act}$ with ready time set to zero
4.  **While** $Q_{act}$ is not empty **Do**
5.      Remove the first activity from $Q_{act}$
6.      **If** the removed activity is a computation activity, denoted as $T_i$
7.          Set $\alpha(i) \leftarrow \max\{$ready time of $T_i$, timestamp of $C(i)\}$
8.          Set $\beta(i)$ to the expected completion time of $T_i$, i.e., $\beta(i) \leftarrow \alpha(i) + t_{i1}$
9.          Set the timestamp of $C(i)$ to $\beta(i)$
10.         Insert all communication activities initiated by $T_i$ into $Q_{act}$ with ready time set to $\beta(i)$ in a
            non-decreasing order of their communication loads
11.     **Else**
12.         Let $E_i = (a, b)$ denote the removed communication activity
13.         Find the channel with the smallest timestamp, say the $j$-th channel
14.         Set $\gamma(i) \leftarrow \max\{$ready time of $E_i$, timestamp of the $j$-th channel$\}$
15.         Set $\delta(i)$ to the expected completion time of $E_i$, i.e., $\delta(i) \leftarrow \gamma(i) + t'_i$
16.         Set the timestamp of the $j$-th channel to $\delta(i)$
17.         Set the ready time of any unscheduled communication activities from $T_a$ to $\delta(i)$
18.         Set the ready time of any unscheduled communication activities to $T_b$ to $\delta(i)$
19.         **If** all the communication activities to $T_b$ have been scheduled
