Solving Energy-Latency Dilemma: Task Allocation for Parallel Applications in Heterogeneous Embedded Systems

Tao Xie
Department of Computer Science
San Diego State University
San Diego, California 92182
xie@cs.sdsu.edu

Xiao Qin, Mais Nijim
Department of Computer Science
New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801
{xqin,mais@cs.nmt.edu}

Abstract

Parallel applications with energy and low-latency constraints are emerging in various networked embedded systems like digital signal processing, vehicle tracking, and infrastructure monitoring. However, conventional energy-driven task allocation schemes for a cluster of embedded nodes only concentrate on energy-saving when making allocation decisions. Consequently, the length of the schedules could be very long, which is unfavorable or in some situations even not tolerated. In this paper, we address the issue of allocating a group of parallel tasks on a heterogeneous embedded system with an objective of energy-saving and short-latency. A novel task allocation strategy, or BEATA (Balanced Energy-Aware Task Allocation), is developed to find an optimal allocation that minimizes overall energy consumption while confining the length of schedule to an ideal range. Experimental results show that BEATA significantly improves the performance of embedded systems in terms of energy-saving and schedule length over an existing allocation scheme.

1. Introduction

All A parallel application consists of a number of tasks that cooperate with each other through communications to fulfill a common mission [5][10]. In the last decade, networked embedded systems have become increasingly popular as platforms for executing parallel applications such as target tracking and infrastructure monitoring [1][4]. Much of this trend can be attributed to rapid advances in processing energy, network bandwidth, and storage capacity. However, embedded systems usually have low energy capacities operating in distributed mobile or wired environments [8][9][15]. Therefore, energy-saving became a critical issue for these systems.

To address this challenge, extensive researches have been conducted to reduce overall energy consumption for a variety of embedded systems using diverse techniques [6][8][9][11][12][15][16]. Source code optimization and profiling were exploited in [12] to minimize energy consumption in embedded systems. Zhu *et al.* devised a mechanism to increase reliability and reduce energy consumption of real-time embedded systems by slack time reclamation [16]. Park *et al*. tried to make a balance between energy efficiency and fairness in multi-resource for multi-tasks in embedded system [9]. A hierarchical approach for energy efficient application design using heterogeneous embedded systems was proposed by Mohanty *et al.*[8]. In [15], Yu *et al.* proposed an energy-balanced task allocation scheme for parallel processing in homogeneous wireless sensor networks with a goal to maximize the lifetime of the entire system. In particular, most of recent researches in energy-saving for embedded systems share two common features (1) applications considered are real-time in nature with hard deadlines; and (2) energy-saving is achieved by employing DVS (Dynamic Voltage Scaling). Our work is fundamentally different from the above approaches as we focus on reducing both energy consumption and response time for soft real-time parallel applications running on heterogeneous embedded systems with no DVS available. Without loss of generality, we assume that different processing nodes have distinct fixed energy consumption rates. Similarly, different communication channels also have various energy assumption rates. The goal of this work is to develop a task allocation strategy that not only conserves energy but also generates a short schedule, which is favorable or even necessary in some scenarios. For example, in a soft real-time embedded system such as a cellular phone [7], it must be able to encode outgoing voice and decode incoming signal during a conversation in a timely manner. Occasional glitches in conversations due to tardy response are not desired. When the response time becomes longer frequent glitches could happen, which are not tolerated at all.

Energy-saving and low-latency, however, are two conflicting objectives in the context of allocating a parallel application represented by a task graph onto a set of connected heterogeneous processing nodes in an embedded system. The dilemma arises from a multidimensional heterogeneity bearing by a heterogeneous embedded system (see Section 2.1). Specifically speaking, a processing node that provides a task with earliest finish time may not be an ideal candidate in terms of energy-saving. This is because the execution time of a task allocated on an embedded node is irrelative to the energy consumption rate offered by the node. Moreover, the computational energy consumption of a task allocated on a node is a product of energy consumption rate of the node and execution time of the task. The motivation of this work is to solve the energy-latency dilemma existed in networked heterogeneous embedded system where both energy-saving and low-latency need to be achieved. In this paper, we address the issue by minimizing energy consumption while confining schedule lengths. To this end, we devised a energy-adaptive window to control the trade-off between energy consumption and response time. Experimental results demonstrate that our scheme is effective in a heterogeneous embedded system.

