An Energy-Delay Tunable Task Allocation Strategy for Collaborative Applications in Networked Embedded Systems

Tao Xie, *Member*, *IEEE*, and Xiao Qin, *Member*, *IEEE*

Abstract—Collaborative applications with energy and low-delay constraints are emerging in various networked embedded systems like wireless sensor networks and multimedia terminals. Conventional energy-aware task allocation schemes developed for collaborative applications only concentrated on energy savings when making allocation decisions. Consequently, the length of the schedules generated by such allocation schemes could be very long, which is unfavorable or, in some situations, even not tolerated. To remedy this problem, we developed a novel task allocation strategy called Balanced Energy-Aware Task Allocation (BEATA) for collaborative applications running on heterogeneous networked embedded systems. The BEATA algorithm aims at blending an energy-delay efficiency scheme with task allocations, thereby making the best trade-offs between energy savings and schedule lengths. Aside from that, we introduced the concept of an energy-adaptive window, which is a critical parameter in the BEATA strategy. By fine-tuning the size of the energy-adaptive window, users can readily customize BEATA to meet their specific energy-delay trade-off needs imposed by applications. Further, we built a mathematical model to approximate the energy consumption caused by both computation and communication activities. Experimental results show that BEATA significantly improves the performance of embedded systems in terms of energy savings and schedule length over existing allocation schemes.

Index Terms—Collaborative applications, energy conservation, energy latency trade-off, heterogeneous embedded systems, task allocation.

1 Introduction

Networked embedded systems like wireless sensor networks and multimedia terminals have increasingly become popular as platforms for executing collaborative applications such as target tracking and infrastructure monitoring since the last decade [2], [8], [12], [17], [22]. Much of this trend can be attributed to rapid advances in processing power, network bandwidth, and storage capacity. A collaborative application in its general form consists of a number of tasks cooperating with each other through communications to fulfill a common mission [3], [15], [18], [28], [32]. A typical example of collaborative applications is a cluster-based target tracking sensor network [32], where all sensor nodes in one cluster transfer parameters, which are sampled from a surrounding environment, to a gateway node via radio transmitters. The gateway node performs the fusion of the received data and then sends reports generated through the fusion of sensor readings to a command node. The command node carries out the system-level fusion of collected reports from multiple clusters for tracking and identifying the target [32]. A collaborative application can be symbolized as a task graph, which is generally represented by a directed acyclic graph (DAG) [30], [31], [35] that is assumed to be known a priori. Efficient allocation of these task graphs considering both computation and communication overhead onto embedded systems plays a vital role in energy conservation and performance improvement [3], [5], [35].

Energy conservation must be achieved for embedded systems because they usually have low power capacities operating in distributed mobile or wired environments [1], [11], [17], [19], [20], [23]. Meanwhile, it is desirable and, in many cases, mandatory to provide collaborative applications with high performance in terms of throughput or response time [4], [9], [13], [15], [34]. For example, a real-time military surveillance application demands a low latency in addition to significant energy savings because tardy responses make the application worthless [33]. Therefore, a task allocation scheme that merely aims at reducing energy dissipation is inadequate for many collaborative applications with low-delay requirements as it is most likely to result in unacceptable latencies.

Energy savings and low delay, however, are two conflicting objectives in the context of allocating a collaborative application represented by a task graph onto a set of connected processing nodes in a heterogeneous networked embedded system. It becomes more challenging to solve the energy-delay dilemma in heterogeneous embedded systems, mainly because of a multidimensional heterogeneity bearing by networked embedded systems that are heterogeneous in nature (see Sections 3.1 and 3.4). When it comes to a heterogeneous embedded system, a processing node providing a task with the earliest finish time may not be an ideal candidate for energy conservation.

- *T. Xie is with the Department of Computer Science, San Diego State University, 5500 Campanile Drive, GMCS 544, San Diego, CA 92182. E-mail: xie@cs.sdsu.edu.*
- *X. Qin is with the Department of Computer Science and Software Engineering, Dunstan Hall, Auburn University, Auburn, AL 36849-5347. E-mail: xqin@auburn.edu.*

Manuscript received 12 Nov. 2006; revised 27 Apr. 2007; accepted 26 July 2007; published online 29 Aug. 2007.
Recommended for acceptance by S. Nikoletseas.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0431-1106.
Digital Object Identifier no. 10.1109/TC.2007.70809.

This statement is arguably true and the reason is threefold. First, the finish time of a task allocated to a node largely depends on the task's execution time. Second, the energy consumption incurred by the task is a product of the node's energy consumption rate and the task's execution time. Third, the node's energy consumption rate and the task's execution time are independent of one another. Therefore, two unfavorable scenarios are possible: 1) A node offering a task the earliest finish time could lead to high energy consumption caused by the task's execution, although the execution time is short, and 2) a node presenting a task the least energy consumption could result in a late finish time for the task, which, in turn, may give rise to a long delay. The objective of this study is to solve the energy-delay dilemma that exists in heterogeneous embedded systems, where energy conservation and delay time optimization are equally crucial. In this paper, we aim at minimizing the energy consumption of heterogeneous embedded systems while confining schedule lengths through task allocations. To be specific, we develop an energy-delay balanced heuristic task allocation strategy or Balanced Energy-Aware Task Allocation (BEATA), where the concept of an energy-adaptive window (EAW) is introduced. Users can readily adjust trade-offs between energy consumption and schedule length by fine-tuning the size of the EAW. Experimental results demonstrate that our scheme can provide significant energy savings while achieving reasonably high performance for heterogeneous embedded systems.

Our main contributions include

1. developing an energy-delay-driven task allocation strategy called BEATA for collaborative applications running in heterogeneous networked embedded systems,
2. constructing an energy consumption model for quantitatively measuring the energy caused by both computation and communications,
3. extending a heterogeneity model to reveal an inherent nature of a heterogeneous embedded system and its impact on the system performance, and
4. simulating a heterogeneous embedded system where the BEATA strategy is implemented and evaluated.

The rest of this paper is organized as follows: In Section 2, we discuss related work. In Section 3, we build the system model, task model, energy consumption model, and heterogeneity model. Section 4 presents the BEATA scheme and introduces two alternative algorithms for collaborative applications in heterogeneous embedded systems. In addition, the optimality of BEATA is proven in Section 4. In Section 5, we evaluate the performance of BEATA based on synthetic benchmarks and a real-world application. Section 6 concludes this paper with a summary and future directions.

2 Related Work and Motivation

Numerous studies over the past decade have been conducted to reduce the overall energy consumption for a variety of embedded systems by using diverse techniques [12], [17], [19], [26], [27], [35], [36]. Source code optimization and profiling were exploited by Simunic et al. to minimize energy consumption in embedded systems [27]. Zhu et al. devised a mechanism to make use of slack-time reclamation to increase reliability and reduce energy dissipation of real-time embedded systems [36]. Park et al. studied a way of making trade-offs between energy efficiency and fairness in multiresource for multitasks in embedded systems [19]. Mohanty and Prasanna proposed a hierarchical approach to improving the energy efficiency of heterogeneous embedded systems [17]. Yu and Prasanna developed an energy-balanced task allocation scheme for collaborative processing in a homogeneous WSN, with a goal of maximizing the lifetime of the entire system [35].

Although many research results on energy conservation have been reported since the early 1990s, only a few pieces of work studying ways of making energy-delay trade-offs have emerged in the literature very recently. In these studies, a diverse array of energy-delay-driven protocols were proposed for WSNs [4], [6], [11], [15], [25], [34]. Ammari and Das devised a data dissemination protocol to realize a trade-off between energy savings and source-to-sink delay so as to increase the lifetime of the WSNs while receiving sensed data in a timely fashion [4]. Boukerche et al. investigated a protocol named Ordering by Confirmation (OBC) for event ordering in wireless sensor and actor networks (WSANs) [6]. The OBC protocol can save more energy and achieve lower latencies to meet the needs of critical conditions monitoring applications [6]. An energy-aware dynamic task allocation algorithm for sequential tasks, with a goal of time-energy efficiency over MANETs, was proposed by Lu et al. [11]. Miller et al. studied the energy-latency-reliability trade-off for broadcast in multi-hop WSNs by presenting a protocol called Probability-Based Broadcast Forwarding (PBBF) [16]. Schurgers et al. [25] proposed a new technique, called sparse topology and energy management (STEM), which efficiently wakes up nodes from a deep sleep state without the need for an ultra low power radio. Consequently, the designer can trade the energy efficiency of this sleep state for the latency associated with waking up the node. Yu et al. [34] utilized a data aggregation tree, that is, a multiple-source single-sink communication paradigm, to represent packet flows under a real-time scenario, where the data gathering must be performed within a specified latency constraint and the overall energy dissipation need to be minimized.

The aforementioned existing energy-delay-driven protocols, however, mainly exploited wireless communication techniques such as modulation scaling [34] and the hybrid radio wakeup scheme [25] to achieve energy-delay efficiency and are only dedicated for WSNs. Nonetheless, there are some other types of networked embedded systems, such as multimedia terminals and 3G cell phones, where energy conservation and low delay need to be simultaneously realized [13], [22]. Therefore, a more general energy-delay efficiency scheme that can be applied to a wide range of networked embedded systems is wanted. Furthermore, none of the existing protocols considered collaborative applications where tasks have precedence constraints, whereas collaborative applications with energy and low-delay requirements are emerging in various networked embedded systems [15], [28], [35]. Aside from that, the existing energy-delay driven schemes normally assumed homogeneous embedded systems and are therefore not suitable for heterogeneous embedded systems. As a consequence, the need for a new energy-delay efficiency strategy that bridges the gap between the existing protocols and the open problems is greatly felt. In this paper, we propose a heuristic task allocation strategy that reduces energy consumption while

generating short schedule lengths for collaborative applications running on heterogeneous networked embedded systems. We assume the following:

1. Different processing nodes have distinct energy consumption rates that are fixed.
2. Communication channels differ in terms of their energy assumption rate.
3. All tasks in a collaborative application are nonpreemptable.
4. Processing nodes are in either an "active" state or an "idle" state without employing the dynamic voltage scaling (DVS) technique.

