

Energy-Balanced Task Allocation for Collaborative Processing in Networked Embedded Systems*

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

ABSTRACT

In this paper, we study the problem of allocating a real-time application onto a set of homogeneous processing elements connected by a single-hop wireless network. A periodic application consisting of a set of communicating tasks is considered. Each element is equipped with discrete dynamic voltage scaling for exploring the energy-latency tradeoffs. The time and energy costs of both computation and communication activities are considered. The goal is to balance the energy dissipation of the elements during each period of the application with respect to the remaining energy of elements, such that the system lifetime is maximized. An Integer Linear Programming (ILP) formulation is first developed, which can be solved to obtain the optimal solution. We then propose an efficient 3-phase heuristic. Experimental results show that for small scale problems, the performance of the heuristic achieves up to 85% of the system lifetime obtained by the ILP-based approach. For large scale problems, the performance of the heuristic shows an improvement of 120-250% in the system lifetime compared with the case where no voltage scaling is used. Further, we present two extensions of our approaches that consider multiple communication channels and techniques for exploring the energy-latency tradeoffs of the communication activities.

Categories and Subject Descriptors

C.3 [SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS]: Real-time and embedded systems;
C.2.4 [COMPUTER-COMMUNICATION NETWORKS]: Distributed Systems—*Distributed applications*

General Terms

Algorithms

*This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-C-00-1633.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

LCTES'03, June 11–13, 2003, San Diego, California, USA.
Copyright 2003 ACM 1-58113-647-1/03/0006 ...\$5.00.

Keywords

networked embedded systems, energy-balanced, task allocation, integer linear programming

1. INTRODUCTION

Networked embedded systems (EmNets [4]) are being developed for a variety of applications, including infrastructure monitoring, habitat study, target tracking, and battlefield surveillance [4, 6]. EmNets usually contain a number of networked and embedded processing elements (or elements in short) with each element consisting of computation, communication, sensing, and actuating devices. These elements must collaborate with each other to accomplish certain applications. For instance, in the target tracking application, hundreds of elements are dispersed on the ground for tracking vehicles passing through a specific area. Elements are usually organized into clusters [9] with each cluster consisting of tens of elements. Distributed signal detection and collaborative data processing are performed within each cluster for detecting, identifying, and tracking vehicles.

Energy efficiency is a key concern for EmNets, since elements are typically powered by batteries. The large number of elements 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 application layer. However, while many hardware techniques [1, 10], network protocols [9, 12], and data processing algorithms [14, 27] have been proposed for energy-aware design, systematic mechanisms for designing energy-aware collaborative processing between elements still need to be addressed.

The state of the art in EmNets design and deployment takes an ad-hoc approach – system planning and resource management are done without a rigorous and consistent methodology. This can lead to cumbersome and inefficient deployment and utilization of the system. Our goal is to overcome these weaknesses of the existing approaches and develop technologies and tools that will automate the rapid design and deployment of EmNets. Toward such a goal, we study the problem of allocating a set of communicating tasks onto a network of embedded elements in this paper. We informally describe our problem as follows.

Energy-Balanced Task Allocation Problem: We consider a set of homogeneous elements that are connected using a single-hop wireless network. The target application consists of a set of communicating tasks. In the paper, the term

activity refers to either a computation task or a communication request between a pair of tasks. We consider a real-time scenario where the application is periodically executed with each execution being completed before the next round of execution is invoked. Such a requirement is called the *latency constraint* and is usually imposed by the nature of the application (for example, to monitor a target at a constant sampling rate). In addition, the time duration between two consecutive rounds of execution is called the *period*.

We consider the case that each element is equipped with the widely used dynamic voltage scaling (DVS) [23]. In addition, the underlying network is assumed to be capable of scheduling the transmissions of data packet according to the start time of each transmission. For instance, wireless networks employing a time-division multiple-access (TDMA) protocol have such characteristics.

Our problem is to find an allocation of tasks onto elements that includes (1) the *assignment* of tasks onto elements, (2) the *voltage settings* of tasks, and (3) the consequent *scheduling* of computation and communication activities. The objective is to balance the energy dissipation of the elements during each period of the application, such that the lifetime of the first element that dies out due to depleted energy is maximized. In addition to the above latency constraint, we also consider the *exclusive access constraint*, which forces that (1) the time periods of executing multiple computation activities on the same element do not overlap; and (2) the time periods scheduled for multiple communication activities over the single-hop wireless network do not overlap.

We present the uniqueness of our problem compared with previous efforts for task allocation in Section 2.

Our Contributions: The idea of energy-balanced task allocation in EmNets is proposed. Accordingly, a new performance metric is defined. As we shall see in Section 2, 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 EmNets, since each element is equipped with its own energy source. Moreover, for re-programmable systems or systems using event-driven strategy, the application often need to be executed after the system has been working for sometime – the remaining energy of the elements can vary from each other. Thus, an *energy-balanced* approach is needed for task allocation, such that the energy dissipation of each element during each period of the application is proportional to the remaining energy of the element. By balancing the energy dissipation across the network, the time when the first element depletes all its remaining energy can be delayed as much as possible.

To the best of the authors' knowledge, this is the first work that considers the time and energy costs of both the computation and communication activities in wireless EmNets. We develop two approaches for solving the problem. We first formulate the problem as an integer linear programming (ILP) problem. The optimal solution of the problem can be obtained by using commercial software packages, such as [20], even though the running time of such softwares can be large. In addition, the optimal solution can be used as a fair baseline for the evaluation of various heuristics. Next, we propose an efficient 3-phase heuristic. Experimental results show that for small scale problems, the performance of the heuristic achieves up to 85% of the system lifetime obtained by the ILP-based approach. For large scale problems,

the performance of the heuristic shows an improvement of 120-250% in the system lifetime compared with the case where no voltage scaling is used.