The main contributions of this paper are: (1) an energy-latency driven task allocation scheme BEATA for parallel applications on heterogeneous embedded systems; (2) an energy consumption model for quantitatively measuring energy cost introduced by both computation and communications; and (3) a simulated heterogeneous embedded system where the BEATA strategy is implemented and evaluated. The rest of the paper is organized as follows. In the next section we describe the system model, the task model and energy consumption model. In Section 3, we propose the BEATA scheme for parallel applications running on heterogeneous embedded systems. We present in Section 4 experimental results based on synthetic benchmarks and a real world application. Section 5 concludes the paper with summary and future directions.

2. System models

We describe in this section mathematical models, which were built to represent a task allocation framework, parallel applications with precedence constraint, and energy consumption model.

2.1. The networked embedded system

A networked embedded system in the most general form consists of a set, e.g., $P = \{p_1, p_2, ..., p_m\}$, of heterogeneous embedded computing nodes (hereinafter referred to as nodes or embedded nodes) connected by a single-hop wired or wireless network. The network embedded system can be represented as a graph of nodes along with their point-to-point links. In the system, an embedded node is modelled 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 Joule per unit time. With respect to energy conservation, each network link is characterized by its energy consumption rate that heavily relies on the link's transmission rate, which is modelled by weight w_{ij} of the edge between node p_i and p_j. 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_{ij} in X is "1" if task t_i is assigned to node p_j and is "0", otherwise. Heterogeneity investigated in this study embrace multiple meanings. First, execution times of a task on different embedded nodes may various, since the nodes may have different processing capabilities. Second, a node offering 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, energy consumption rates of the nodes may not necessarily be identical. For sake of simplicity and without any loss of generality, 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.

2.2. The task model

Applications with dependent tasks can be modelled by Directed Acyclic Graphs (*DAGs*) [14]. Throughout this paper, a parallel application is specified as a pair, i.e, (T, E), where $T = \{t_1, t_2, ..., t_n\}$ represents a set of non-preemptable tasks, E is a set of weighted and directed edges representing communications among

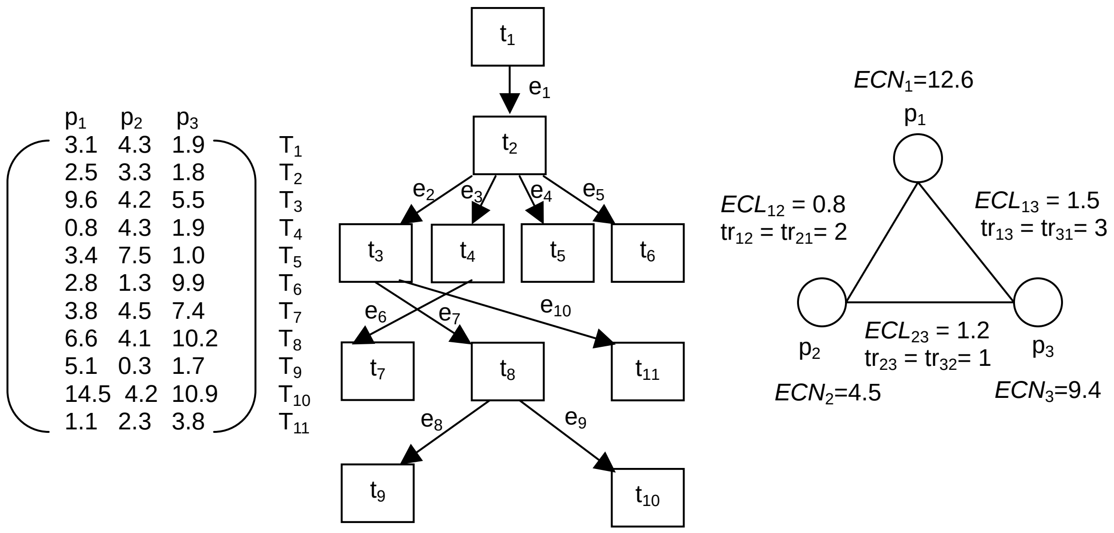


Figure 1. Example task and networked embedded system. *ECN$_i$* is the energy consumption rate of node *i*, and *ECL$_{ij}$* is the energy consumption of a link between node *i* and *j*.

tasks, e.g., $(t_i, t_j) \in E$ is a message transmitted from task t_i to t_j. Precedence constraints of the parallel application are represented by all edges in E. 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}, where s_{ij} is the data size of the message and b_{uv} is the transmission rate of a link connecting p_u and p_v. The execution time of task t_i is modelled by a vector, i.e., $c_i = \left(c_i^1, c_i^2, \cdots, c_i^m\right)$, where c_i^j represents the execution time t_i on the jth embedded node.