Note that the fourth assumption does not limit our approach to embedded systems using variable voltage processors since our allocation scheme can be easily integrated with the DVS technique to provide further energy savings.

3 System Models

In this section, we describe mathematical models which were built to represent a task allocation framework, collaborative applications with precedence constraints, an energy consumption model, and a heterogeneity model.

3.1 Networked Embedded Systems

A networked embedded system, in its most general form, consists of a set, for example, $P = \{p_1, p_2, \ldots, p_m\}$, of heterogeneous embedded computational nodes (hereafter referred to as nodes or embedded nodes) connected by a single-hop wired or wireless network. The networked embedded system can be represented as a graph of nodes and their point-to-point links. In the system, an embedded node is modeled as a vertex. There exists a weighted edge between two vertices if they can communicate with each other. Each node in the system has an energy consumption rate measured by Joules per unit of time. With respect to energy conservation, each network link is characterized by its energy consumption rate, which heavily relies on the link's transmission rate, which is modeled by the weight b_{uv} of the edge between node p_u and p_v. An allocation matrix X is an $n \times m$ binary matrix used to reflect a mapping of n tasks to m embedded nodes. Element x_{iu} in X is "1" if task t_i is assigned to node p_u; otherwise, this is "0." The heterogeneity investigated in this study embraces multiple meanings. First, the execution times of a task on different embedded nodes may vary as the nodes might have different processing capabilities. Second, a node that offers task t_i a shorter execution time does not necessarily provide another task t_j with a shortened execution time because different nodes may have distinct processor architectures. This implies that different nodes in a system are suitable for different kinds of tasks. Third, the transmission rates of links may be distinct. Last, the energy consumption rates of the nodes may not necessarily be identical. For the sake of simplicity, we assume that all nodes are fully connected with a dedicated communication system. Each node communicates with other nodes through message passing and the communication time between two tasks assigned to the same node is negligible.

3.2 Task Model

Applications with dependent tasks can be modeled by DAGs [30]. Throughout this paper, a collaborative application is specified as a DAG $G = (T, E)$, where $T = \{t_1, t_2, \ldots, t_n\}$ represents a set of nonpreemptable tasks and E is a set of weighted and directed edges representing communications among tasks (for example, $(t_i, t_j) \in E$ is a message transmitted from task t_i to t_j). The precedence constraints of the collaborative application are represented by all edges in E. Note that our task model considers only one message sent at the end of a task to its subsequent task(s). The communication time spent in delivering a message $(t_i, t_j) \in E$ from task t_i on node p_u to t_j on p_v is determined by s_{ij}/b_{uv}. Here, s_{ij} is the data size of the message and b_{uv} is the transmission rate of a link that connects p_u and p_v. The execution time of task t_i is modeled by a vector, that is, $c_i = \left(c_i^1, c_i^2, \cdots, c_i^u, \cdots, c_i^m\right)$, where c_i^u represents the execution time of t_i on the uth embedded node.

Example 1. Fig. 1 illustrates an example task graph and an example networked embedded system. The task graph has 11 tasks and the processor graph has three processors. The transmission rate and energy consumption rate of the channel between processors p_1 and p_2 are 2 and 0.8, respectively. The energy consumption rate of processor p_1 is 12.6. The matrix of execution times for each task on the three processors is illustrated in the matrix in Fig. 1. For example, task t_1 has an execution time of 3.1, 4.3, and 1.9 s on processors p_1, p_2, and p_3, respectively.

3.3 Energy Consumption Model

Let e_i^u be an energy dissipation caused by task t_i running on node p_u. We denote the energy consumption rate of the uth node when it is active by ECN_u^{active} and the energy dissipation e_i^u can be written as follows:

$$e_i^u = ECN_u^{active} \cdot c_i^u. \tag{1}$$

The energy consumption rate of a networked embedded system is represented by a vector $ECN^{active} = \left(ECN_1^{active}, ECN_2^{active}, \cdots, ECN_m^{active}\right)$. Given a collaborative application with task set T and an allocation matrix X, the total energy consumed by all tasks of the application is

$$\begin{aligned} en^{active}\left(T, X, ECN^{active}\right) &= \sum_{i=1}^{n}\sum_{u=1}^{m} x_{iu} \cdot e_i^u \\ &= \sum_{i=1}^{n}\sum_{u=1}^{m} x_{iu} \cdot ECN_u^{active} \cdot c_i^u \\ &= \sum_{u=1}^{m} ECN_u^{active} \cdot \sum_{i=1}^{n} x_{iu} \cdot c_i^u. \end{aligned} \tag{2}$$

We assume in (2) that no energy consumption is incurred when nodes are sitting idle. However, this assumption is not valid for real-world embedded systems. Before removing this assumption, we introduce a vector of energy consumption rates for the nodes when their energy states are idle, that is, $ECN^{idle} = (ECN_1^{idle}, ECN_2^{idle}, \cdots, ECN_m^{idle})$, where ECN_u^{idle} is an energy consumption rate of node p_u when it is inactive. Additionally, we define f_i as the

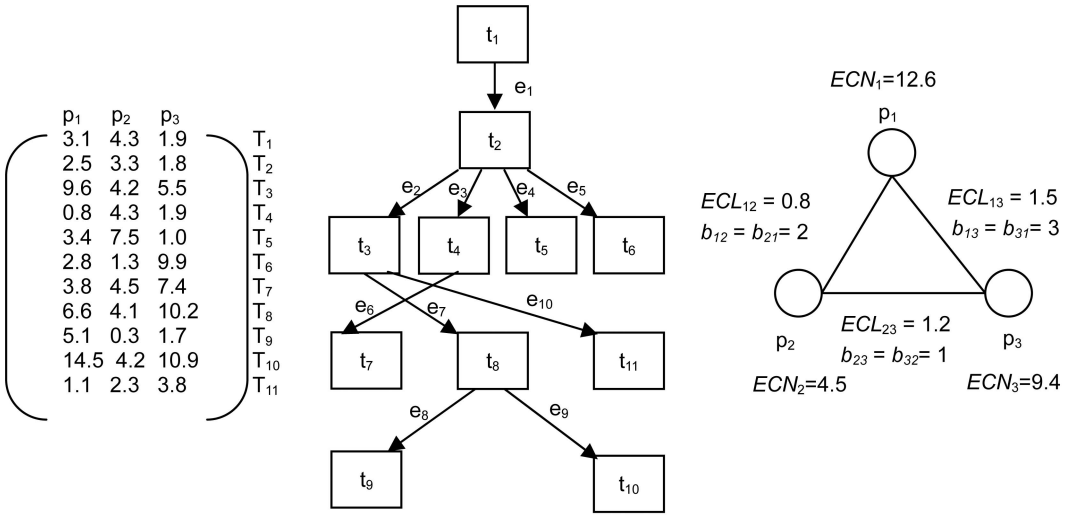


Fig. 1. Example task and networked embedded system. ECN_u is the energy consumption rate of node p_u and ECL_{uv} is the energy consumption of a link between nodes p_u and p_v.

complete time of task t_i. Then, we obtain the analytical formula for the energy consumed by the embedded nodes when they are idle:

$$en^{idle}\big(T, X, ECN^{idle}\big) = \sum_{u=1}^{m} ECN_u^{idle} \cdot \left(\max_{i=1}^{n}(f_i) - \sum_{u=1}^{n} x_{iu} \cdot c_i^u \right), \tag{3}$$

where $\max_{i=1}^{n}(f_i)$ is the schedule length and $\max_{i=1}^{n}(f_i) - \sum_{i=1}^{n} x_{iu} \cdot c_i^u$ is the total idle time on node p_u. Schedule length is the length of a schedule generated by a task allocation scheme for a collaborative application represented by a DAG. It is defined as the latest task completion time among all tasks in a collaborative application. Equation (3) is valid because the energy consumed by an idle node is a product of the corresponding consumption rate and the idle period.

Thus, the total energy consumption of the embedded nodes is derived from (2) and (3) as

$$\begin{aligned} &en\big(T, X, ECN^{active}, ECN^{idle}\big) \\ &= en^{active}(T, X, ECN^{active}) + en^{idle}(T, X, ECN^{idle}). \end{aligned} \tag{4}$$

Similarly, let $\hat{e}_{ij}$ denote the energy consumption of a message $(t_i, t_j) \in E$. Supposing t_i and t_j are, respectively, allocated to nodes p_u and p_v, we can express the energy consumption $\hat{e}_{ij}$ as

$$\hat{e}_{ij} = ECL_{uv}^{active}(b_{uv}) \cdot \frac{s_{ij}}{b_{uv}}, \tag{5}$$

where $ECL_{uv}(b_{uv})$ is the energy consumption rate of the link between nodes p_u and p_v and s_{ij}/b_{uv} is the data transmission time. Note that the energy consumption rate of a network link depends on the transmission rate of the link. In our model, we used the same energy-latency trade-off function presented in [2]. Thus, the total energy consumed by t_i and the messages sent from its immediate predecessors when t_i is allocated on node p_u can be calculated as follows:

$$te_i^u = e_i^u + \sum_{t_j \in IP(t_i)} \hat{e}_{ij}, \tag{6}$$

where $IP(t_i)$ is the set of t_i's immediate predecessors. The definition of $IP(t_i)$ can be found in Definition 2 in Section 3.4.