20.             Insert $T_b$ into $Q_{act}$ with ready time set to $\delta(i)$
21. **Return** the largest timestamp among all clusters
```

Fig. 5. Pseudo code for function Traverse()

```
1.  Sort $\Pi$ in a non-increasing order of the energy dissipation of clusters
2.  **While** $\Pi$ is not empty **Do**
3.      Select the first element, $\pi$, in $\Pi$
4.      Calculate the expected norm-energy for each sensor node (set to infinity if two source tasks
        are assigned to the same sensor node)
5.      Assign $\pi$ to the sensor node that gives the minimal expected norm-energy
6.      Update the norm-energy of the sensor node
7.      Remove $\pi$ from $\Pi$
8.  $L \leftarrow$ TraverseAssigned()
9.  **If** $L > P$, **Return** failure
```

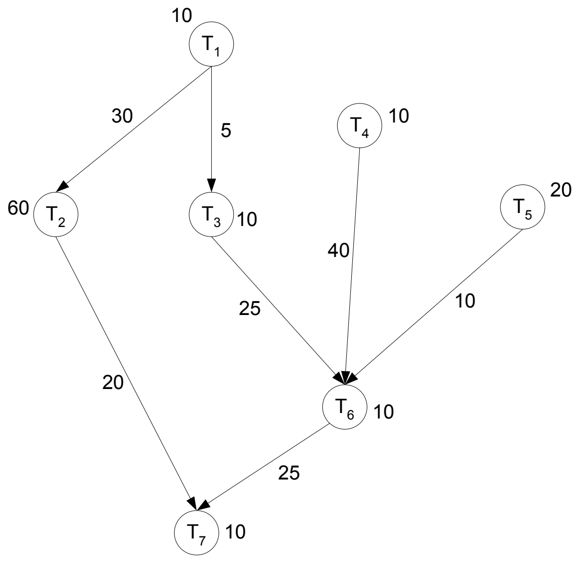
Fig. 6. Pseudo code for Phase 2

```
1.  For each $PE_i$, sort $ED_i$ in a non-increasing order
2.  **Do**
3.      $i \leftarrow 1$
4.      Let $PE_r$ denote the critical sensor node and $\mathcal{E}$ denote the norm-energy of $PE_r$
5.      **While** $i \leq |ED_r|$ **Do**
6.          Select the $i$-th item in $ED_r$; let $T_j$ denote the corresponding task
7.          **If** $L + td_j \leq P$
8.              $L \leftarrow L + td_j$
9.              Lower the voltage of $T_j$ to the next level
10.             Update $ed_j$ in $ED_r$; resort $ED_r$ if necessary