Example 1. Figure 1 illustrates an example task graph and an example networked embedded system. The task graph has eleven tasks and the processor graph has three processors. The transmission rate and energy consumption rate of the channel between processor 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 as below. For example, task t_1 has execution time 3.1 second, 4.3 second, and 1.9 second on processor p_1, p_2, and p_3, respectively.

2.3. Energy consumption model

Let e_i^j be an energy dissipation caused by task t_i running on node p_j. We denote the energy consumption rate of the jth node when it is active by ECN_j^{active}, and the energy dissipation e_i^j can be written as below

$$e_i^j = ECN_j^{active} \cdot c_i^j . \tag{1}$$

The energy consumption rate of a networked embedded system is represented by a vector. Given a parallel application with task set T and allocation matrix X, we obtain the total energy consumed by all tasks of the application from Eq. (2).

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

We assume in Eq. (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, i.e., $ECN^{idle} = \left(ECN_1^{idle}, ECN_2^{idle}, \cdots, ECN_m^{idle}\right)$, where ECN_j^{idle} as an energy consumption rate of node j when it is inactive. Additionally, we define f_i as the complete time of task t_i. Then, we obtain the analytical formula for the energy consumed by the embedded nodes when they are idle:

$$en^{idle}\left(T, X, ECN^{idle}\right)=\sum_{j=1}^{m} ECN_j^{idle} \cdot\left(\max_{i=1}^{n}\left(f_i\right)-\sum_{i=1}^{n} x_{ij} \cdot c_i^j\right), \quad (3)$$

where $\max_{i=1}^{n}\left(f_i\right)$ is the schedule length (also referred to as makespan time), and $\max_{i=1}^{n}\left(f_i\right)-\sum_{i=1}^{n} x_{ij} \cdot c_i^j$ is the total idle time on node *j*. Eq. (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 Eqs. (2) and (3) as

$$\begin{aligned} &en\left(T, X, ECN^{active}, ECN^{idle}\right)= \\ &en^{active}(T, X, ECN^{active})+en^{idle}(T, X, ECN^{idle}), \end{aligned} \quad (4)$$

Similarly, let $\hat{e}_{ij}$ denote the energy consumption of a message $(t_i, t_j) \in E$. Suppose t_i and t_j are respectively allocated to node *u* and *v*, we can express the energy consumption $\hat{e}_{ij}$ as

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

where $ECL_{uv}\left(b_{uv}\right)$ is the energy consumption rate of the link between node u and 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 tradeoffs function presented in [1].

The energy consumption rate of the network links can be modelled by an $m \times m$ matrix $ECL^{active}=\left(ECL_{uv}^{active}\right)$, 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 Eq. (5). Then, we have

$$\begin{aligned} el_{uv}^{active}\left(T, X, ECL^{active}\right) &=\sum_{(e_i, e_j \in L_{uv})} \hat{e}_{ij} \\ &=\sum_{(e_i, e_j \in L_{uv})} ECL_{uv}^{active}\left(b_{uv}\right) \cdot \frac{s_{ij}}{b_{uv}} \\ &=\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x_{iu} \cdot x_{jv} ECL_{uv}^{active}\left(b_{uv}\right) \cdot \frac{s_{ij}}{b_{uv}}, \end{aligned} \quad (6)$$

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 $L_{uv}=\left\{\forall(t_i, t_j) \in E, 1 \leq u, v \leq m \mid s_{ij}>0 \wedge x_{iu}=1 \wedge x_{jv}=1\right\}$.

It is assumed in Eq. (6) that all 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 message to be transmitted at various rates, depending on an underlying energy-aware message scheduling mechanism, which we recently developed [1]. 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

$$\begin{aligned} &el_{uv}^{active}\left(T, X, ECL\right)= \\ &\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x_{iu} \cdot x_{jv} \cdot ECL_{uv}^{active}\left(b_{uv}^{ij}\right) \cdot \frac{s_{ij}}{b_{uv}^{ij}}. \end{aligned} \quad (7)$$