The energy consumption rate of the network links can be modeled by an $m \times m$ matrix $ECL^{active} = (ECL_{uv}^{active})$, where ECL_{uv}^{active} is the energy consumption rate function of the link between p_u and p_v. The energy consumption in a link between p_u and p_v, denoted by el_{uv}^{active}, is calculated as a cumulative energy consumption of all messages transmitted on the link. The link's energy consumption el_{uv}^{active} can be derived from (5). Then, we have

$$\begin{aligned} el_{uv}^{active}\big(T, X, ECL^{active}\big) &= \sum_{(t_i,t_j)\in L_{uv}} \hat{e}_{ij} \\ &= \sum_{(t_i,t_j)\in L_{uv}} ECL_{uv}^{active}(b_{uv}) \cdot \frac{s_{ij}}{b_{uv}} \\ &= \sum_{i=1}^{n} \sum_{\substack{j=1, j\neq i \\ (t_i,t_j)\in L_{uv}}}^{n} x_{iu} \cdot x_{jv} ECL_{uv}^{active}(b_{uv}) \cdot \frac{s_{ij}}{b_{uv}}, \end{aligned} \tag{7}$$

where L_{uv} is a set of messages transmitted over the link between p_u and p_v and L_{uv} can be defined as $L_{uv} = \{\forall (t_i, t_j) \in E, 1 \le u, v \le m | s_{ij} > 0 \wedge x_{iu} = 1 \wedge x_{jv} = 1\}$.

It is assumed in (7) that all of the messages are transmitted over the link at the same transmission rate, which may not be true for realistic traffic patterns. Hence, we relax the assumption by allowing different messages to be transmitted at various rates, depending on an underlying energy-aware message scheduling mechanism, which we recently developed [2]. Let b_{uv}^{ij} denote the transmission rate at which the message (t_i, t_j) is delivered along the link between p_u and p_v. Then, el_{uv}^{active} is modified as

$$el_{uv}^{active}(T, X, ECL) = \sum_{i=1}^{n} \sum_{\substack{j=1, j\neq i \\ (t_i,t_j)\in L_{uv}}}^{n} x_{iu} \cdot x_{jv} \cdot ECL_{uv}^{active}\big(b_{uv}^{ij}\big) \cdot \frac{s_{ij}}{b_{uv}^{ij}}. \tag{8}$$

The energy consumption of links, that is, $el^{active}(T, X, ECL^{active})$, in the networked embedded system is derived from (8). To be specific, $el^{active}(T, X, ECL^{active})$ is equivalent to the summation of all the link's energy consumption. Thus, $el^{active}(T, X, ECL^{active})$ can be expressed as

$$\begin{aligned} el^{active}\big(T, X, ECL^{active}\big) &= \sum_{u=1}^{m}\sum_{v=1,v\neq u}^{m} el_{uv}^{active}\big(T, X, ECL_{uv}^{active}\big) \\ &= \sum_{i=1}^{n}\sum_{\substack{j=1,j\neq i \\ (t_i,t_j)\in L_{uv}}}^{n}\sum_{u=1}^{m}\sum_{v=1,v\neq u}^{m} x_{iu}\cdot x_{jv}\cdot ECL_{uv}^{active}\big(b_{uv}^{ij}\big)\cdot\frac{s_{ij}}{b_{uv}^{ij}}. \end{aligned} \tag{9}$$

Again, we assume in (9) that no energy consumption is incurred when a link has no message to transmit. We relax this assumption by considering energy consumption when a link is idle during an application's execution. The energy consumption rate of a link sitting idle is denoted by ECL_j^{idle} and we obtain the energy consumed by the link when it is inactive as follows:

$$\begin{aligned} &el_{uv}^{idle}\big(T, X, ECL^{idle}\big) \\ &= ECL_{uv}^{idle}\cdot\left(\max_i^n(f_i) - \sum_{i=1}^{n}\sum_{j=1,j\neq i,(t_i,t_j)\in L_{uv}}^{n} x_{iu}\cdot x_{jv}\cdot\frac{s_{ij}}{b_{uv}^{ij}}\right), \end{aligned} \tag{10}$$

where

$$\max_i^n(f_i) - \sum_{i=1}^{n}\sum_{\substack{j=1,j\neq i \\ (t_i,t_j)\in L_{uv}}}^{n} x_{iu}\cdot x_{jv}\cdot\frac{s_{ij}}{b_{uv}^{ij}}$$

is the total idle time over the link and el_{uv}^{idle} is computed as a product of the consumption rate and the idle period of the link.

The energy consumption of all the links during their idle periods is expressed as

$$\begin{aligned} &el^{idle}(T, X, ECL) \\ &= \sum_{u=1}^{m}\sum_{v=1,v\neq u}^{m} ECL_{uv}^{idle}\left(\max_i^n(f_i) - \sum_{i=1}^{n}\sum_{\substack{j=1,j\neq i \\ (t_i,t_j)\in L_{uv}}}^{n} x_{iu}\cdot x_{jv}\cdot\frac{s_{ij}}{b_{uv}^{ij}}\right). \end{aligned} \tag{11}$$

The total energy consumption of the network links is derived from (8) and (10) as follows:

$$\begin{aligned} &el\big(T, X, ECL^{active}, ECL^{idle}\big) \\ &= el^{active}(T, X, ECL^{active}) + el^{idle}(T, X, ECL^{idle}). \end{aligned} \tag{12}$$

Based on (4) and (11), we can calculate the energy dissipation experienced by a collaborative application with task set T and allocation matrix X. Given the energy consumption rate vectors ECN^{active}, ECN^{idle}, ECL^{active}, and ECL^{idle}, the energy consumption of the networked embedded system can be expressed as

$$\begin{aligned} &e(T, X, ECN^{active}, ECN^{idle}, ECL^{active}, ECL^{idle}) \\ &= en(T, X, ECN^{active}, ECN^{idle} + el(T, X, ECL^{active}, ECL^{idle}). \end{aligned} \tag{13}$$

3.4 Heterogeneity Model

In a heterogeneous embedded system, the computational times of a particular task on different nodes are distinctive, which is referred to as computational heterogeneity. In addition, the energy consumption incurred by the task on different nodes is diverse as different nodes provide distinctive energy consumption rates. We name this energy consumption rate heterogeneity. We believe that both the computational and energy consumption rate heterogeneities are essential characteristics of heterogeneous embedded systems. Therefore, task allocation algorithms for heterogeneous embedded systems need to take the two heterogeneities into account when making allocation decisions. The impacts of these two heterogeneities on system performance and energy conservation will be investigated in Section 5.5.

The computational weight of task t_i on node p_u (for example, cw_i^u) is defined as the ratio between its execution time on p_u and that on the fastest node for t_i in the system. Hence, we have $cw_i^u = c_i^u / \min_{k=1}^{m}\{c_i^k\}$. The computational heterogeneity of task t_i, referred to as HT_i^C, can be quantitatively measured by the standard deviation of the computational weights. Formally, HT_i^C is expressed as

$$HT_i^C = \sqrt{\frac{1}{m}\sum_{u=1}^{m}\big(cw_i^{avg} - cw_i^u\big)^2}, \text{where } cw_i^{avg} = \left(\sum_{u=1}^{m} cw_i^u\right)/m. \tag{14}$$

The computational heterogeneity of a task set T can be computed as follows:

$$HT^C = \frac{1}{|T|}\sum_{t_i\in T} HT_i^C. \tag{15}$$

Similarly, the heterogeneity of energy consumption rates of nodes is expressed as follows:

$$HT_{ECN} = \sqrt{\frac{1}{m}\sum_{u=1}^{m}\Big(ECN_{avg}^{active} - ECN_u^{active}\Big)^2}, \tag{16}$$

where $ECN_{avg}^{active} = \frac{1}{m}\sum_{u=1}^{m} ECN_u^{active}$.

Equations (14) and (16) will be used to calculate the degrees of computational heterogeneity and energy consumption rate heterogeneity in Section 5.1 (see Table 1).

4 The BEATA Algorithm

4.1 Energy-Adaptive Window

The BEATA algorithm strives to minimize $e(T, X, ECN^{active}, ECN^{idle}, ECL^{active}, ECL^{idle})$ expressed by (13), which indicates that there are two means of conserving energy for networked embedded systems. The first item on the right-hand side of (13) is the total energy consumed by the tasks of a collaborative application and, therefore, it is imperative for the BEATA algorithm to allocate each task in the application to a node that can lead to the least energy consumption. The second item on the right-hand side of (13)

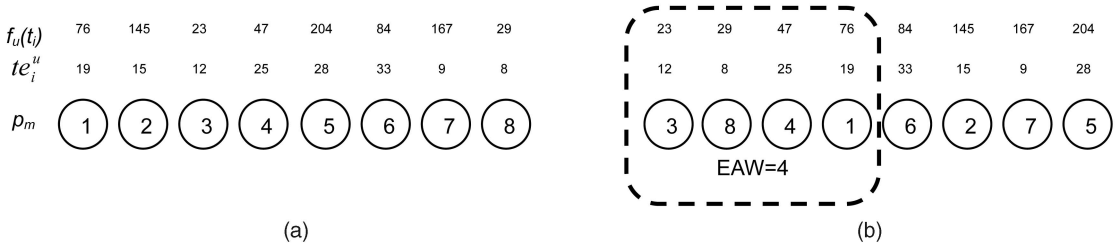


Fig. 2. Example of an energy-adaptive window.

is the total energy consumption caused by message transmissions. It is suggested that BEATA needs to judiciously consider message energy dissipations while allocating tasks on the embedded nodes. Importantly, if we merely address the issue of energy conservation without considering schedule lengths, there is a strong likelihood that the allocation algorithm will yield a long and unacceptable schedule length for the collaborative application.

To make the best trade-off between energy savings and schedule lengths, we employ an EAW, within which a node is chosen for each task in such a way as to offer lower energy consumption and earlier finish time of the task. The example shown in Fig. 2 delineates the process of using the EAW (that is, $\mathrm{EAW} = 4$) to achieve a good trade-off between energy conservation and make-span time. Supposing there is a networked embedded system $P = \{p_1, p_2, \ldots, p_8\}$, Fig. 2a shows task t_i's finish time $f_u(t_i)$ and the total energy consumption te_i^u (including the message energy consumption, if applicable) provided by each node in the system. Step 6 (see Fig. 3) of the BEATA algorithm sorts all of the nodes in the finish time of t_i in nondecreasing order (Fig. 2b). Since the size of the EAW is set to 4 $(\mathrm{EAW} = 4)$, BEATA only chooses the node whose energy savings is the most significant among the first four candidate nodes (Fig. 2b). Therefore, node p_8 will be selected for task t_i in this case. It is easy to understand that enlarging the EAW could result in more energy conservation, but a worse performance in delay as the make-span time will accordingly increase. As such, EAW is essentially an energy-delay handle from which a user can tune the trade-off between energy savings and low delay. In other words, by providing the handle, BEATA turns out to be a tunable energy-delay efficiency scheme that can be applied in various environments where users have their special requirements in energy savings and responsiveness.