Two extensions of our approaches are presented. Multiple communication channels and techniques for exploring the energy-latency tradeoffs of communication activities are considered in the two extensions. Both techniques can be used to further reduce the energy dissipation and hence increase the system lifetime.

Paper Organization: We discuss the related work in Section 2. A formal definition of the energy-balanced task allocation problem is presented in Section 3. The ILP formulation of the problem is given in Section 4. The 3-phase heuristic is described in Section 5. Two extensions to the proposed approaches are studied in Section 6. Experimental results are shown in Section 7. Concluding remarks and a direction for future work are discussed in Section 8.

2. RELATED WORK

In general, the problem of task allocation can be categorized as part of the mission for resource management. Task allocation focuses on finding an initial allocation of resources to tasks such that the required system performance can be delivered. In such a problem, the amount and availability of system resources and the computation and communication requirements of application are known a priori. Besides task allocation, resource management also requires dynamic adaptation of resources to sustain the system performance at run-time. Variations in the task arrival time and its computation and communication requirements need to be considered in the adaptation. Although to handle such variations is important, the ability of the system to sustain an acceptable performance largely depends on the initial allocation. Further, methodologies for task allocation provide basis for developing techniques for dynamic adaptation.

The application of DVS to real-time applications was first proposed in [23]. Further efforts that consider uni-processor real-time systems include [2, 11, 17]. Recent research interests have been shifted to the allocation of real-time tasks onto systems with multiple processors (or processing elements). The most relevant works include [8, 13, 26, 28]. In [8], a list-scheduling based heuristic (LEneS) is proposed to dynamically recalculate the priority of dependent tasks. However, [8] assumes a given assignment of tasks to processors. In [13], 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, [13] assumes that the task assignment and scheduling are given. A similar problem to the one studied in this paper is investigated in [26]. A two-phase framework is presented to first determine the allocation of tasks onto elements and then the voltage settings of tasks using convex programming. In [28], a dynamic processor voltage adjustment mechanism for frame-based tasks in a homogeneous multi-processor environment is discussed. However, the time and energy costs for communication activities are not addressed in either [26] or [28].

As previously mentioned, the goal of all the above works is to minimize the overall energy dissipation of the system. While such a goal is reasonable for centralized systems, it does not capture the nature of EmNets. The reason is that to minimize the overall energy dissipation can lead to heavy

use of energy-effective elements, regardless of the remaining energy of such elements. The consequent short lifetime of such elements will very likely hinder the system from delivering required performance. This weakness is the 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 [18], wireless communication is a major source of energy dissipation in EmNets. Thus, it is important to carefully allocate the tasks in order to reduce the energy cost for communication.

Energy-balanced task allocation bears some resemblance to load-balance in parallel and distributed computing. However, the communication activities over the single-hop wireless network need to be serialized such that the time periods scheduled for distinct activities do not overlap. 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.

3. PROBLEM DEFINITION

In this section, we first formally define the system and application models. The energy-balanced task allocation problem is then described.

System Model: We consider a set of m homogeneous processing elements, $\{PE_1, PE_2, \dots, PE_m\}$, connected by a single-hop wireless network. The homogeneity refers to the same computation capability and radio power. Each element is equipped with d discrete voltage levels, listed as $\{V_1, V_2, \dots, V_d\}$ in decreasing order. Each voltage level corresponds to a specific computation speed (given in cycles per second), with SP_j denoting the speed of V_j . For ease of analysis, we use a normalized computation speed, with $SP_1 = 1$. Each element is powered by a non-rechargeable battery, with R_i denoting the remaining energy of PE_i . In addition, a non-preemptive scheduling policy is employed by each element. For ease of analysis, we assume that the processors of elements consumes zero power during idle state.

The underlying network protocol is assumed to be capable of scheduling transmissions according to the start time of each transmission. Thus, run-time contentions in the network are avoided. Let τ denote the time for transmitting one unit data packet between elements. For ease of analysis, the transmission power and the reception power of the embedded radios are considered to be equal. Hence, the transmission of one unit data packet between two elements 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 each communication activity. 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 . The data transmission between two tasks on the same element is performed through the local memory with zero time and energy costs. In addition, we assume that computation and communication activities can be parallelly executed on an element.

Note that low power paging channel mechanisms [6] can be used for synchronization between elements when the radios are shutdown. Such synchronization is necessary due to the variations in the arrival time or the computation and communication requirements of tasks at run-time. However, the modeling of the power consumption for such mechanisms is beyond the scope of this paper.

Application Model: A periodic application consisting of a set of communicating tasks is considered. Let P denote the period of the application. An instance of the application is activated at time kP , and must be completed by the relative deadline, $(k+1)P$, where $k = 0, 1, 2, \dots$.

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, $\{T_i : i = 1, \dots, n\}$, and edge set E denotes the set of directed communication activities between tasks. Every communication activity in E is represented as a tuple (i, j) , meaning 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 node with no incoming edges is called a source node, denoted by *source*. A node with no outgoing edges is called a sink node, denoted by *sink*. We assume that there is only one source node and one sink node in the DAG. Such an assumption can be easily satisfied by adding a dumb source or sink node into the DAG.