The energy consumption of links, i.e, $el^{active}(T, X, ECL^{active})$, in the networked embedded system is derived from Eq. (7). Specifically, $el^{active}(T, X, ECL^{active})$ is equivalent to the summation of all the links energy consumption. Thus, $el^{active}(T, X, ECL^{active})$ can be expressed as

$$\begin{aligned} &el^{active}\left(T, X, ECL^{active}\right) \\ &=\sum_{u=1}^{m} \sum_{v=1, v \neq u}^{m} el_{uv}^{active}\left(T, X, ECL_{uv}^{active}\right) \\ &=\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \sum_{u=1}^{m} \sum_{v=1, v \neq u}^{m} x_{iu} \cdot x_{jv} \cdot ECL_{uv}^{active}\left(b_{uv}^{ij}\right) \cdot \frac{s_{ij}}{b_{uv}^{ij}} \end{aligned} \quad (8)$$

Again, we assume in Eq. (8) 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 the cause of an application's execution. An 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:

$$\begin{aligned} &el_{uv}^{idle}\left(T, X, ECL^{idle}\right)= \\ &ECL_{uv}^{idle} \cdot\left(\max_{i}^{n}\left(f_i\right)-\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x_{iu} \cdot x_{jv} \cdot \frac{s_{ij}}{b_{uv}^{ij}}\right) \end{aligned} \quad (9)$$

where $\max_i^n (f_i) - \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x_{iu} \cdot x_{jv} \cdot \frac{s_{ij}}{b_{uv}^{ij}}$ is the total idle time over the link, and el_{uv}^{idle} is computed as a product of the consumption rate and idle period of the link.

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

$$el^{idle}(T, X, ECL) = \sum_{u=1}^{m} \sum_{v=1, v \neq u}^{m} ECL_{uv}^{idle} \left(\max_i^n (f_i) - \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} x_{iu} \cdot x_{jv} \cdot \frac{s_{ij}}{b_{uv}^{ij}} \right). \quad (10)$$

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

$$el(T, X, ECL^{active}, ECL^{idle}) = el^{active}(T, X, ECL^{active}) + el^{idle}(T, X, ECL^{idle}), \quad (11)$$

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

$$e(T, X, ECN^{active}, ECN^{idle}, ECL^{active}, ECL^{idle}) = en(T, X, ECN^{active}, ECN^{idle}) + el(T, X, ECL^{active}, ECL^{idle}) \quad (12)$$

3. The BEATA algorithm

In an effort to reduce an overall energy consumption of the heterogeneous system, we designed the BEATA algorithm, which aims at blending an energy conservation scheme with task allocation for networked embedded systems that are heterogeneous in nature. To make the best trade-off between energy-saving and schedule lengths, we employ an energy-adaptive window, within which a node is chosen for each task in a way to offer lower energy consumption and earlier finish time of the task.

Now we present the BEATA algorithm in Figure 2 BEATA is conducive to increasing heterogeneous nodes' lifetime while maintaining high performance in terms of makespan time for parallel applications running on networked embedded systems. In other words, BEATA can increase processing nodes' lifetimes by dramatically reducing energy dissipation (see Step 14). Before minimizing the energy consumption of task t_i, BEATA organizes all the nodes in a non-decreasing order in terms of t_i's earliest finish time (see Eq. 16). Step 8 determines the energy consumption incurred by the task on a node, whereas Steps 9-10 calculate the energy consumed by all the messages received by the task from its predecessors. Among all the candidate nodes listed in the energy-adaptive window, Step 14 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 15 allocates the task to the best candidate node. After the allocation of the task is accomplished, Step 16 updates the schedule of the node to which the task is allocated.

```
1.for each task $t_i \in T$ do
2.    for each node $p_u \in P$ in the system do
3.        Compute $est_u(t_i)$
4.        Compute $f_u(t_i)$ (see Eq. 15)
5.    end for
6.    Sort all nodes in finish time of $t_i$
7.    for each node in energy-adaptive window do
8.        Compute energy consumption of $t_i$
9.        for each $t_i$'s predecessor $t_j$, do
10.           Compute the energy consumption
              cause by message $(t_j, t_i)$
11.           Compute the total energy consumed
              by $t_i$ and the messages sent from
              the predecessors
12.       end for
13.   end for
14.   Select $p_v$ in energy-adaptive window that
      offers the smallest energy consumption for $t_i$
      and messages sent from $t_i$'s predecessors
15.   Assign $t_i$ to $p_v$
16.   Update the schedule on node $p_v$
17.   Compute the energy consumed by $t_i$ on $p_v$
          and the messages received by $t_i$
18.   Record start time and finish time for task $t_i$
19.end for
```

Figure 2. The BEATA algorithm.