11.             Find the new critical sensor node, $PE_{r'}$; update $\mathcal{E}$
12.             **If** $r \neq r'$
13.                 $r \leftarrow r'$; $i \leftarrow 1$
14.         **Else** $i \leftarrow i + 1$
15.     $L \leftarrow$ TraverseAssigned()
16. **Until** $\mathcal{E}$ can not be reduced any more
```

Fig. 7. Pseudo code for Phase 3

```
1.  For each $PE_i$, sort $ED_i$ in a non-increasing order
2.  Do
3.      $i \leftarrow 1$
4.      Let $PE_r$ denote the critical sensor node
5.      While $i \leq |ED_r|$ Do
6.          Select the $i$-th component in $ED_r$; let $a$ denote the corresponding activity
7.          If $L + td_a > P$, $i = i + 1$
8.          Else
9.              $L \leftarrow L + td_a$
10.             If $a$ is a computatin activity
11.                 Lower the voltage level of $a$ to the next available option
12.             Else
13.                 If to lower the modulation level of $a$ to the next available option does not increase $\mathcal{E}$
14.                     Lower the modulation level of $a$ to the next available option
15.                 Else $i \leftarrow i + 1$
16.             If any voltage or modulation scaling is performed
17.                 Update $ed_a$ and $td_a$; resort $ED_r$ if necessary
18.                 Find the new critical sensor node, $PE_{r'}$; update $\mathcal{E}$