The workload of a task, T_i , is measured by the worst-case number of computation cycles required for executing the task, denoted by C_i . Thus, the execution time of T_i on voltage level V_j , t_{ij} , can be calculated as $\frac{C_i}{SP_j}$ (recall that SP_j is the computation speed corresponding to V_j). The voltage level of an element 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. The power consumption for executing a task follows a monotonically increasing and strictly convex function [3] of the computation speed, $g_i(\cdot)$, which can be represented as a polynomial function of at least second degree [3]. Hence, the energy dissipation for executing T_i on V_j , e_{ij} , can be calculated as $g_i(SP_j)t_{ij}$. Note that the exact forms of $g_i(\cdot)$ can vary from task to task due to different instruction components.

The communication load of an edge, (i, j) , is represented by its weight, $w_{(i,j)}$, in the number of data packets to be transmitted. Different edges incident at the same node may have different weights. Let $t'_{(i,j)}$ and $e'_{(i,j)}$ denote the time and energy costs of (i, j) if tasks T_i and T_j are not assigned onto the same element. We have $t'_{(i,j)} = \tau_s + \tau w_{(i,j)}$ and $e'_{(i,j)} = \varepsilon_s + \varepsilon w_{(i,j)}$.

Task Allocation: A task allocation is defined as (1) an assignment of all the tasks onto elements, (2) the setting of voltage levels for tasks, and (3) the scheduling of the computation/communication activities, specified by the start and finish time of activities. Each task can be assigned to exactly one element with a fixed voltage setting on that element. An allocation is feasible if each application instance can be completed before the corresponding relative deadline.

The *system lifetime* is defined as the time period from the beginning of the execution of the application to the time when any element fails due to depleted energy. Given a task allocation, let \mathcal{E}_i denote the energy dissipation of PE_i during each application period. The corresponding 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 system lifetime would be the time period from the beginning of the execution of the application to the time when not enough elements are alive to deliver required performance. For example, it is shown that to perform the Line-of-Bearing algorithm at an acceptable accuracy requires at least three alive elements. However, such a definition is quite application-specific. Thus, a simple but general definition of the system lifetime is adopted in this paper. Intuitively, to optimize the system lifetime with the above more complex definition, we may recursively apply the proposed optimization approaches to the resulting systems after elements die out.

Thus, our task allocation problem can be stated as:

Find an allocation of a set of communicating tasks onto a set of single-hop networked processing elements that maximizes the system lifetime.

4. INTEGER LINEAR PROGRAMMING FORMULATION

In this section, we present an ILP formulation of the task allocation problem defined in Section 3. The formulation captures the behavior of the system during one application period. This is fairly enough since the activities of the system repeat for each 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
$t'_{(i,j)}, e'_{(i,j)}$:	time and energy costs of (i, j) if T_i and T_j are not assigned to the same element
$a b$:	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 task T_i is assigned onto 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
$\{s_{ij}\}$:	a set of 0-1 variables such that s_{ij} equals one iff tasks T_i and T_j are assigned onto the same element
$\alpha(i), \beta(i)$:	the time when task T_i starts and completes execution
$\gamma(i, j), \delta(i, j)$:	the time when communication activity (i, j) starts and completes transmission
$\{z_{ij}\}$:	a set of 0-1 variables such that z_{ij} equals one iff the execution of T_i finishes before the execution of T_j starts
$\{z'_{(i,j)(i',j')}\}$:	a set of 0-1 variables such that $z'_{(i,j)(i',j')}$ equals one iff the communication activity (uv) finishes before $(u'v')$ starts

To capture the relative order imposed by the precedence constraints among the computation and communication activities, we define the Constraint set 1 shown in Figure 1. It is easy to verify that the exclusive access requirements for computation and communication activities with precedence constraints are also forced by Constraint set 1. However, for those computation and communication activities that do not have precedence constraints between them, an extra set of constraints are needed (Constraint set 2 in Figure 1).

As stated in Section 3, the system lifetime is calculated as $\min_i \{ \lfloor \frac{R_i}{E_i} \rfloor \}$. Thus, the objective function of the formula-

tion is to minimize the maximal energy dissipation among all elements within the period, normalized by their remaining energy. The energy dissipation of an element is calculated as the sum of the energy cost for executing tasks assigned on the element and the energy cost for corresponding communication activities. Let an auxiliary variable, \mathcal{E} , denote the maximal energy dissipation among all elements, normalized by their remaining energy. The complete formulation is given as follows.

Minimize \mathcal{E}

Subject to

$$\forall PE_k \quad \frac{\sum_i \{x_{ik} \sum_j (y_{ij} e_{ij})\} + \sum_{(i,j)} \{e'_{(i,j)} |x_{ik} - x_{jk}|\}}{R_k} \leq \mathcal{E}$$

and Constraint sets 1 and 2

In the above formulation, 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 onto PE_k , but not both.

Clearly, the above formulation is non-linear. Several standard linearization techniques [19] can be applied to transform the non-linear form into an ILP formulation. The resultant ILP formulation can then be solved using several existing tools, such as LINDO [20]. Due to the space limitation, we omit the details of linearization in this paper.

5. 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 options (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, clusters are assigned onto elements such that the energy dissipation of each element is proportional to the remaining energy of the element. In the last phase, the system lifetime is maximized by lowering the voltage levels of tasks. To lower the complexity of the algorithm, no backtracking is used in the heuristic – the structure of clusters does not change in phases two and three. The details of the heuristic are as follows.

Phase 1: A task cluster is defined as a set of tasks assigned onto the same element 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 elements, which means 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.

Traditional approaches for task clustering assume an environment of fully connected processors, which enables parallel communication across the system. However, in our problem, the exclusive access requirement of the single-hop wireless network requires the communication activities to be serialized. Thus, a new challenge is to select the right policies for the serialization that facilitates the reduction of the execution time of the application. We use a 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. Note that more sophisticated policies can also be applied.