4.2 Algorithm Description

The BEATA algorithm (see Fig. 3) is conducive to increasing the heterogeneous embedded nodes' lifetimes while maintaining high performance in terms of make-span time for collaborative applications running on networked embedded systems. In other words, BEATA can increase the embedded nodes' lifetimes by dramatically reducing energy dissipation (see Step 13). Before minimizing the energy consumption of task t_i, BEATA organizes all the nodes in nondecreasing order in terms of t_i's finish time (see (20)). Step 8 determines the energy consumption incurred by the task on an embedded node, while Steps 9 and 10 calculate the energy consumed by all the messages received by the task from its predecessors. Among all the candidate nodes listed in the EAW, Step 13 chooses the most appropriate node that yields the minimal energy dissipation for the task and its corresponding messages, thereby conserving energy without excessive performance deterioration. Then, Step 14 allocates the task to the best candidate node. After the allocation of the task is accomplished, Step 15 updates the schedule of the node to which the task is allocated.

Two important parameters, the earliest start time and finish time on a node, are used in the above algorithm. We denote the earliest start time and finish time of task t_i on node p_u by $est_u(t_i)$ and $f_u(t_i)$, respectively. In what follows, we present derivations that lead to the final expressions for these two parameters. The t_i can start execution on a node p_u if and only if data arrives from all of its immediate parents in order to satisfy precedence constraints. Suppose task t_i has only one immediate predecessor task t_j. The earliest available time of task t_i on node p_u, denoted as $eat_u(t_j, t_i)$, is defined as the data arrival time of t_i on p_u. It relies on 1) the finish time f_j of t_j, 2) the message start time $mst(t_j, t_i)$, and 3) the transmission time s_{ji}/b_{vu} for the message sent from t_j to t_i, where p_v is the processor to which task t_j has been allocated. It is assumed that, if both tasks are allocated to the same embedded node, the transmission time is negligible. Thus, $eat_u(t_j, t_i)$ is expressed as

$$eat_u(t_j, t_i) = \begin{cases} f_j, & \text{if } p_u = p_v, \\ mst(t_j, t_i) + s_{ji}/b_{vu}, & \text{otherwise.} \end{cases} \quad (17)$$

The earliest available time of t_i, which is denoted by $eat_u(t_i)$, is the maximum of $eat_u(t_j, t_i)$ among all of its immediate predecessors. Considering all immediate predecessors of t_i, we can obtain $eat_u(t_i)$ as

$$eat_u(t_i) = \max_{(t_j, t_i) \in E} \{eat_u(t_j, t_i)\}. \quad (18)$$

Now, t_i can start execution on p_u after its earliest available time $eat_u(t_i)$, either at the ready time of the node p_u^R (the finish time of the last task scheduled on p_u) or at a time earlier than p_u^R if a suitable scheduling hole (the idle time slot existing in a node's schedule because of the nonavailability of data earlier) is available on p_u. The definition of a suitable scheduling hole is given as follows:

Definition 1. *Given a set of r tasks, $\{t_1, t_2, \ldots, t_r\}$, scheduled on node p_u, the idle time slot (scheduling hole) H_k (between t_{k+1} and t_k, with H_k^S and H_k^F being its start time and finish time, respectively) is suitable for task t_i if*

$$H_k^F - \max\{eat_u(t_i), H_k^S\} \geq c_i^u,$$

Input: An embedded system *P*, a collaborative application *DAG G* = (*T*, *E*), and *EAW*
Output: An allocation matrix *X* and schedules of the tasks in *T*
0. Create a linear ordering queue *Q* of *G* using topological sort such that if (t_i, t_j) $\in$ *E* then t_i appears before t_j in *Q*.
1. **for** each task $t_i \in Q$ **do**
2. **for** each node $p_u \in P$ in the system **do**
3. compute $est_u(t_i)$, the earliest start time of t_i on embedded node p_u (see **Eq. 19**)
4. compute $f_u(t_i)$, the finish time of t_i on node p_u (see **Eq. 20**)
5. **end for**
6. Sort all nodes in finish time of t_i in a non-decreasing order
7. **for** each embedded node in the energy-adaptive window **do**
8. Use **Eq. 1** to compute the energy consumption consumed by task t_i
9. **for** each t_j, where (t_j, t_i) $\in E$ **do**
10. Use **Eq. 5** to compute the energy consumption cause by message (t_j, t_i)
11. Use **Eq. 6** to compute the total energy consumed by t_i and the messages received by t_i
12. **end for**
13. Select p_v in the energy-adaptive window that offers the smallest total energy consumption
14. Assign t_i to p_v
15. Update the schedule on node p_v
16. Compute the energy consumed by t_i on p_v and messages received by t_i
17. Record start time and finish time for task t_i
18. **end for**

Fig. 3. The BEATA task allocation algorithm.

where $H_0^S = 0$, $H_0^F = S_{1,u}$, $H_r^S = F_{r,u}$, *and* $H_r^F = \infty$ *for* $0 \le k \le r$, *and* $S_{d,u}$ *and* $F_{d,u}$ *are the start time and finish time of task* t_d $(1 \le d \le r)$ *on* p_u.

Thus, the earliest start time of t_i on p_u is given by (19):

$$est_u(t_i) = \max\{eat_u(t_i), \min\{p_u^R, H_k^S\}\}. \quad (19)$$

With the value of $est_u(t_i)$ in place, we can obtain the finish time of t_i on p_u by using (20). The finish time is equal to the summation of the earliest start time $est_u(t_i)$ and t_i's execution time on p_u:

$$f_u(t_i) = est_u(t_i) + c_i^u. \quad (20)$$

Hence, the earliest finish time of t_i in the system is given by (21):

$$eft\ (t_i) = \min_{p_u \in P}\{f_u(t_i)\}. \quad (21)$$

4.3 Optimality of the BEATA Algorithm

Before proceeding with the qualitative comparisons among our algorithm, an existing algorithm, and a baseline scheme, we demonstrate the time complexity of the BEATA algorithm and prove some optimality guarantees of BEATA in schedule length and energy consumption.

Theorem 1. *Given a networked embedded system* $P = \{p_1, p_2, \ldots, p_m\}$ *and a collaborative application represented by a task graph* $G = (T, E)$, *the time complexity of BEATA is* $O(nmlgm + nkq) + O(n + |E|)$, *where* n *is the number of tasks,* m *is the number of embedded nodes in the system,* k *is the EAW size,* q *is the maximum indegree of* G, *and* $|E|$ *is the number of edges in* G.

Proof. The time complexity of a topological sort of the graph G is $O(|T| + |E|)$ (see Step 0), where $|T|$ is the number of vertices in G , and $|E|$ is the number of edges in G. It takes $O(m)$ time to compute the earliest start times and earliest finish times for a task on all of the nodes (see Steps 3 and 4). The time complexity of sorting the earliest finish times is $O(mlgm)$ since we only have m nodes (see Step 6). To determine the most appropriate node that offers the minimal energy consumption of a task, the time complexity is $O(kq)$ (see Steps 7-12). Other steps simply take $O(1)$ time. Hence, the time complexity of the BEATA algorithm is $O(n + |E|) + O(n)(O(m) + O(mlgm) + O(kq)) = O(nmlgm + nkq) + O(n + |E|)$. □

Theorem 1 indicates that the time complexity of the BEATA algorithm is typically low. For example, in our experiments, the values of n and m are set to about 300 and 64, the values of k and q are around 4 and 2, and the value of $|E|$ is in the range [54, 543], which should take less than hundreds of microseconds to complete the BEATA algorithm in modern processors. An implication of Theorem 1 is that BEATA has the potential of being extended to deal with dynamic scheduling because of its low complexity.

Lemmas 1 and 2 are helpful in the proofs of Theorems 2 and 3, which signify that BEATA has good performance in terms of the finish times of collaborative tasks in networked embedded systems.

Lemma 1. *Given a task* t_i, *a networked embedded system* P, *and a topological sort* Q *of a collaborative application* G, *it is impossible for BEATA to further reduce* $est_u(t_i)$, *which is the earliest start time of* t_i *on any node* p_u *in* P.

Proof. Recall that, given task t_i and node p_u in the system, the earliest start time of t_i on p_u is calculated by $est_u(t_i) = \max\{eat_u(t_i), \min\{p_u^R, H_k^S\}\}$ (see (19)). If $eat_u(t_i) \ge \min\{p_u^R, H_k^S\}$, the proof of Lemma 1 is immediate from (19). In cases where $eat_u(t_i) < \min\{p_u^R, H_k^S\}$, the BEATA algorithm scans the idle time slots on p_u, followed by choosing the first suitable scheduling hole (see Definition 1) to accommodate t_i. If no suitable scheduling hole is found, BEATA schedules t_i to start at the ready time p_u^R of node p_u. Thus, it is proven that the task is unable to start its execution earlier than $est_u(t_i)$, given the above two scenarios. Hence, we proved that the earliest start time of t_i on any node p_u in P can no longer be improved. □

Lemma 2. *Given a task t_i, a networked embedded system P, and a topological sort Q of a collaborative application G, it is impossible for BEATA to further reduce $f_u(t_i)$, which is the finish time of t_i on any node p_u in P.*

Proof. The proof of Lemma 2 is immediate from (20) and Lemma 1. Recall that the finish time of t_i is equal to the summation of the earliest start time $est_u(t_i)$ and t_i's execution time c_i^u on p_u (see (20)). The summation of $est_u(t_i)$ and c_i^u is minimized by BEATA because $est_u(t_i)$ cannot be further reduced (see Lemma 1) and c_i^u is a constant. This completes the proof of Lemma 2. □

In the following, we show that BEATA is capable of minimizing schedule lengths under the condition that the EAW is equal to 1.