19.                 If $r \neq r'$
20.                     $r \leftarrow r'$; $i \leftarrow 1$
21.     $L \leftarrow$ TraverseAssigned()
22. Until $\mathcal{E}$ can not be reduced any more
```

Fig. 8. Pseudo code for the modified Phase 3 that incorporates modulcation scaling

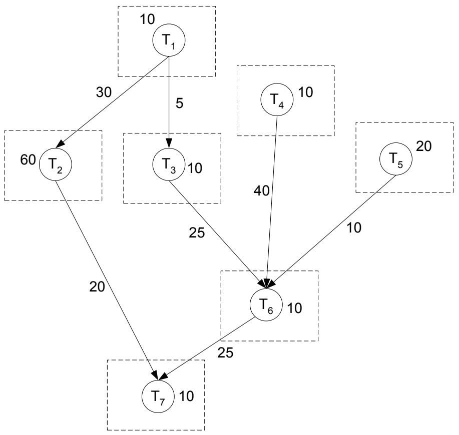


(a) application graph

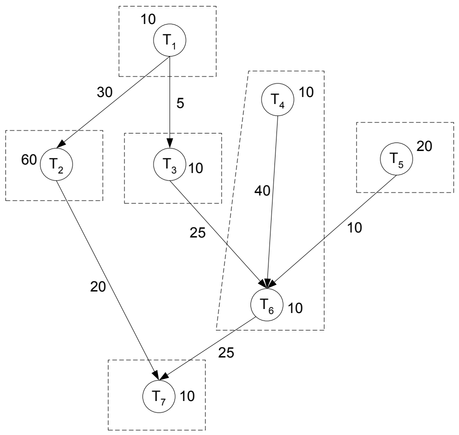
task	time cost		energy cost	
	V_h	V_l	V_h	V_l
T_1	10	33	20	6
T_2	60	199	120	36
T_3	10	33	20	6
T_4	10	33	20	6
T_5	20	66	40	12
T_6	10	33	20	6
T_7	10	33	20	6

(b) time and energy costs for executing tasks at voltage levels V_h and V_l

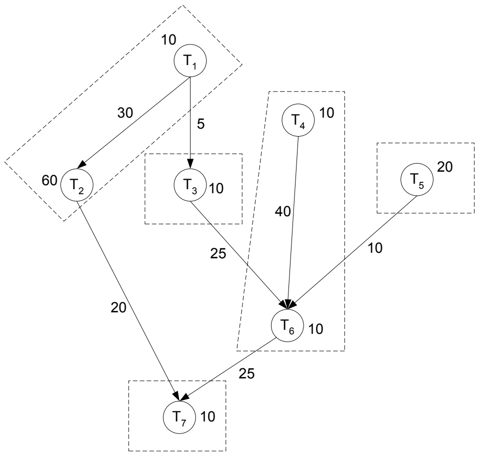
Fig. 9. An application example



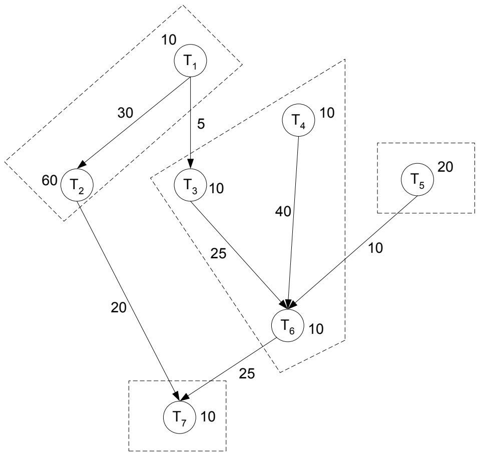
(a)



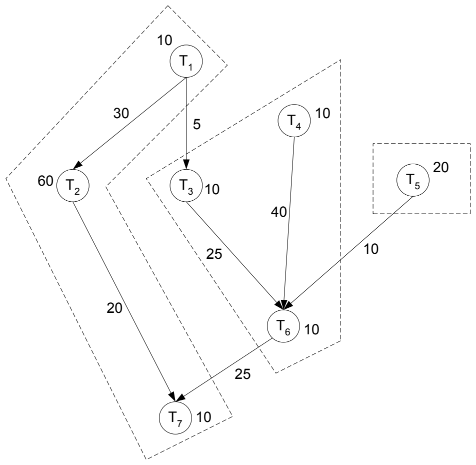
(b)



(c)



(d)



(e)

step i	edge examined	L if clustering	clustering?	L_i
0				145
1	(T_4, T_6)	135	yes	135
2	(T_1, T_2)	120	yes	120
3	(T_3, T_6)	100	yes	100
4	(T_6, T_7)	100	no	100
5	(T_2, T_7)	80	yes	80
6	(T_5, T_6)		no	80
7	(T_1, T_3)		no	80

(f)

Fig. 10. Clustering steps for the application in Figure 9

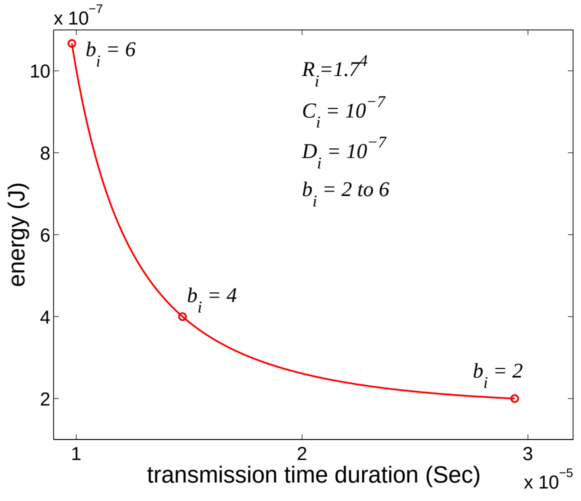
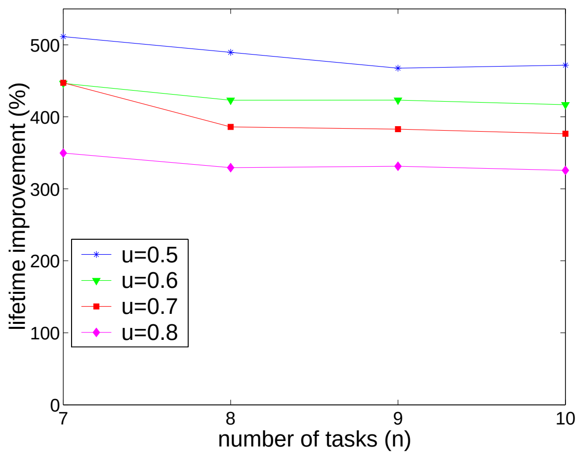
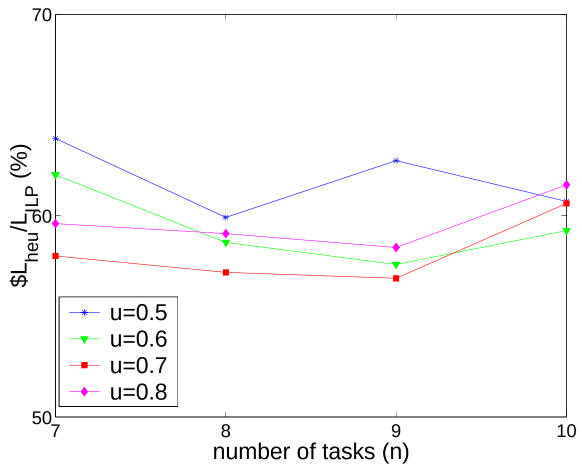


Fig. 11. Energy-latency tradeoffs for transmitting one bit of data

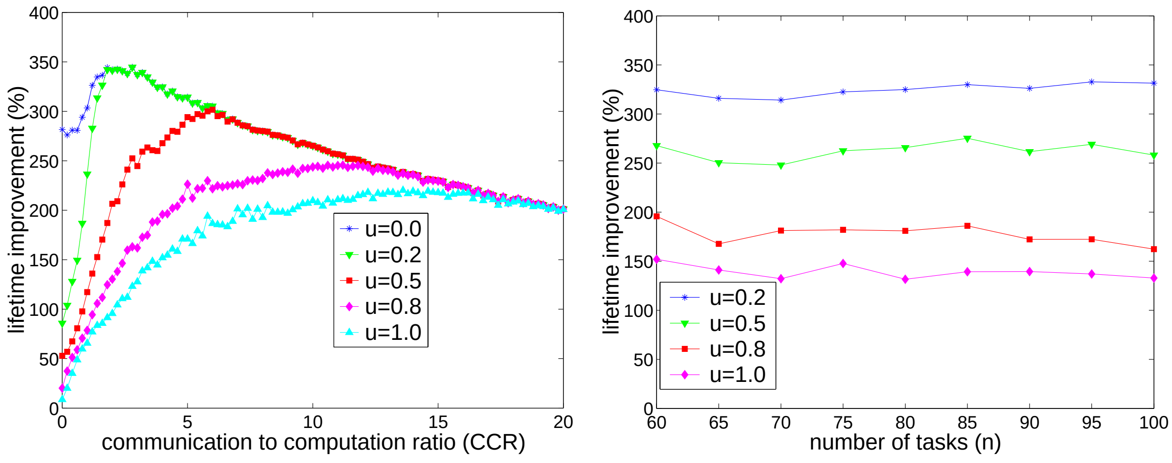


(a) lifetime improvement achieved by the ILP-based approach



