Energy-Efficient Task Mapping for Data-Driven Sensor Network Macroprogramming*

Animesh Pathak and Viktor K. Prasanna

Ming Hsieh Department of Electrical Engineering,
University of Southern California, USA
{animesh,prasanna}@usc.edu

Abstract. Data-driven macroprogramming of wireless sensor networks (WSNs) provides an easy to use high-level task graph representation to the application developer. However, determining an energy-efficient initial placement of these tasks onto the nodes of the target network poses a set of interesting problems. We present a framework to model this task-mapping problem arising in WSN macroprogramming. Our model can capture task placement constraints, and supports easy specification of energy-based optimization goals. Using our framework, we provide mathematical formulations for the task-mapping problem for two different metrics — energy balance and total energy spent. Due to the complex nature of the problems, these formulations are not linear. We provide linearization heuristics for the same, resulting in mixed-integer programming (MIP) formulations. We also provide efficient heuristics for the above. Our experiments show that the our heuristics give the same results as the MIP for real-world sensor network macroprograms, and show a speedup of up to several orders of magnitude.

1 Introduction

Various high-level programming abstractions have been proposed recently to assist in application development for Wireless Sensor Networks (WSNs). Specifically, *Data-driven macroprogramming* [1] refers to the general technique of specifying the WSN application from the point of view of data-flow. In sense-and-respond applications such as traffic management [2], building environment management [3], target tracking etc., the system can be represented as a set of tasks running on the system's nodes – producing, processing and acting on data items or streams to achieve the system's goals. The mapping of these tasks onto the nodes of the underlying system (details of which are known at compile time) is an important part of the compilation of the macroprogram, and optimizations can be performed at this stage for energy-efficiency.

Although the initial information (positions, energy levels) about the target nodes is known, during the lifetime of the WSN, changing conditions, either external or internal may alter the circumstances. We do not address these unpredictable situations, and instead aim to provide a "good" initial mapping of tasks. We assume that during the lifetime of the system, remapping of tasks will occur to face these circumstances, for

* This work is partially supported by the National Science Foundation, USA, under grant number CCF-0430061 and CNS-0627028.

S. Nikoletseas et al. (Eds.): DCOSS 2008, LNCS 5067, pp. 516–524, 2008.

example, a distributed task-remapping algorithm can be triggered when the energy at any node goes below a certain fraction of its initial energy level. Our work attempts to utilize the global knowledge available at compile-time to obtain efficient results.

In this paper, we make three **contributions**. In Sect. 2 we provide a **modeling framework** for the problem of task-mapping for data-driven sensor network applications. In Sect. 3 we propose a **mixed integer programming (MIP) formulation** to obtain task mappings in order to optimize for the energy balance and total-energy minimization goals. Since the formulation is non-linear, we provide substitution-based techniques to linearize the MIPs. Although the MIP formulations give optimal results, they may take inordinately large times to terminate for large real-world scenarios. In Sect. 4, we provide **greedy heuristics** for the two problem instances.

Our experimental results, discussed in Sect. 5, show the performance comparison between the techniques, using realistic applications and deployment scenarios. Our heuristics are shown to obtain the optimal solution for these scenarios, while gaining significant speedups over the MIP technique. Section 6 discusses the differences of our work from other closely related work in parallel and distributed systems and sensor networks. Section 7 concludes.