Theorem 2. *Given a networked embedded system $P = \{p_1, p_2, \ldots, p_m\}$, a collaborative application $G = (T, E)$, and a topological sort Q of G, when the size of EAW is set to 1 (that is, $\mathrm{EAW} = 1$), for each task in T, the BEATA algorithm assigns it on a node that offers the task the minimum possible finish time.*

Proof. Without loss of generality, let us consider a task t_i in a scenario where the value of EAW is set to 1. Now, we need to prove the theorem by demonstrating that the finish time of t_i on p_v given by BEATA cannot be further improved. First, Lemma 1 proves that the earliest start time of t_i on any node p_u in the system is minimized. Next, Lemma 2 proves that the finish time of t_i on any node p_u in the system cannot be further reduced. Last, we need to show that the earliest finish time of t_i in the entire system cannot be further improved. The BEATA algorithm sorts all of the nodes in nondecreasing order of the finish times of t_i (see Step 6). As a consequence, the first node p_v in the list is the one that provides the earliest finish time for t_i in the entire system. It is proven that, if the value of EAW is 1, only p_v (see Step 7) is allowed to enter into the loop from Steps 7 to 12. This means that task t_i is allocated to the node that offers the earliest finish time for the task in the system. In addition, the earliest finish time can be computed by (21), which guarantees that the earliest finish time of t_i is the minimum among all m possible finish times of t_i in the system, that is, the earliest finish time of t_i in the entire system cannot be further reduced. Hence, the proof holds. □

Note that Theorem 2 only proves that, for each task in T, the BEATA algorithm assigns it on a node that offers the task the minimum possible finish time under a given P, G, and Q. It has no implication that the make span of the final schedule of the entire DAG G generated by BEATA is minimized. In other words, the minimum finish time for each task in T cannot guarantee a schedule with minimal length because the optimal scheduling of tasks of a DAG onto a set of processors is a strong NP-hard problem [24]. To prove the optimality of BEATA with respect to energy savings, we first prove the following lemmas. Lemma 3 and Lemma 4 prove that BEATA optimizes the energy consumption of each task and its corresponding messages. Theorem 3 shows a way of computing the energy dissipation of a collaborative application. In our experiments (see Section 5), the energy consumed by idle resources is negligible and, hereafter, we ignore the energy dissipation of resources when they are sitting idle.

Definition 2. *Let $IP(t_i)$ be the set of t_i's immediate predecessors and, thus, we have $IP(t_i) = \{t_k \in T | (t_k, t_i) \in E\}$. Let $IM(t_i)$ be the set of messages directly transmitted from t_i's immediate predecessors. An element t_k in $IP(t_i)$ implies the existence of a message transmitted from t_i's immediate predecessor t_k.*

Lemma 3. *The energy consumption of task t_i and the messages transmitted from t_i's immediate predecessors is equal to*

$$\sum_{u=1}^{m}\left(x_{ij}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right).$$

Proof. If task t_i is allocated to p_u, the energy dissipation of t_i can be expressed by $ECN_u^{active}\cdot c_i^u$ (see (1)). Given an allocation matrix X and an embedded system P, we compute the energy consumption of t_i in the system as

$$\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right). \tag{22}$$

We derive from (5) that the energy consumption of the message $(t_k, t_i) \in E$ is equal to

$$\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right).$$

Therefore, the energy dissipation of all the messages transmitted from t_i's immediate predecessors is

$$\sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right), \tag{23}$$

which is the summation of the energy consumed by messages in the set $IM(t_i)$. As a consequence, the energy consumption of t_i and the messages transmitted from t_i's immediate predecessors is derived from (22) and (23) as

$$\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right), \tag{24}$$

which completes the proof of Lemma 3. □

Lemma 4. *We consider a task (for example, t_i) in a networked embedded system P, where the number of nodes is m and m is the maximal possible value for the EAW. If the value of EAW is set to m, the BEATA algorithm optimizes the energy consumption of task t_i and the messages transmitted from t_i's immediate predecessors. More formally, if the value of EAW is set to m, then BEATA is able to generate an allocation matrix X that minimizes the value of the following:*

$$\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right).$$

Proof. To prove the correctness of Lemma 4, we have to show that, if the value of EAW is set to m, the BEATA algorithm minimizes the energy consumption of t_i and the messages transmitted from t_i's immediate predecessors. First, BEATA executes the loop between Steps 7 and 12 for all of the m nodes in the system (see Steps 7-12) since the value of EAW is m. For each node p_v in P, the energy consumed by t_i and the messages transmitted from t_i's immediate predecessors can be calculated by applying (1) and (5). Thus, the energy dissipation caused by t_i and its corresponding messages can be written as

$$ECN_v^{active}\cdot c_i^v + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\left(x_{ku}\cdot ECL_{uj}^{active}(b_{uj}^{ki})\cdot\frac{s_{ki}}{b_{uj}^{ki}}\right)\right). \quad (25)$$

Second, all of the items in (25) are fixed values and are known a priori. As a consequence, the energy consumption calculated by (25) is a constant from the perspective of each node in the system. Last, from the candidate node in the EAW, Step 13 chooses a node that offers the smallest energy consumption for t_i and the messages sent from t_i's immediate predecessors. Step 14 assigns t_i to the node selected by Step 13, which means that the energy consumption of t_i and its corresponding messages is minimized by BEATA. Thus, BEATA minimizes the value of (25). Therefore, it is proven that the BEATA algorithm is capable of generating an allocation matrix X in a judicious way to minimize the value of

$$\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right).$$

This completes the proof of Lemma 4. □

Theorem 3. *The energy consumption induced by a collaborative application with task set T and message set E is*

$$\sum_{i=1}^{n}\left(\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)\right).$$

Proof. The proof is immediate from Lemma 3. The energy consumption of t_i and the messages in set $IM(t_i)$ is

$$\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right).$$

The energy dissipation of the application with task set T and message set E is the cumulative energy of all the tasks in T and all of the messages in E. The energy consumption of all of the tasks in T is $\sum_{i=1}^{n}\sum_{u=1}^{m}x_{iu}\cdot ECN_u^{active}\cdot c_i^u$ (see 2), whereas the energy consumption induced by all the messages in E is derived from (23) as

$$\sum_{i=1}^{n}\left(\sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)\right).$$

Consequently, the energy consumption induced by the collaborative application is equal to

$$\sum_{i=1}^{n}\left(\sum_{u=1}^{m}\left(x_{iu}\cdot ECN_u^{active}\cdot c_i^u\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)\right).$$

□

Now, we are positioned to prove the optimality of BEATA with respect to energy conservation.

Theorem 4. *Given a networked embedded system $P=\{p_1,p_2,\ldots,p_m\}$ and a collaborative application $G=(T,E)$, where m is the number of the node and the maximal possible value of the EAW. If the value of EAW is set to m, then the BEATA algorithm optimizes the energy consumption of G in P.*

Proof. The proof of this theorem is immediate from Lemma 4 and Theorem 3. Lemma 4 proves that, if the value of EAW is set to m, BEATA minimizes the value of

$$\sum_{j=1}^{m}\left(x_{ij}\cdot ECN_j^{active}\cdot c_i^j\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)$$

for each task t_i. Theorem 3 shows that the energy consumption induced by a collaborative application with T and E is

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(x_{ij}\cdot ECN_j^{active}\cdot c_i^j\right) + \sum_{t_k\in IP(t_i)}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)\right).$$

Hence, if the value of EAW is set to m, the BEATA algorithm can minimize the value of

TABLE 1
Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
Number of tasks	(300) – (50, 100, 200, 300, 400, 500)
Energy-adaptive window	(4) – (2, 4, 6, 8, 10, 12, 14, 16)
Number of nodes	(64) – (16, 32, 64,128)
Energy consumption rate heterogeneity degree	(1.2) – (1, 1.2, 1.4, 1.6, 1.8, 2.0) (see Eq. 16)
Computational heterogeneity degree	(5) – (2, 3, 4, 5, 6) (see Eq. 14)
Standard node energy consumption rate	200 mW
Communication energy consumption rate	See the energy-transmission time model in [2].

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{m}\left(x_{ij}\cdot ECN_{j}^{active}\cdot c_{i}^{j}\right)+\sum_{t_k\in IP(t_i)_i}\left(\sum_{u=1}^{m}\sum_{v=1}^{m}\left(x_{ku}\cdot x_{iv}\cdot ECL_{uv}^{active}(b_{uv}^{ki})\cdot\frac{s_{ki}}{b_{uv}^{ki}}\right)\right)\right),$$

which is the energy consumption caused by the application. □

4.4 Existing and Baseline Algorithms

In this section, we qualitatively compare BEATA with an existing algorithm and a baseline algorithm, namely, the LIST scheduling scheme and the Greedy Energy-Aware Task Allocation (GEATA) algorithm. The LIST and GEATA algorithms are briefly described as follows:

1. *LIST*. The LIST algorithm is a well-known heuristic for DAG scheduling in heterogeneous systems [10], [21]. For each task allocation, LIST chooses a node that can offer the task the earliest finish time, considering both computation and communication times. The goal of LIST is to generate a schedule for a DAG with the shortest schedule length.
2. *GEATA*. For comparison purposes, we also developed a baseline scheduling algorithm GEATA, which is a variant of the BEATA algorithm. For each task allocation, GEATA selects a node that can yield the least energy consumption induced by the task and its corresponding messages. The GEATA algorithm aims at generating a schedule that provides the least total energy consumption.