Initially, every task is assumed to constitute a cluster by itself. We examine all the edges within the application DAG

Constraint set 1:

$$\begin{aligned} \forall T_i \in T & \\ \sum_j x_{ij} &= 1 & // \text{ every task can be assigned to exactly one element} \\ \sum_j y_{ij} &= 1 & // \text{ every task can be executed using exactly one voltage level} \\ \alpha(i) &\geq \max_{(j,i) \in E} \{\delta(j,i)\} & // T_i \text{ starts execution after receiving all input data} \\ \beta(i) &= \alpha(i) + \sum_j (y_{ij} t_{ij}) & // \text{ execution time of } T_i \text{ depends on its voltage level} \\ \forall (i,j) \in E & \\ s_{ij} &= 1 \text{ iff } \forall k = 1, \dots, n, x_{ik} = x_{jk} & // s_{ij} \text{ equals one if } T_i \text{ and } T_j \text{ are allocated to the same element} \\ \gamma(i,j) &\geq \beta(i) & // (i,j) \text{ starts transmission after } T_i \text{ completes execution} \\ \delta(i,j) &= \gamma(i,j) + t'_{(i,j)}(1 - s_{ij}) & // \text{ the transmission time of } (i,j) \text{ depends on the location of } T_i \text{ and } T_j \\ \alpha(\text{source}) &\geq 0 & // \text{ the source node can start execution at time 0} \\ \beta(\text{sink}) &\leq P & // \text{ the sink node must complete execution before the relative deadline} \end{aligned}$$

Constraint set 2:

$$\begin{aligned} \forall T_i, T_j \in T, \text{ such that } i \neq j \text{ and } T_i || T_j & \\ z_{ij} &= 1 - z_{ji} & // z_{ij} \text{ is the inverse of } z_{ji} \\ \alpha(j) &\geq z_{ij} s_{ij} \beta(i) & // \text{ if } T_i \text{ and } T_j \text{ are assigned to the same element, } T_i \text{ completes before} \\ & & // T_j \text{ starts execution iff } z_{ij} = 1 \\ \alpha(i) &\geq z_{ji} s_{ij} \beta(j) & // \text{ the reverse of the previous constraint} \\ \forall (i,j), (i',j') \in E, \text{ such that } (i,j) \neq (i',j') \text{ and } (i,j) || (i',j') & \\ z'_{(i,j)(i',j')} &= 1 - z'_{(i',j')(i,j)} & // z'_{(i,j)(i',j')} \text{ is the inverse of } z'_{(i',j')(i,j)} \\ \gamma(i',j') &\geq z'_{(i,j)(i',j')} (1 - s_{ij})(1 - s_{i'j'}) \delta(i,j) & // \text{ if both } (i,j) \text{ and } (i',j') \text{ are not local transmission, } (i,j) \text{ completes} \\ & & // \text{ before } (i',j') \text{ starts transmission iff } z'_{(i,j)(i',j')} = 1 \\ \gamma(i,j) &\geq z'_{(i',j')(i,j)} (1 - s_{ij})(1 - s_{i'j'}) \delta(i',j') & // \text{ the reverse of the previous constraint} \end{aligned}$$

Figure 1: Constraint sets for the ILP formulation

-
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 their weights
 3. $L = \text{Traverse}()$
 4. **While** E is not empty **Do**
 5. Remove the first edge from E , denoted as (i,j)
 6. $L' = \text{Traverse}()$ as if $C(i)$ and $C(j)$ are merged
 7. **If** $L' < L$
 8. Merge $C(i)$ and $C(j)$
 9. $L = L'$
 10. **If** $L > P$, return failure
-

Figure 2: Pseudo code for Phase 1

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 the cluster containing T_i with the one containing T_j , we perform the merge. Otherwise, T_i and T_j remain in two different clusters. The pseudo code for Phase 1 is shown in Figure 2. In the code, L denotes the overall execution time of the application and $C(i)$ denotes the cluster that contains task T_i . In line 3 and 6, the function $\text{Traverse}()$ is called to traverse the DAG in order to determine the schedule of the tasks and hence the execution time of the application.

The pseudo code of the function $\text{Traverse}()$ is shown in Figure 3. In the code, we maintain a queue of activity, Q_{act} , that stores all the ready computation or communication activities in their expected execution order. Initially, only the source node is appended to Q_{act} with ready time 0. In line 6,

-
1. Initialize Q_{act} and the timestamps
 2. append source to Q_{act}
 3. **While** Q_{act} is not empty **Do**
 4. Remove the first activity, a , from Q_{act}
 5. **If** a is a computation activity
 6. Schedule the execution of a in the cluster that contains a ; update the timestamp accordingly
 7. Append all communication activities initiated by a to Q_{act}
 8. **Else**
 9. Schedule a ; update TS accordingly
 10. **If** the destination task of a has received all incoming packets, append the task to Q_{act}
 11. **Return** the largest timestamp among all clusters
-

Figure 3: Pseudo code for function $\text{Traverse}()$

to schedule the execution of a , we maintain a timestamp for each cluster that indicates the finish time for all scheduled tasks contained by the cluster. In addition, a timestamp, TS , with initial value 0 indicates the nearest available time of the wireless network. TS is updated every time a communication activity is scheduled in line 9.