(b) Performance comparison of the ILP-based approach and the 3-phase heuristic

Fig. 12. Lifetime improvement of our approaches for samll scale problems (3 sensor nodes, 3 voltage levels, 2 channels, CCR = 1)



(a) lifetime improvement vs. system utilization (u) and communication to computation ratio (CCR)

(b) lifetime improvement vs. number of tasks ($CCR = 4$)

Fig. 13. Lifetime improvement of the 3-phase heuristic for large scale problems (10 sensor nodes, 8 voltage levels, 4 channels, 60-100 tasks)

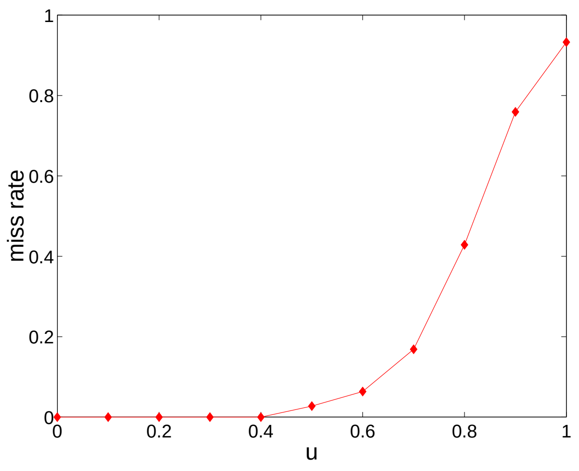


Fig. 14. Miss rate of the 3-phase heuristic (10 sensor nodes, 8 voltage levels, 4 channels, 60 tasks, $CCR = 0$)

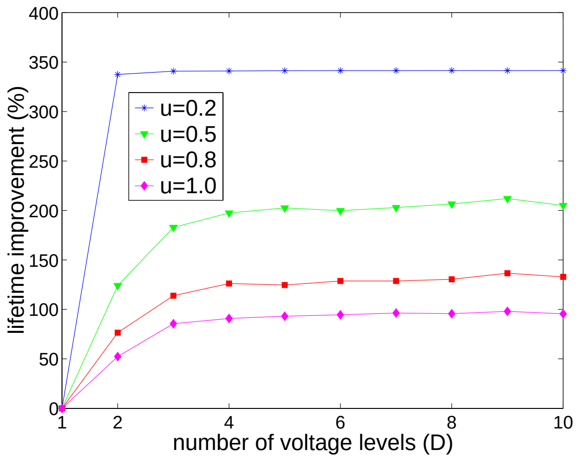
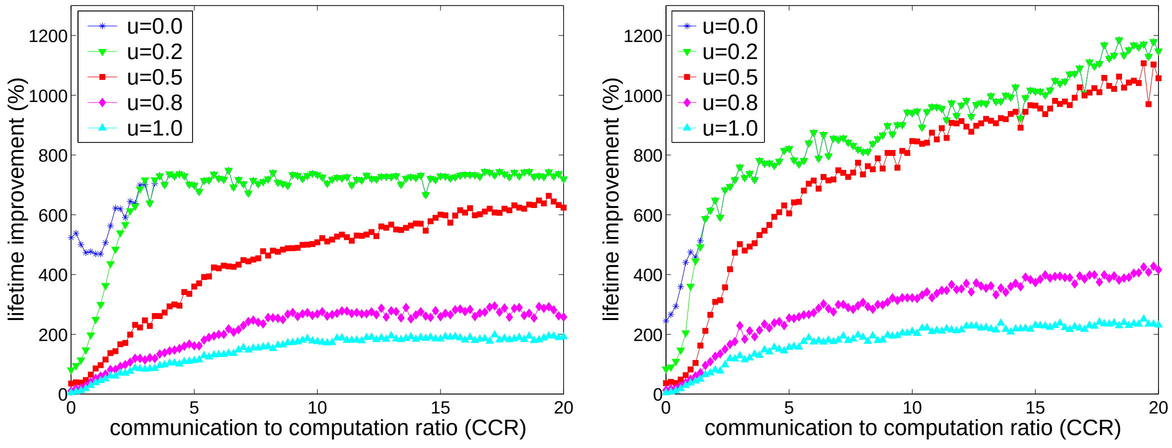


Fig. 15. Impact of variation in number of voltage levels (10 sensor nodes, 4 channels, 60 tasks, $CCR = 2$)



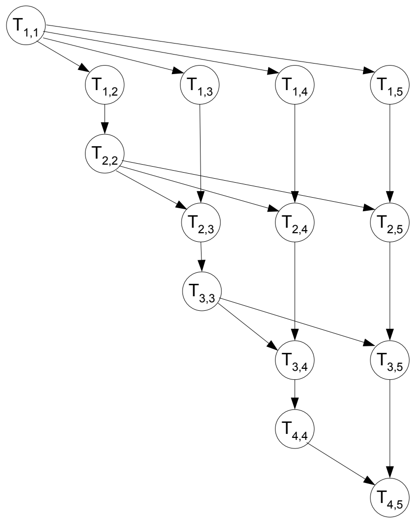
(a) small energy/time ratio for communication activities ($C_i = 10^{-7}$)

(b) large energy/time ratio for communication activities ($C_i' = 10^{-6}$)

Fig. 16. Lifetime improvement of the 3-phase heuristic incorporated with modulation scaling (10 sensor nodes, 8 voltage levels, 4 channels, 3 modulation levels, 60 tasks)

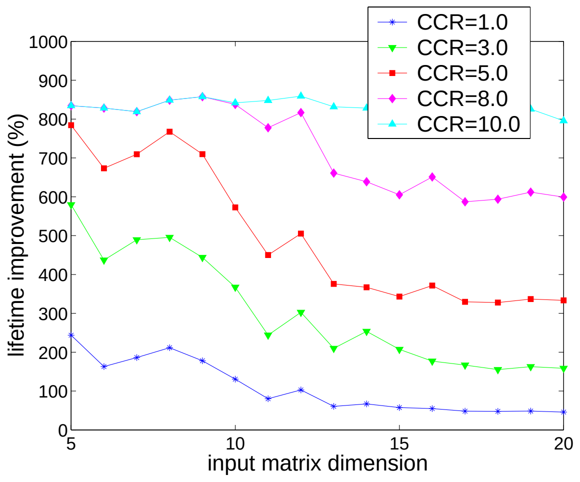
```
MatrixFactorization($a$)
1. For $k = 1$ to $s - 1$ Do