Both LIST and GEATA employ the same topological sort as BEATA does at Step 0 in Fig. 3. In light of the description of BEATA (see Fig. 3) and the two theorems (Theorems 2 and 4) proven in Section 4.3, we can see that LIST and GEATA are two variants of BEATA. More precisely, two observations can easily be made. First, when the EAW size is set to 1 (that is, $\mathrm{EAW}=1$), the BEATA algorithm degrades to the LIST algorithm. Second, when the EAW size is set to m (that is, $\mathrm{EAW}=m$), the BEATA algorithm degrades to the GEATA algorithm. The correctness of these two observations is experimentally validated by the empirical results presented in Section 5.4.

5 Performance Evaluation

In this section, we evaluate the effectiveness of the proposed energy-latency-driven task allocation scheme. To demonstrate the strength of BEATA, we quantitatively compare it with LIST and GEATA, the descriptions of which can be found in Section 4.4. The purpose of comparing BEATA with LIST is to evaluate the performance improvements over an existing algorithm without considering energy conservation. The goal of comparing BEATA with GEATA is to experimentally reveal that BEATA can explore trade-offs between energy savings and performance. In addition, the numerical results presented in Section 5.4 validate the correctness of Theorems 5 and 6 (see Section 4.4). Part of the preliminary results in Sections 5.2 and 5.4 were presented in [31].

5.1 Simulation Setup

Before presenting our empirical results, we present the simulation model as follows: Table 1 summarizes the configuration parameters of the simulated heterogeneous embedded systems used in our experiments. The parameters of nodes in the heterogeneous embedded systems are chosen to resemble real-world processors like the Intel StrongARM 1100. The relationship between energy rate and transmission rate is 100 mW at 100 Kbps, which means that the time and energy cost for transmitting 1 bit are around 10 μs and 1 μJoule [2]. All synthetic parallel jobs used from Sections 5.2 to 5.5 were created by TGFF [7], which is a randomized task graph generator. The task graph used in Section 5.6 is based on a digital signal processing system detailed in [29].

Although the number of tasks, number of nodes, values of outdegree, and task execution time are synthetically generated, we examined the impacts of these important workload parameters on system performance by controlling the parameters. The performance metrics by which we evaluate the system performance include the following:

- *Make span*. This is the latest task completion time in the task set represented by a DAG.
- *Energy consumption*. This is the total energy consumed by the task set, including the computation energy consumption and communication energy consumption (see (12)).
- *Utilization standard deviation (USD)*. This is the standard deviation of nodes utilization in the simulated heterogeneous embedded systems.
- *Energy standard deviation (ESD)*. This is the standard deviation of the energy consumption of nodes in the simulated heterogeneous embedded systems.

5.2 Overall Performance Comparisons

The goal of conducting this experiment is 1) to compare the proposed BEATA algorithm against the well-known LIST algorithm and the baseline heuristic GEATA and 2) to understand the sensitivity of the three algorithms to the number of tasks in a collaborative application. We tested six applications represented in the form of task graphs, with the

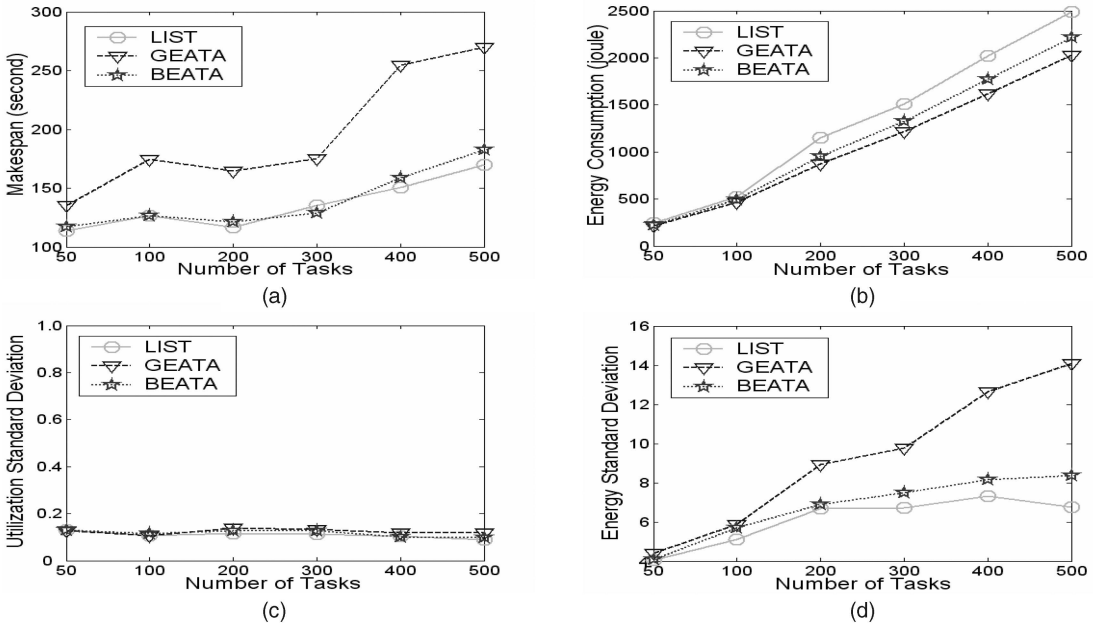


Fig. 4. Performance impact of the number of tasks.

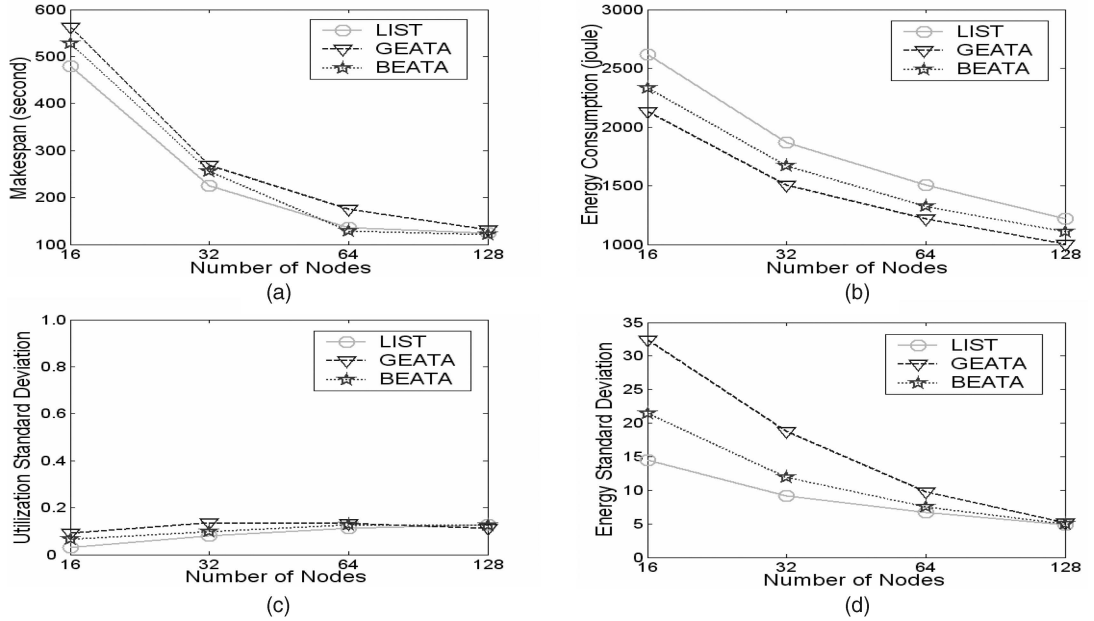
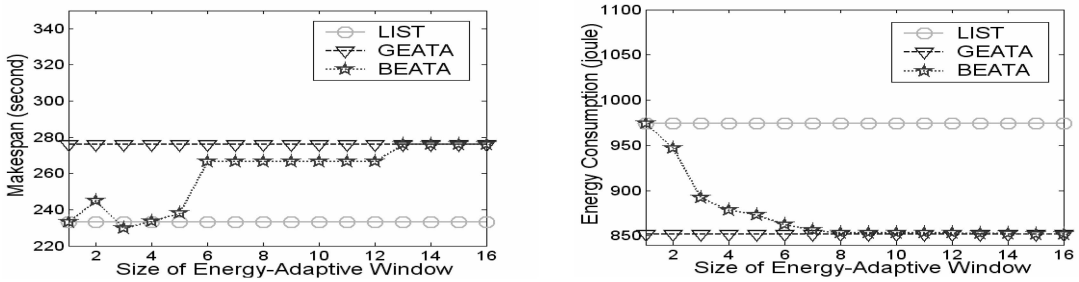


Fig. 5. Performance impact of the number of nodes.

number of tasks varying from 50 to 500 with precedence constraints.

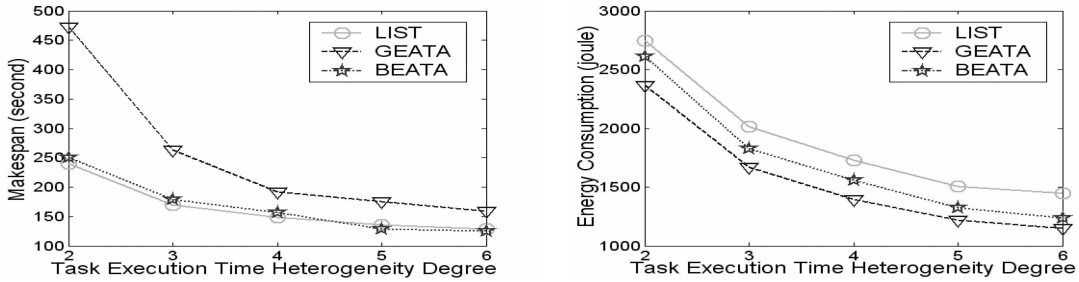
We observe, based on Fig. 4a, that BEATA and LIST exhibit very similar performance in terms of make span, whereas BEATA noticeably outperforms the GEATA algorithm. This is because BEATA considers both energy consumption and make-span time when allocating each task onto a computing node, whereas GEATA only takes energy consumption into account. An interesting observation is that BEATA even generates a shorter schedule than LIST when the number of tasks is 300. The "anomaly" can be explained by the fact that the LIST algorithm cannot guarantee the shortest schedule in a heterogeneous system due to the lack of the information about tasks that are not yet scheduled and the varying execution times for each task on different nodes. Compared with LIST, BEATA, on average, only increases the make span by 2.9 percent but saves energy by 12.1 percent. Fig. 4b reveals that BEATA and GEATA consistently perform better than LIST in terms of energy consumption. In particular, GEATA achieves an improvement, on average, of 19.3 percent.