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 leading to the final expressions for these two parameters. Suppose task t_i has only one predecessor task t_j, the earliest available time $eat_u(t_j, t_i)$ of t_i relies on (a) the finish time f_j of t_j, (b) the message start time, $mst(t_j, t_i)$, and (c) 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 the tasks are allocated to the same embedded node, the transmission time is negligible. Thus, $eat_u(v_j, v_i)$ is expressed as

$$eat_u(v_j, v_i) = \begin{cases} f_j & \text{, if } p_u = p_v \\ mst(t_j, t_i) + s_{ji} / b_{vu} & \text{, otherwise} \end{cases} \quad (13)$$

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

$$eat_u(t_i) = \max_{(t_j, t_i) \in E} \{eat_u(t_j, t_i)\}. \quad (14)$$

Now we are positioned to derive the earliest start time $est_u(t_i)$, which is computed based on $eat_u(t_i)$. More specifically, $est_u(t_i)$ is calculated by checking the schedule on p_u to identify an idle time slot that starts later than the task's $eat_u(t_i)$ and is large enough to accommodate the task. With the value of $est_u(t_i)$ in place, we can obtain the finish time of ti on pu using Eq. (15). The finish time equals to the summation of the earliest start time $est_u(t_i)$ and t_i's execution time on p_u.

$$f_u(t_i) = est_u(t_i) + c_i^u. \quad (15)$$

The following theorem gives the time complexity of the proposed BEATA algorithm.

Theorem 1. Given a networked embedded system and a parallel application represented as a task graph. The time complexity of *BEATA* is $O(nmlgm+nkq)$, where n is the number of tasks, m is the number of nodes, k is the energy-adaptive window size, and q is the maximum in-degrees of the task graph.

Proof. It takes $O(m)$ time to compute the earliest start times and earliest finish times for a task on all 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 7-13). Other steps simply take $O(1)$ time. Hence, the time complexity of the BEATA algorithm is given as follows: $O(n)(O(m) + O(mlgm) + O(kq)) = O(nmlgm+nkq)$.

4. Performance evaluation

Now we are in a position to evaluate the effectiveness of the proposed energy-latency driven task allocation scheme. To demonstrate the strength of BEATA, we compare it with the list scheduling scheme, which is a well-known scheduler for parallel applications. The LIST algorithm is briefly described below.

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)
Energy consumption rate heterogeneity	1.2 (see Eq. 16)
Standard node energy consumption rate	200 mW
Communication energy consumption rate	The energy-transmission time model in [1].

Table 1. System parameters.

LIST: The most common heuristic for DAG scheduling in a heterogeneous system. For each task allocation, it chooses the computing node that can offer the task earliest finish time considering both computation time and communication time. Its goal is to generate a schedule for a DAG with the shortest length.

4.1. Simulation setup

Before presenting empirical results, we present the simulation model as follows. Table 1 summarizes the configuration parameters of simulated networked embedded systems used in our experiments. The parameters of computing nodes in the networked embedded systems are chosen to resemble real-world processors like Intel StrongARM 1100. The relationship between energy rate and transmission rate is 100 mW at 100 Kbps, which means the time and energy cost for transmitting one bit are around 10 µsec and 1 µJoule [1]. All synthetic parallel jobs used from Section 4.2 to Section 4.3 were created by TGFF [3], a randomized task graph generator [13].

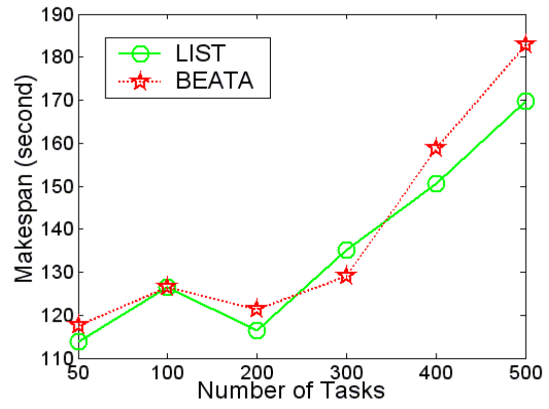
Although number of tasks, number of computing nodes, out degree, and task execution time are synthetically generated, we examined impacts of these important workload parameters on system performance by controlling the parameters. The performance metrics by which we evaluate system performance include:

- *Makespan* (the latest task completion time in the task set represented by a DAG).
- *Energy consumption*: total energy consumed by the task set including computation energy consumption and communication energy consumption (see Eq. 12).