Phase 2: In this phase, an energy-balanced assignment of task clusters obtained in Phase 1 onto the elements is determined. Note that multiple clusters can be assigned to the same element. Based on the contained tasks and the corresponding communication activities, it is easy to calculate the energy dissipation of each cluster. Let $\Pi = [\pi_1, \pi_2, \dots, \pi_c]$

denote the list of clusters and ξ_i denote the energy dissipation of cluster π_i . To capture the fact that the remaining energy of elements can vary from each other, we generate a matrix Γ of size $c \times m$ with element $\Gamma[i][j]$ set to $\frac{\xi_i}{R_j}$, which corresponds to the energy dissipation of cluster π_i normalized by the remaining energy of PE_j .

The normalized energy dissipation of an element is given as the sum of the normalized energy dissipation of the clusters assigned to the element. Let \mathcal{E} denote the maximum of the normalized energy dissipation of all elements. Clusters are checked in a non-increasing order of their energy dissipation and assigned to the element that minimizes \mathcal{E} . If two clusters are assigned onto the same element, the energy cost of communication activities between them becomes zero. The pseudo code of Phase 2 is shown in Figure 4. In the code, function `TraverseAssigned()` is used to find the execution time of the application based on the obtained assignment of clusters. Compared with `Traverse()`, the modification in `TraverseAssigned()` is that in line 6 of Figure 3, each computation activity is scheduled on the element that it is assigned to. Thus, a timestamp is maintained for each element, instead of each cluster.

-
1. Sort Π in a non-increasing order of the energy dissipation of clusters
 2. $\mathcal{E} = 0$
 3. **While** Π is not empty **Do**
 4. Select the first element, π , in Π
 5. Based on Γ , select the element for assigning π , such that \mathcal{E} is minimized
 6. Update \mathcal{E}
 7. Remove π from Π
 8. $L = \text{TraverseAssigned}()$
 9. If $L > P$, return failure
-

Figure 4: Pseudo code for Phase 2

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. In each iteration, we find the task that by lowering its current voltage level to the next level, the value of \mathcal{E} can be decreased the most. The extra latency caused by lowering the voltage is added to L . However, since the schedule of communication activities can be changed by the extra latency, the value of L is recomputed by traversing the DAG every time it reaches P .

The pseudo code of Phase 3 is shown in Figure 5. In the code, ed_i denotes the energy gain by lowering the current voltage of T_i to the next level, while td_i denotes the corresponding amount of extra latency. The array composed by ed_i 's of all tasks assigned to PE_j is denoted as ED_j . Furthermore, the element with its normalized energy dissipation equal to \mathcal{E} is called the *critical element*.

Time Complexity Analysis: Let e denote the number of edges in the application DAG. In Phase 1 (Figure 2), the **While** iteration is executed e times. To traverse the DAG in line 6 takes $O(n + e)$ time. Thus, the time complexity of Phase 1 is $O(e(n + e))$. In Phase 2 (Figure 4), 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

-
1. For each PE_i , sort ED_i in a non-increasing order
 2. **Do**
 3. $i = 1$
 4. Let PE_r denote the critical element
 5. **While** $i \leq |ED_r|$ **Do**
 6. Select the i -th component in ED_r ; let T_j denote the corresponding task
 7. **If** $L + td_j \leq P$
 8. $L = L + td_j$
 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 element, $PE_{r'}$
 12. **If** $r \neq r'$
 13. $r = r'$; $i = 1$
 14. Update \mathcal{E}
 15. **Else** $i = i + 1$
 16. $L = \text{TraverseAssigned}()$
 17. **Until** \mathcal{E} can not be reduced any more
-

Figure 5: Pseudo code for Phase 3

compared in line 5. The traverse in line 8 takes $O(n + e)$ time. Therefore, Phase 2 takes $O(c \log c + mc + n + e)$ time. In Phase 3 (Figure 5), 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 takes $O(\log n)$ time. Let p denote the number for calling function `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 is observed during experiments that p usually equals 1 or 2. We conclude that the time complexity of the 3-phase heuristic is $O((e + p)(n + e) + mc + dn \log n + c \log c)$, which can be $O(dn(n + e + \log n) + e^2 + mn)$ in the worst case.

6. EXTENSIONS OF OUR APPROACHES

So far, we have defined the energy-balanced task allocation problem and proposed two approaches for solving the problem. In this section, we present two extensions of our approaches that covers a broader range of problems.

6.1 Using Multiple Communication Channels

In many real situations, multiple wireless channels are available for communication among elements. Though at any time, each element can send or receive data using at most one channel, multiple packets can be simultaneously transferred over different channels. By doing so, we may reduce the overall time cost for communication activities and provide more opportunities to trade computation energy with increased latency. In addition, when multiple channels are available, the exclusive access constraint on communication is expressed as that for each channel of the network, the time duration of different communication activities scheduled on that channel cannot overlap.

We now give the modified ILP formulation. Let l denote the number of channels. We first define a set of 0-1 variables, $\{v_{(i,j),k}\}$, such that $v_{(i,j),k}$ equals one if edge (i, j) is scheduled on the k -th channel, and zero otherwise. Further, we define another set of 0-1 variables, $\{q_{(i,j),(i',j')}\}$, such that $q_{(i,j),(i',j')}$ equals one if the edge (i, j) and edge (i', j')