To understand computation workload distribution and energy dissipation among all the nodes, we measured the USD and ESD in Figs. 4c and 4d, respectively. We observe, based on Fig. 4c, that the computation workload is evenly distributed in the three examined algorithms. In terms of the even distribution of energy dissipation among all the nodes, GEATA is obviously inferior to the other two algorithms (see Fig. 4d).



(a) (b)

Fig. 6. Performance impact of the size of the energy-adaptive window.



(a) (b)

Fig. 7. Performance impact of computational heterogeneity.

5.3 Scalability

This experiment is intended to investigate the scalability of the BEATA algorithm. We scaled the number of nodes in the system from 16 to 128. The task graph used in this experiment is comprised of 300 tasks. Fig. 5 plots the performance of the three algorithms as functions of the number of nodes.

The results show that BEATA exhibits good scalability. To be specific, Fig. 5 shows that the three algorithms deliver increasing performance in both make span and energy consumption when the number of nodes increases. The results are expected because increasing the number of nodes provides more opportunities for tasks and messages to shorten the finish times and reduce the energy consumption. A second important observation is that BEATA outperforms LIST in terms of both make span and energy savings in cases where the number of nodes is larger than 64. The implication of this observation is that BEATA is beneficial for large-scale embedded systems, where performance and energy conservation are two objectives to be addressed. In terms of USD and ESD, all three algorithms achieve good performance when the system size is enlarged (Figs. 5c and 5d).

5.4 Sensitivity to Size of Energy-Adaptive Window

To verify the performance impact of EAW, we evaluate the performance as functions of the size of EAW. Since LIST and GEATA do not have EAWs, their performance in all metrics is kept constant. The results plotted in Fig. 6 validate the relationships among the three algorithms proved in Section 4.4. When the EAW is set to 1, BEATA gracefully degrades to LIST. On the other hand, if the EAW is increased to 16, which is the total number of nodes in the system, then BEATA degrades to GEATA. We observe, based on Fig. 6, that BEATA achieves the best trade-offs between make span and energy consumption when the size of EAW falls in the range between 3 and 5. The EAW size in this range enables BEATA to achieve almost the same performance in make span as that of LIST while conserving energy by up to 10.4 percent. We attribute the performance improvements of BEATA over LIST and GEATA to the fact that BEATA is an energy-adaptive scheme that judiciously assigns tasks to nodes by considering both execution times and energy savings. Note that the choice of the optimal size of EAW is application dependent as different applications have distinct characteristics such as computational heterogeneities and computation-to-communication ratios. Thus, for a particular collaborative application, one can use a binary search algorithm to quickly discover an appropriate value of EAW in the range from 1 to the total number of nodes m.

5.5 Impacts of Heterogeneities

In this experiment, we investigate the impacts of the two heterogeneities (see Section 3.4) on system performance. To be specific, we evaluate the performance of the three algorithms in cases where 1) the energy consumption rate heterogeneity is kept constant while the computational heterogeneity varies (see Fig. 7) and 2) the computational heterogeneity is fixed while the energy consumption rate heterogeneity is increased (see Fig. 8).

Fig. 7a demonstrably shows that BEATA can maintain a similar performance to that of LIST in terms of make span when the heterogeneity degree of energy consumption rate increases. Furthermore, Fig. 7a reveals that the make-span performance of BEATA and LIST is insensitive to the heterogeneity degree of energy. Unlike BEATA and LIST, the make span of GEATA dramatically goes up with the increasing heterogeneity degree of energy. Fig. 7b intuitively shows that BEATA consistently delivers better energy performance than LIST.

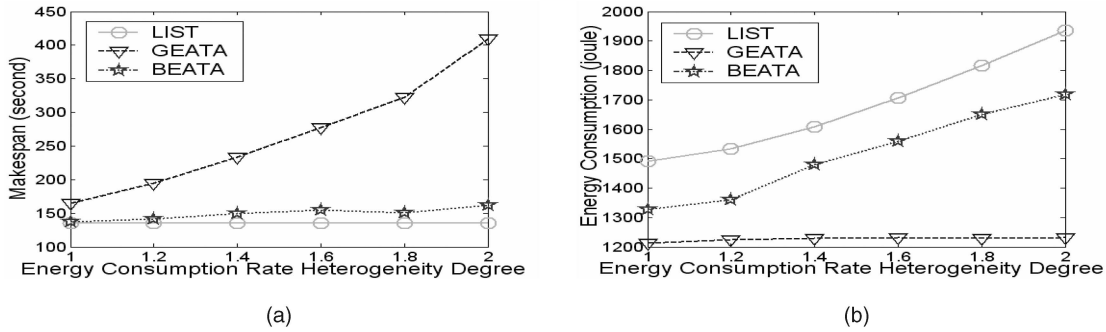


Fig. 8. Performance impact of energy consumption rate heterogeneity.

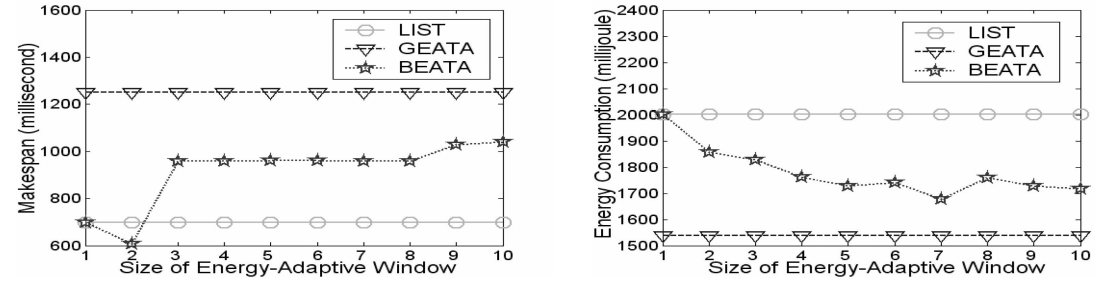


Fig. 9. Performance impact of the size of the energy-adaptive window in DSP.

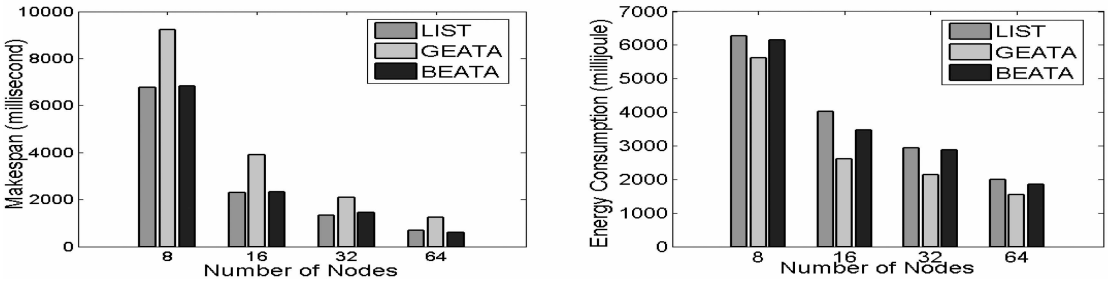


Fig. 10. Performance impact of the number of nodes in DSP.

Fig. 8 shows that BEATA is almost tied with LIST in make span and is obviously superior to LIST in energy consumption. In energy consumption, the performance of BEATA is close to that of GEATA. The implication is that BEATA can show its strength in an embedded system where its computational heterogeneity varies.

5.6 Evaluation in a Real Application

To validate the results from the synthetic collaborative applications, we evaluate in this experiment the BEATA algorithm by using a real system, that is, a digital signal processing system [28], [29].

The DSP system processes sonar data with five independent threads, each driven by its own sensors [29]. The entire system can be represented as a DAG composed of five sub-DAGs, with a total of 119 tasks. The execution time of tasks varies from 0.28 to 1728 ms and the data size of messages changes from 1 to 32 Kbytes. The detailed information pertinent to the DSP application can be found in [28], [29]. We conducted two groups of experiments. The first group of experiments (see Fig. 9) is focused on the size of EAW, while the second group (see Fig. 10) is intended to test the scalability of BEATA in the context of a real-world application. The performance patterns plotted in Fig. 9 are similar to those reported in Section 5.4, thereby verifying that BEATA can gain performance improvements in make span and energy savings for a real application. Fig. 10 shows that, in the case of DSP application, the three algorithms can scale well to large numbers of nodes. The results in Figs. 9 and 10 can be envisioned as a strong validation for the experimental results based on synthetic applications. The important conclusion drawn from these groups of experiments is that the BEATA algorithm can be successfully deployed in heterogeneous embedded systems to save on energy consumption of real-world collaborative applications.

6 Conclusions

In this paper, we address the issue of allocating tasks of collaborative applications in heterogeneous embedded systems, with an objective of energy-delay efficiency. BEATA, which is a tunable heuristic task allocation strategy, is developed to accomplish a variety of energy-delay trade-offs required by different application environments. Compared with traditional energy-delay-driven protocols for WSNs, BEATA has several desired features. First, it is a polynomial-time heuristic task allocation strategy, which is easier to implement without any extra

hardware cost. Second, it is a more general scheme that can be applied in various networked embedded systems. Next, it considered precedence constraints in collaborative applications, which are emerging in a range of networked embedded systems. Last, it targeted heterogeneous networked embedded systems, which are more common and complicated than their homogeneous peers. We conducted extensive experiments by using a real-world application and synthetic benchmarks. The experimental results show that BEATA significantly improves performance in terms of energy dissipation and make-span time over two baseline allocation schemes. Compared with LIST, BEATA achieves improvement in energy savings, on average, of 12.1 percent, with only a 2.9 percent increase in make span. Compared with GEATA, BEATA reduces the make span by an average of 38.9 percent.