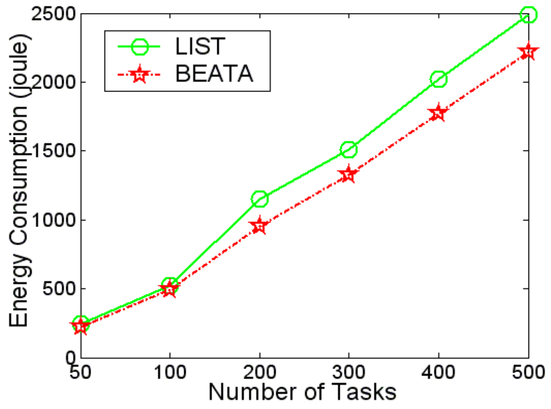
- *Utilization standard deviation (USD)*: standard deviation of computing nodes utilization in the simulated networked embedded systems.
- *Energy standard deviation (PSD)*: standard deviation of computing nodes energy consumption in the simulated networked embedded systems.

4.2. Overall performance comparisons

The goal of this experiment is to compare the proposed BEATA algorithm against the conventional list scheduling scheme to understand the sensitivity of the two heuristics to the number of tasks in a DAG. We tested 6 task graphs with the number of tasks varying from 50 to 500 with precedence constraints.



(a)



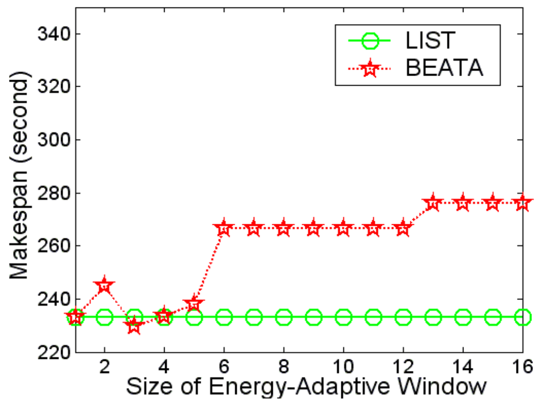
(b)

Figure 3. Performance impact of number of tasks.

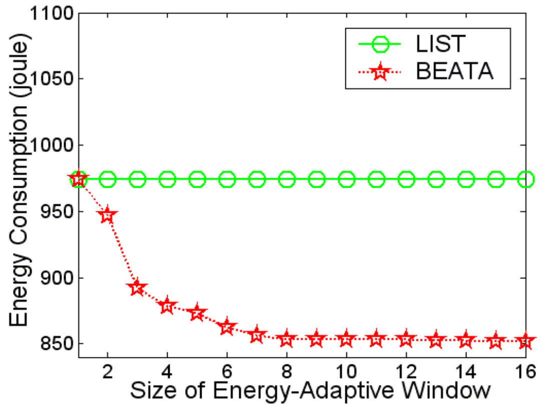
We observe from Figure 3 (a) that BEATA and LIST exhibit very similar performance in terms of makespan. An interesting observation is that BEATA even generates a shorter schedule than LIST when the number of tasks is 300. The “anomaly” can be explained by the fact that the LIST algorithm cannot guarantee the shortest schedule in a heterogeneous system due to lack of the information about tasks not yet scheduled and the varying execution times for each task on different computing nodes. Compared with LIST, BEATA on average only increases makespan by 2.9% but saves energy by 12.1%. Figure 3 (b) reveals that BEATA consistently performs better than LIST in terms of energy consumption.

4.3 Sensitivity to energy-adaptive windows

To verify the performance impact of energy-adaptive window, we evaluate the performance as functions of size of energy-adaptive window. Since LIST does not have an energy-adaptive window, its performance in all metrics keeps constant.



(a)



(b)

Figure 4. Performance impact of size of energy-adaptive window.

The results from Figure 4 validate the relationships between the two algorithms described in Section 3.

When energy-adaptive window was set to 1, BEATA degraded to LIST. We observe from Figure 4 that BEATA achieves an excellent trade-off between makespan and energy consumption when the size of energy-adaptive window falls in the range [3, 5]. Within this range, BEATA in terms of makespan achieves almost the same performance as LIST (on average merely 0.42% longer), while it can save energy up to 10.4%.