$\forall (i, j), (i', j') \in E$, such that $(i, j) \neq (i', j')$ and $(i, j) \parallel (i', j')$ $z'_{(i,j)(i',j')} = 1 - z'_{(i',j')(i,j)}$ $q_{(i,j),(i',j')} = 1$ iff $\forall k = 1 \dots l$, $v_{(i,j),k} = v_{(i',j'),k}$ $\gamma(i', j') \geq z'_{(i,j)(i',j')} (1 - s_{ij}) (1 - s_{i'j'}) q_{(i,j),(i',j')} \delta(i, j)$ $\gamma(i, j) \geq z'_{(i',j')(i,j)} (1 - s_{ij}) (1 - s_{i'j'}) q_{(i,j),(i',j')} \delta(i', j')$	// $z'_{(i,j)(i',j')}$ is the inverse of $z'_{(i',j')(i,j)}$ // $q_{(i,j),(i',j')}$ equals one if (i, j) and (i', j') are scheduled on the // same channel // if (i, j) and (i', j') are scheduled on the same channel, // (i, j) completes before (i', j') starts transmission // iff $z'_{(i,j)(i',j')} = 1$ // the reverse of the previous constraint
---	--

Figure 6: Modified constraint sets for the task allocation problem using multiple channels

are scheduled on the same channel, and zero otherwise. To capture the exclusive access constraint for each channel, we replace the last three constraints in Constraint set 2 (Figure 1) using the constraint sets in Figure 6.

For the 3-phase heuristic, we need to modify the functions `Traverse()` and `TraverseAssigned()` to capture the multiple channel network. Specifically, we need to maintain a timestamp for each channel that indicates the nearest available time for the channel. In line 9 of Figure 3, a communication activity is scheduled onto the channel with the smallest timestamp. The timestamp of the selected channel is then updated accordingly.

6.2 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 [7] 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 [16] have been proposed for implementing such tradeoffs. Further, algorithms [7, 24] for applying such techniques in the context of packet transmissions or data gathering in wireless networks have been studied.

Our approaches can be extended to incorporate the above tradeoffs. Intuitively, the way we determine the DVS settings for computation activities can be modified to determine the configurations for communication activities. In the following, we briefly discuss the modifications through the example of modulation scaling.

To modify our ILP formulation, a set of variables are needed to indicate the modulation level of the communication activities. Moreover, a set of constraints are needed to determine the transmission time of communication activities based on the selected modulation levels. For the 3-phase heuristic, we need to modify Phase 3, such that the energy savings achieved by lowering the modulation levels of communication activities are also examined. Thus, a uniform framework for exploring the energy-latency tradeoffs of both computation and communication activities can be achieved [25].

7. EXPERIMENTAL RESULTS

A simulator based on the system and application models presented in Section 3 was developed to evaluate the

performance of our approach using synthetic task sets. The goals of our experiments 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 by assuming no voltage scaling is available (i.e., every element is running 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.

7.1 Experimental Setup

In the experiments, the maximum computation speed of each element was set to 1.0 Mcps (million cycles per second) and the minimum speed was set to 0.3 Mcps. It is assumed that other levels of computation speed are distributed uniformly between the maximum and minimum speeds. For example, if there are four voltage levels in total, the two middle levels of the computation speed are set to 0.77Mcps and 0.53Mcps. The remaining energy of elements follows a uniform distribution between $E_{mean}(1 \pm 0.3)$, where E_{mean} is a fairly large number.

The structure of the application DAG was generated using a method similar to the one described in [5]. The only difference is that we enforce exactly one sink node in the generation of the DAG. The computation requirements of the tasks followed a gamma distribution with a mean value $\mu_c = 2000$ and a standard deviation of 1000. The energy function of task T_i , $g_i(SP)$, was of the form $a_i \cdot SP^{b_i}$, where a_i and b_i were random variables with uniform distribution between 2 and 10, and 2 and 3 [15], respectively.

The time and energy costs of communication activities are determined by the number of packets for transmission and the value of τ and ε . Based on [22], the power of a Lucent Orinoco card with 11 Mbps bandwidth is 1.49 W. By assuming that each packet consists of 100 bits, τ and ε were set to be 0.1 mSec and 0.15 mJ, respectively. 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 packets per communication activity follows a

uniform distribution between $\frac{\mu_c}{100} CCR(1 \pm 0.2)$, where CCR is a parameter indicating the ratio of the average execution time of communication activities to the average execution time of computation activities. Intuitively, a larger value of CCR implies heavier communication loads compared with the computation loads. In our experiments, CCR was set to 0.5, 1.0, and 1.5 for different instances.

The period of the application, P , was generated based on the structure of the DAG and the load of computation/communication activities. We first define the *distance* of a node as the number of edges in the longest path from the source to the node. Nodes in the DAG are then divided into layers, with nodes in each layer having the same value of distance. The computation time of a layer is then calculated as $\mu_c \lceil \frac{p}{m} \rceil$, where p is the number of tasks in the layer. By doing so, we implicitly assume full parallelism for executing the tasks at each layer. In addition, the number of communication requests of a task is estimated as $(outdegree-1) + (indegree-1)$. The corresponding time cost is estimated as the above number of requests times $\frac{\mu_c}{100} CCR$. The communication time of a layer is calculated as the sum of the estimated communication time of all tasks at the layer. P is then set to the sum of the computation and communication time cost of all layers times $\frac{1}{u}$, where u is a parameter that approximates the overall utilization of the system. The value of u is important to the amount of energy saving that can be achieved by our approaches. Intuitively, a larger value of u implies a tighter latency constraint and thus less opportunities to trade energy for increased latency. In our experiments, the value of u was set to 0.5, 0.6, 0.7, and 0.8 for different instances.