2 Problem Formulation

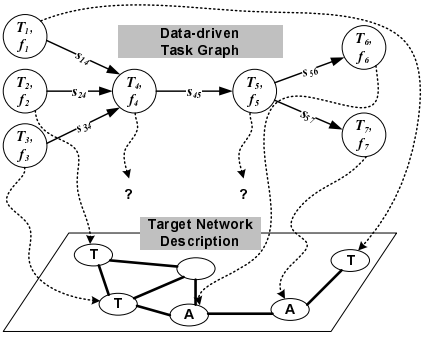


Fig. 1. Temperature mgmt. application

As an example of data-driven macroprogramming representation, consider the following (simple) application – A room is instrumented with six wireless nodes, with three nodes equipped with temperature sensors, and two nodes connected to actuators for controlling the temperature of the room. We need to periodically determine the average temperature in the room, compare it with a threshold, and produce the corresponding actuation. One way of designing such an application at a high-level using a data-driven approach is shown in the top part of Fig. 1. Tasks T_1, T_2 and T_3 are temperature sampling tasks, which fire at a rates of f_1, f_2, f_3 and generate ambient temperature readings of size s_{14}, s_{24}, s_{34}. Task T_4 calculates the average of these readings and feeds it to T_5, which determines the action to be taken. Tasks T_6 and T_7 act upon the data generated by T_5, and control the actuators. The system for which this application is being designed is shown in the lower part of the same figure. The nodes equipped with temperature sensors and actuators are marked with a **T** and **A** respectively.

The mapping of tasks T_1 through T_7 onto the nodes of the target network is an instance of the problem faced while compiling data-driven macroprograms for WSNs. The placement of the sensing tasks (T_1, T_2, T_3) and the actuating tasks(T_6 and T_7) are pre-determined to the nodes with the relevant capabilities. This fact is shown using curved broken lines in the figure. However, tasks T_4 and T_5 can be placed on any of the nodes in the floor, thus allowing for optimizations in this process.

Our aim is to capture the various aspects of systems like the one above, including the placement constraints and firing rates of the tasks, the data-flow between them, the heterogeneity in the nodes and links of the system, and the energy spent in sensing, computation and communication.

Application and System Model

A **Network Description** N represents the target system of nodes where the WSN application is to be deployed. Each node k ($k = 1, \ldots, n$) has an initial energy reserve e_k^0. We assume that the system operates in *rounds*, and denote the energy remaining at node k after t rounds by e_k^t. A **round** is defined as the least time-period after which the system behavior repeats itself.

A **Data-driven Task** i represents the sensing, processing or actuation activity in a WSN. Its firing rate f_i, denotes the number of times it is invoked in one round.

A **Data-driven Task Graph** $D = (DT, DE)$ is a directed acyclic graph (DAG) consisting **a)** A set $DT = \{1, \ldots, i, \ldots, m\}$ of data-driven tasks, and **b)** A set $DE \subseteq DT \times DT$ of edges. Each edge (i, j) is labeled with the size s_{ij} of the data that task i produces for task j upon each invocation.

The **Task Execution Energy Matrix** $\mathcal{T}$ is an $m \times n$ matrix, where $\mathcal{T}_{ik}$ denotes the energy spent by node k per invocation of task i, if i is mapped onto node k.

The **Routing Energy Cost Matrix** $\mathcal{R}$ for N is a $n \times n \times n$ matrix, with $\mathcal{R}_{\beta\gamma k}$ denoting the energy consumed at node k while routing one unit of data from node β to γ.

The **Task Mapping** is a function $M : DT \rightarrow N$, placing task i on node $M(i)$.

Energy Costs: In a sensor network, the *cost* that developers are largely concerned with is the *energy spent* by the nodes as the system operates. We therefore use the terms *cost* to mean the energy spent at a node throughout this paper, unless otherwise stated. Using the model defined above, we compute the following costs[1]. At each node $k \in N$, the **computation cost** in each round is given by

$$C_{\text{comp}}^k = \sum_{i:M(i)=k} f_i \cdot \mathcal{T}_{ik} \tag{1}$$

and the **energy cost of communicating messages** in each round is given by

$$C_{\text{comm}}^k = \sum_{(i,j)\in DE} f_i \cdot s_{ij} \cdot \mathcal{R}_{M(i)M(j)k} \tag{2}$$

Performance Metrics: The above modeling framework can be used to easily model two common optimization goals. The first is *energy balance*, which we consider to be achieved when the maximum fraction of energy spent by any node in the system is minimized.