Future studies in this research can be performed in the following directions: First, we will extend our scheme to multidimensional computing resources from which energy savings can be achieved. For now, we simply consider CPU time and network communication time. Memory access and I/O activities will be considered in the future. Second, we intend to enable the BEATA scheme to deal with real-time collaborative applications, where the hard deadlines must be guaranteed.

ACKNOWLEDGMENTS

This work was supported in part by US National Science Foundation Computing Processes and Artifacts (CISE-CCF) Grant 0742187. The authors wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] K. Akkaya and M. Younis, "An Energy-Aware QoS Routing Protocol for Wireless Sensor Networks," *Proc. 23rd Int'l Conf. Distributed Computing Systems,* pp. 710-715, 2003.

[2] M. Alghamdi, T. Xie, and X. Qin, "PARM: A Power-Aware Message Scheduling Algorithm for Real-Time Wireless Networks," *Proc. First ACM Workshop Wireless Multimedia Networking and Performance Modeling,* pp. 86-92, 2005.

[3] W. Alsalih, S. Akl, and H. Hassancin, "Energy-Aware Task Scheduling: Towards Enabling Mobile Computing over Manets," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp.,* Apr. 2005.

[4] H.M. Ammari and S.K. Das, "Trade-Off between Energy Savings and Source-to-Sink Delay in Data Dissemination for Wireless Sensor Networks," *Proc. Eighth ACM/IEEE Int'l Symp. Modeling, Analysis and Simulation of Wireless and Mobile Systems,* pp. 126-133, 2005.

[5] S. Bansal, P. Kumar, and K. Singh, "An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, pp. 533-544, 2003.

[6] A. Boukerche, F.H.S. Silva, R.B. Araujo, and R.W.N. Pazzi, "A Low-Latency and Energy-Aware Event Ordering Algorithm for Wireless Actor and Sensor Networks," *Proc. Eighth ACM/IEEE Int'l Symp. Modeling, Analysis and Simulation of Wireless and Mobile Systems,* pp. 111-117, 2005.

[7] R.P. Dick, D.L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. Sixth Int'l Workshop Hardware/Software Codesign,* pp. 97-101, 1998.

[8] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the World with Wireless Sensor Networks," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing,* pp. 2033-2036, 2001.

[9] H.S. Kim, T.F. Abdelzaher, and W.H. Kwon, "Dynamic Delay-Constrained Minimum-Energy Dissemination in Wireless Sensor Networks," *ACM Trans. Embedded Computing Systems,* vol. 4, pp. 679-706, 2005.

[10] Y.-K. Kwok and I. Ahmad, "Benchmarking the Task Graph Scheduling Algorithms," *Proc. 12th Int'l Parallel Processing Symp.,* pp. 531-537, 1998.

[11] X. Lu, H. Hassanein, and S. Akl, "Energy-Aware Dynamic Task Allocation in Mobile Ad Hoc Networks," *Proc. Int'l Conf. Wireless Networks, Comm. and Mobile Computing,* pp. 534-539, 2005.

[12] J. Luo and N.K. Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems," *Proc. IEEE/ACM Int'l Conf. Computer Aided Design,* pp. 357-364, 2000.

[13] S. Malik, M. Martonosi, and Y.S. Li, "Static Timing Analysis of Embedded Software," *Proc. ACM Design Automation Conf.,* pp. 147-152, 1997.

[14] C.X. Mavromoustakis and H.D. Karatza, "Handling Delay Sensitive Contents Using Adaptive Traffic-Based Control Method for Minimizing Energy Consumption in Wireless Devices," *Proc. 38th Ann. ACM Symp. Simulation,* pp. 295-302, 2005.

[15] C. Meesookho, S. Narayanan, and C.S. Raghavendra, "Collaborative Classification Applications in Sensor Networks," *Proc. Second IEEE Sensor Array and Multichannel Signal Processing Workshop,* pp. 370-374, 2002.

[16] M.J. Miller, C. Sengul, and I. Gupta, "Exploring the Energy-Latency Trade-Off for Broadcasts in Energy-Saving Sensor Networks," *Proc. 25th IEEE Int'l Conf. Distributed Computing Systems,* pp. 17-26, 2005.

[17] S. Mohanty and V.K. Prasanna, "A Hierarchical Approach for Energy Efficient Application Design Using Heterogeneous Embedded Systems," *Proc. Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems,* pp. 243-254, 2003.

[18] A.B. Olsen, F.H.P. Fitzek, and P. Koch, "Energy Aware Computing in Cooperative Wireless Networks," *Proc. Int'l Conf. Wireless Networks, Comm. and Mobile Computing,* pp. 16-21, 2005.

[19] S. Park, V. Raghunathan, and M.B. Srivastava, "Energy Efficiency and Fairness Tradeoffs in Multi-Resource Multi-Tasking Embedded Systems," *Proc. ACM Int'l Symp. Low Power Electronics and Design,* pp. 469-474, 2003.

[20] V. Paruchuri, A. Durresi, and L. Barolli, "Energy-Aware Routing Protocol for Heterogeneous Wireless Sensor Networks," *Proc. 16th Int'l Workshop Database and Expert Systems Applications,* pp. 133-137, 2005.

[21] A. Rădulescu and A.J.C. Gemund, "On the Complexity of List Scheduling Algorithms for Distributed-Memory Systems," *Proc. 13th Int'l Conf. Supercomputing,* pp. 68-75, 1999.

[22] V. Raghunathan, C.L. Pereira, M.B. Srivastava, and R.K. Gupta, "Energy-Aware Wireless Systems with Adaptive Power-Fidelity Tradeoffs," *IEEE Trans. Very Large Scale Integration Systems,* vol. 13, pp. 211-225, 2005.

[23] V. Raghunathan, C. Schurgers, P. Sung, and M.B. Srivastava, "Energy-Aware Wireless Microsensor Networks," *IEEE Signal Processing Magazine,* vol. 19, pp. 40-50, 2002.

[24] S. Ranaweera and D.P. Agrawal, "A Task-Duplication-Based Scheduling Algorithm for Heterogeneous Systems," *Proc. 14th IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 445-450, 2000.

[25] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing Sensor Networks in the Energy-Latency-Density Design Space," *IEEE Trans. Mobile Computing,* vol. 1, pp. 70-80, 2002.

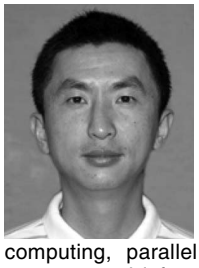
[26] Z. Shao, "High Performance, Low Power and Secure Embedded Systems," PhD dissertation, Dept. of Computer Science, Univ. of Texas, Dallas, 2005.

[27] T. Simunic, L. Benini, G.D. Micheli, and M. Hans, "Source Code Optimization and Profiling of Energy Consumption in Embedded Systems," *Proc. 13th Int'l Symp. System Synthesis,* pp. 193-198, 2000.

[28] M. Singh and V.K. Prasanna, "A Hierarchical Model for Distributed Collaborative Computation in Wireless Sensor Networks," *Proc. 17th IEEE Int'l Parallel and Distributed Processing Symp.,* 2003.

[29] C.M. Woodside and G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, pp. 164-174, 1993.

[30] T. Xie and X. Qin, "A New Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters," *Proc. Seventh IEEE Int'l Conf. Cluster Computing,* 2005.
[31] T. Xie, X. Qin, and M. Nijim, "Solving Energy-Latency Dilemma: Task Allocation for Parallel Applications in Heterogeneous Embedded Systems," *Proc. 35th Int'l Conf. Parallel Processing,* pp. 12-22, 2006.
[32] M. Younis, M. Youssef, and K. Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks," *Proc. 10th IEEE Int'l Symp. Modeling, Analysis and Simulation of Computer and Telecomm. Systems,* pp. 129-136, 2002.
[33] M. Youssef, M. Younis, and K. Arisha, "A Constrained Shortest-Path Energy-Aware Routing Algorithm for Wireless Sensor Networks," *Proc. IEEE Wireless Comm. and Networks Conf.,* pp. 794-799, 2002.
[34] Y. Yu, B. Krishnamachari, and V.K. Prasanna, "Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks," *Proc. IEEE INFOCOM '04,* pp. 244-255, 2004.
[35] Y. Yu and V.K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *Mobile Networks and Applications,* vol. 10, pp. 115-131, 2005.
[36] D. Zhu, R. Melhem, and D. Mossé, "The Effects of Energy Management on Reliability in Real-Time Embedded Systems," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design,* pp. 35-40, 2004.



Tao Xie received the BSc and MSc degrees from Hefei University of Technology, China, in 1991 and 2000, respectively, and the PhD degree in computer science from the New Mexico Institute of Mining and Technology in 2006. He is currently an assistant professor in the Department of Computer Science at San Diego State University, California. His research interests include security-aware scheduling, high-performance computing, cluster and grid computing, parallel and distributed systems, real-time/embedded systems, and information security. He is a member of the IEEE and the IEEE Computer Society.

Xiao Qin received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1996 and 1999, respectively, and the PhD degree in computer science from the University of Nebraska, Lincoln, in 2004. He is a currently an assistant professor of computer science at Auburn University. Prior to joining Auburn University in 2007, he was with the New Mexico Institute of Mining and Technology for three years. He has been on the program committees of various international conferences, including IEEE Cluster, IPCCC, and ICPP. From 2000 to 2001, he was a subject area editor of the *IEEE Distributed Systems Online*. His research interests include parallel and distributed systems, real-time computing, storage systems, and fault tolerance. He is a member of the IEEE and the IEEE Computer Society. He received a US National Science Foundation Computing Processes and Artifacts (CPA) Award in 2007.