7.2 Results

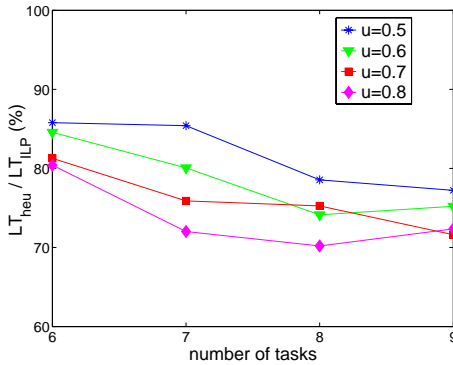
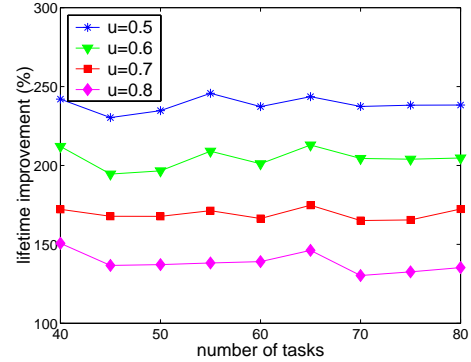


Figure 7: Performance comparison of the ILP-based approach and the 3-phase heuristic ($n = 3$, $d = 3$, $CCR = 1.0$)

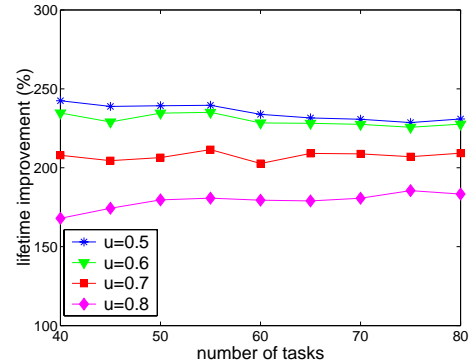
Small Scale Problems: For small scale problems with $n = 3$, $d = 3$, and $m = 5 - 9$, we compare the performance of the 3-phase heuristic against that of the ILP-based approach. Accordingly, the evaluation metric is calculated as $\frac{L_{T_{heu}}}{L_{T_{ILP}}}$. A commercial software package, LINDO [20], 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 by LINDO was returned.

The data shown in Figure 7 is averaged over more than

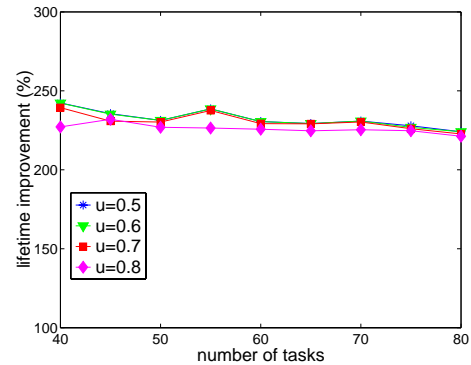
40 instances such that each data point has a 95% confidence interval with a 10% precision. From the plots, we can see that the 3-phase heuristic achieved up to 85% of the solution obtained by the ILP-based approach for the conducted experiments. While the running time of the heuristic is around zero, the average running time of the ILP-based approach ranges from 30 Sec ($n = 6$, $u = 0.5$) to 2830 Sec ($n = 9$, $u = 0.8$) on a Sun Blade1000 machine with a UltraSparc III 750 Mhz CPU.



(a) $CCR = 0.5$



(b) $CCR = 1.0$



(c) $CCR = 1.5$

Figure 8: Lifetime improvement of the 3-phase heuristic, $m = 10$, $d = 10$

Large Scale Problems: A set of experiments were conducted for evaluating the performance of the 3-phase heuristic for large scale problems, with $m = 10$, $d = 10$, and n varying from 40 to 80 in increments of 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 $\frac{LT_{heuristic}}{LT_{raw}} - 1$. The data for three different values of CCR (0.5, 1.0, and 1.5) is shown in Figure 8. The data was averaged over more than 100 instances such that each data point has a 95% confidence interval with a 10% (or better) precision.

An improvement of 120 – 250% in the system lifetime can be observed from the plots. It can be observed that the improvement decreases when the system utilization u increases. This is because the latency laxity decreases when u increases, which reduces the opportunities for trading energy for latency. However, the effect of u becomes weak when CCR increases. This is because for the same value of u , the latency laxity actually increases as CCR increases, due to our procedure to calculate the period P (described in Section 7.1). When $CCR = 1.5$, the laxity is large enough such that the voltage settings for almost all tasks can be set to the lowest level even if $u = 0.8$. Thus, variations in u have almost no impact on the performance of the heuristic.

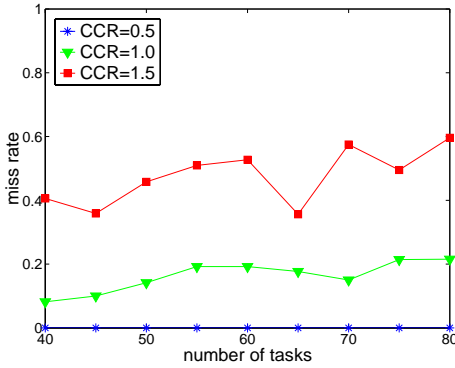


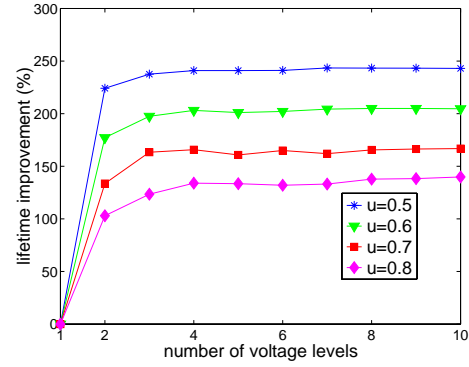
Figure 9: Miss rate of the 3-phase heuristic, $m = 10$, $d = 10$, $u = 0.8$

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 to consider. Note that in the experiments, not all instances are guaranteed to have feasible solutions. We observed that the miss rate of the 3-phase heuristic increases when the value of u or CCR increases. For instances with $u \leq 0.7$, the miss rate is under 0.2. However, the miss rate increase to 0.6 for instances with $u = 0.8$ and $CCR = 1.5$ (Figure 9). We conclude that the heuristic is unsuitable in situations when both u and CCR are high. Moreover, the running time of the heuristic is less than 2 mSec on a Sun Blade1000 machine with a UltraSparc III 750 Mhz CPU.