2.     For $i = k + 1$ to $s$ Do            // $T_{k,k}$
3.         $a_{ik} = a_{ik} / a_{kk}$
4.     For $j = k + 1$ to $s$ Do
5.         For $i = k + 1$ to $s$ Do        // $T_{k,j}$
6.             $a_{ij} = a_{ij} - a_{ik} / a_{kj}$
```

(a) sequential algorithm

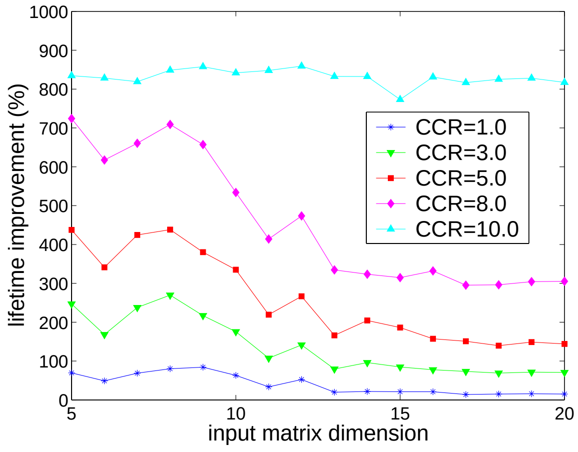


(b) example application graph with a 4×4 matrix

Fig. 17. Matrix factorization algorithm



(a) $u = 0.5$

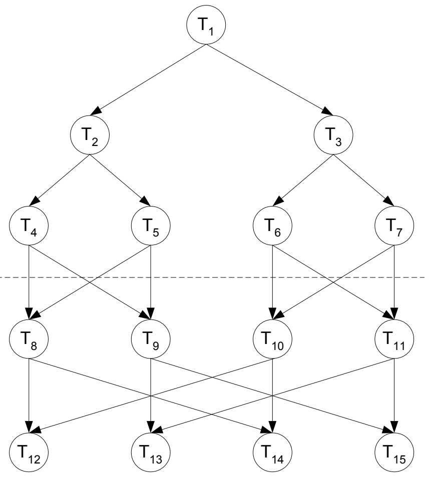


(b) $u = 0.8$

Fig. 18. Lifetime improvement for the matrix factorization algorithm (10 sensor nodes, 8 voltage levels, 4 channels, 3 modulation levels)

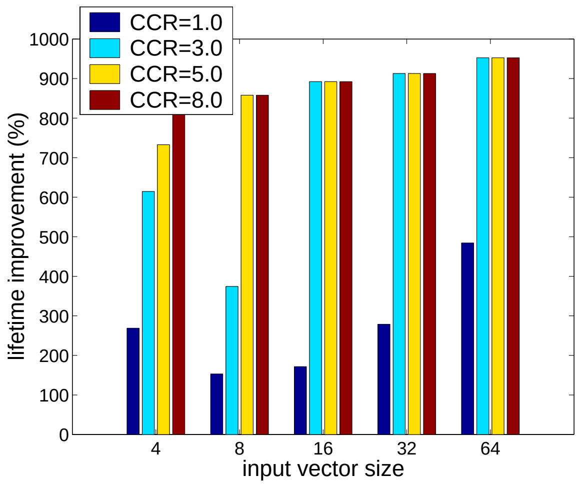
```
FFT($A$, $\omega$)
1. Set $l$ = length($A$)
2. **If** $l = 1$, return $A$
3. $Y^{(0)} = \mathrm{FFT}((A[0], A[2], \ldots, A[l-2]), \omega^2)$
4. $Y^{(1)} = \mathrm{FFT}((A[1], A[3], \ldots, A[l-1]), \omega^2)$
5. **For** $i = 0$ to $l/2 - 1$ **Do**
6.         $Y[i] = Y^{(0)}[i] + \omega^i \times Y^{(1)}[i]$
7.         $Y[i + l/2] = Y^{(0)}[i] - \omega^i \times Y^{(1)}[i]$
8. **Return** Y
```

(a) sequential algorithm

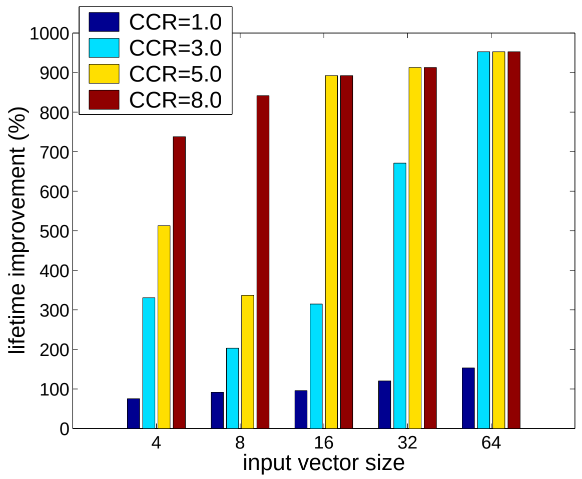


(b) example application graph with 4 points

Fig. 19. Fast Fourier Transformation (FFT) algorithm



(a) $u = 0.5$



(b) $u = 0.8$

Fig. 20. Lifetime improvement for the FFT algorithm (10 sensor nodes, 8 voltage levels, 4 channels, 3 modulation levels)