5. Conclusions

In this paper, we address the issue of allocating tasks of parallel applications in heterogeneous embedded systems with an objective of energy-saving and latency-reducing. BEATA (Balanced Energy-Aware Task Allocation), a task allocation scheme considering both energy consumption and schedule length, is developed to solve the energy-latency dilemma. To facilitate the presentation of BEATA, we also proposed mathematical models to describe a system framework, parallel applications with precedence constraints, and energy consumption model. We conducted extensive experiments using a real world application as well as synthetic benchmarks. The experimental results show that BEATA significantly improves the performance in terms of energy dissipation and makespan time over an existing allocation scheme. Compared with LIST, BEATA achieves improvement in energy-saving on averages of 12.1% with only 2.9% increase in makespan.

Future studies in this research can be performed in the following directions. First, we will extend our scheme to multi-dimensional computing resources from which energy-saving 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 parallel applications, where the hard deadlines must be guaranteed.

Acknowledgements

The work reported in this paper was supported in part by the New Mexico Institute of Mining and Technology under Grant 103295 and by Intel Corporation under Grant 2005-04-070.sible.

References

[1] M. Alghamdi, T. Xie, X. Qin, “PARM: A Power-Aware Message Scheduling Algorithm for Real-Time Wireless Networks,” *ACM Workshop Wireless Multimedia Networking and Performance Modeling,* Canada, 2005.

[2] W.S. Conner, L. Krishnamurty, R. Want, “Making Everyday Life Easier Using Dense Sensor Networks,” *Ubicomp*, pp. 49-55, 2001.

[3] R. P. Dick, D. L. Rhodes, and W. Wolf, “TGFF:Task graphs for free,” *Proc. Int’l Workshop. Hardware/Software Codesign*, pp. 97-101, Mar. 1998.

[4] D. Estrin, L. Girod, G. Pottie, M. Srivastava, “Instrumenting the world with wireless sensor networks,” *Proc. Int’l Conf. Acoustics, Speech, and Signal Processing,* Salt Lake City, Utah, May 2001.

[5] L. He, A. Jatvis, and D. P. Spooner, “Dynamic scheduling of parallel real-time jobs by modelling spare capabilities in heterogeneous clusters,” *Proc. Int’l Conf. Cluster Computing*, pp. 2-10, Dec. 2003.

[6] J. Luo, N.K. Jha, “Energy-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems”, *IEEE/ACM Int’l Conf. Computer Aided Design*, 2000.

[7] S. Malik,M. Martonosi, Y.S. Li, "Static Timing Analysis of Embedded Software", *Design Automation Conference*", pp.147-152, 1997.

[8] S. Mohanty, V.K. Prasanna, “A hierarchical approach for energy efficient application design using heterogeneous embedded systems,” *CASES*, pp. 243-254, 2003.

[9] S. Park, V. Raghunathan, M. B. Srivastava, "Energy Efficiency and Fairness Tradeoffs in Multi-Resource Multi-Tasking Embedded Systems", *ACM Int’l Symp. Low Energy Electronics and Design*, August 2003.

[10] X. Qin and H. Jiang, “Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems,” *Proc. 30th Int’l Conf. Parallel Processing,* pp.113-122, 2001.

[11] Z. Shao, "High performance, low energy and secure embedded systems", *Ph.D. Dissertation*, Department of Computer Science, University of Texas at Dallas, 2005.

[12] T. Simunic, L. Benini, G. D. Micheli, M. Hans, “Source code optimization and profiling of energy consumption in embedded systems,” *Int’l Symp. System Synthesis*, 2000.

[13] C.M. Woodside and G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", *IEEE Trans. on Parallel and Distributed Systems,* Vol. 4, No. 2, pp. 164-174, Feb. 1993.

[14] T. Xie and X. Qin, “A New Allocation Scheme for Parallel Applications with Deadline and Security Constraints on Clusters,” *Proc. 7th IEEE Int’l Conf. Cluster Computing,* Boston, USA, 2005.

[15] Y. Yu, V. K. Prasanna, "Energy-balanced task allocation for parallel processing in wireless sensor networks," *Mobile Networks and Applications*, Vol. 10, pp. 115-131, 2005.

[16] D. Zhu, R. Melhem, D. Mossé, “The Effects of Energy Management on Reliability in Real-Time Embedded Systems,” *Int’l Conf. Computer Aidded Design*, San Jose, Nov. 2004.