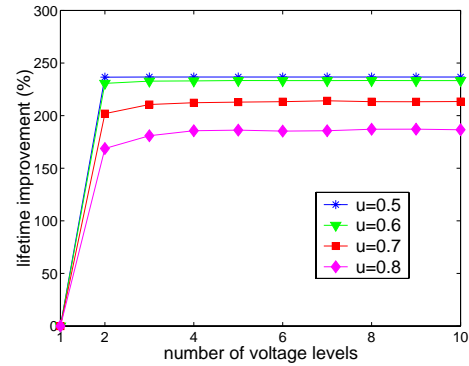
Impact of the Number of Voltage Levels: A set of experiments were conducted to investigate the impact of variation in the number of voltage levels on the performance of the 3-phase heuristic. The data obtained with $m = 10$, $m = 60$, and d varying from 1 – 12 in increments of 1 is shown in Figure 10. The data was averaged over more than 100 instances such that each data point has a 95% confidence interval with a 10% (or better) precision.

The plots show that when $CCR = 0.5$, 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, which is proportional to the voltage level. The first derivative of the energy function tends to ∞ when the voltage tends to ∞ . Thus, the most portion of energy saving is achieved by changing the voltage level from the highest option to some lower options, which can be sufficiently achieved with 4 voltage levels per element. Further energy saving by increasing the voltage levels is insignificant.



(a) $CCR = 0.5$



(b) $CCR = 1.0$



(c) $CCR = 1.5$

Figure 10: Impact of variation in number of voltage levels, $m = 10$, $n = 60$

It is quite surprising that when $CCR = 1.5$, there is almost no advantage to increase the number of voltage levels beyond 2. This is because when $CCR = 1.5$, the latency laxity is so large that the voltage settings of almost all tasks can be set to the lowest level. Thus, there is no effect to further increase the number of voltage levels.

8. CONCLUDING REMARKS

This paper discussed the problem of allocating a real-time application in EmNets. A new performance metric was proposed to balance the energy dissipation of elements during each period of the application, with respect to the remaining energy of the elements. The problem was first formulated as an ILP problem that can be solved to obtain optimal solutions. An efficient 3-phase heuristic was then developed. We demonstrated that for small scale problems, the performance of the 3-phase heuristic achieves up to 85% of the system lifetime obtained by the ILP-based approach. For large scale problems, a 120 – 250% improvement in the system lifetime was also observed. Further, we presented two extensions of our approaches that consider multiple communication channels and techniques for exploring the energy-latency tradeoffs of communication activities.

In the future, we plan to validate our techniques using real-life applications for EmNets. While it is informative to apply our techniques for several kernel operations, such as Fast Fourier Transformation and matrix multiplication, we are particularly interested in system-level task models for advanced EmNets applications. Task allocation in multi-hop EmNets is another important problem. Compared with single-hop connection, the communication activities are challenging to model and schedule in multi-hop networks.

Algorithmic techniques for optimizing application performance in wirelessly networked embedded systems can be found in [21].

9. 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 Conf. on Real-Time Systems*, 2001.
- [3] 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.
- [4] Computer Science and Telecommunications Board, *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*, National Academy Press, 2001.
- [5] R. P. Dick, D. L. Rhodes, and W. Wolf. Tgff: Task graphs for free. In *Intl. Workshop Hardware/Software Codesign*, pages 97–101, Mar. 1998.
- [6] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Intl. Conf. on Acoustics, Speech and Signal Processing*, May 2001.
- [7] 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.
- [8] F. Gruian and K. Kuchcinski. LEneS: task scheduling for low-energy systems using variable supply voltage processors. In *38th Design Automation Conf.*, pages 449–455, 2001.
- [9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application specific protocol architecture for wireless microsensor networks. *IEEE Transaction on Wireless Networking*, 2002.
- [10] J. Hill, R. Szwedczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.
- [11] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *19th IEEE Real-Time Systems Symposium*, Dec. 1998.
- [12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM MobiCom*, 2000.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] C. Schurgers, O. Aberhorne, and M. Srivastava. Modulation scaling for energy-aware communication systems. In *ISLPED*, pages 96–99, 2001.
- [17] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Intl. Conf. on Computer-Aided Design*, pages 365–368, 2000.
- [18] M. Singh and V. K. Prasanna. System level energy tradeoffs for collaborative computation in wireless networks. In *IEEE IMPACCT Workshop*, May 2002.
- [19] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, Ltd, 1999.
- [20] The LINDO System Inc. <http://www.lindo.com>.
- [21] The PACMAN Project. <http://pacman.usc.edu>.
- [22] The Agere System Inc. <http://www.wavelan.com>.
- [23] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.
- [24] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Packet scheduling algorithms for data gathering in wireless sensor networks. submitted to SenSys 2003.
- [25] Y. Yu and V. K. Prasanna. A uniform framework for exploring the energy-latency tradeoffs in networked embedded systems. Technical report. in preparation.
- [26] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *39th Design Automation Conf.*, 2002.
- [27] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, Mar. 2002.
- [28] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *22nd IEEE Real-Time Systems Symposium*, Dec. 2001.