$$\text{OPT}_1 = \min_{\text{all Mappings } M} \max_{k\in N} \frac{1}{e_o^k} \cdot (C_{\text{comp}}^k + C_{\text{comm}}^k) \tag{3}$$

The second performance goal we model using our framework is the more commonly used *total energy spent* in the entire system. Although we believe that energy balance is

[1] Note that the cost of sensing is included in the T_{ik} of the sensing tasks.

a better metric to measure the quality of task placement, we use the goal of minimizing the total energy spent in the system to illustrate the modeling power of our framework.

$$\text{OPT}_2 = \min_{\text{all Mappings } M} \sum_{k \in N} (C^k_{\text{comp}} + C^k_{\text{comm}}) \tag{4}$$

For each of the two metrics, a *feasible* solution is possible only when all nodes have non-zero energy left at the end of one round. If there are no mappings possible for which this holds, the task-mapping algorithms should report failure. In addition to the above, our framework can be used to model other application scenarios also, e.g. when multiple paths between two nodes are possible.

3 Mathematical Formulations for Task Mapping on WSNs

3.1 Mixed Integer Programming Formulation for OPT_1

To formulate the problem as a mixed integer programming (MIP) problem, we represent task mapping M by an $m \times n$ *assignment matrix* X, where x_{ik} is 1 if task i is assigned to node k, and 0 otherwise. The problem can then be defined as:

Inputs:

- $D = (DT, DE)$: Data-driven Task Graph, f_i: Firing rate for task i, and s_{ij}: Size of data transferred from task i to j on each invocation of i
- N: Network description, $\mathcal{T}$: Task execution energy matrix, $\mathcal{R}$: Routing energy cost matrix

Output: X: Assignment Matrix. x_{ik} is binary.
Optimization Goal:

$$\text{minimize } c$$

Constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for } i = 1, 2, \ldots, m \tag{5}$$

$$\frac{1}{e^0_k}\Big(\sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} + \sum_{(i,j) \in DE} \sum_{\beta=1}^{n} \sum_{\gamma=1}^{n} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot \mathcal{R}_{\beta\gamma k}\Big) \leq c, \forall k \in \{1, \ldots, n\} \tag{6}$$

$$x_{ik} \in \{0, 1\} \text{ for } (i, k) = (1, 1), \ldots, (m, n) \tag{7}$$

$$0 \leq c < 1 \tag{8}$$

The summation terms in (6) denote C^k_{comp} and C^k_{comm} respectively. The final constraint ensures that the MIP fails if no feasible solution exists. Note that the above is an MIP since c is real whereas x_{ik} are binary integers. Also, it is not a linear program since product terms $x_{i\beta} \cdot x_{j\gamma}$ appear in the constraints.

The above problem can be converted to a linear MIP by replacing each $x_{i\beta} \cdot x_{j\gamma}$ term with a binary variable $y_{i\beta j\gamma}$, and adding the following constraints:

$$y_{i\beta j\gamma} - x_{i\beta} \leq 0 \tag{9}$$

$$y_{i\beta j\gamma} - x_{j\gamma} \leq 0 \tag{10}$$

$$x_{i\beta} + x_{j\gamma} - y_{i\beta j\gamma} \leq 1 \tag{11}$$

Using techniques similar to the ones above, we also designed a linear MIP to solve the task-mapping problem for OPT_2, i.e., minimizing the *total energy* spent by the system. Owing to the space limitations, it is not discussed in detail here.

4 Heuristic for Task Mapping

4.1 Greedy Algorithms for Task Mapping

Although the MIP formulation leads to optimal results, solving an MIP can be quite time consuming in practice. Our greedy heuristic for the goal of minimizing the maximum fraction of energy spent at a node (OPT_1) is detailed in Algorithm 1. The main intuition is that the algorithm sorts the edges in the task graph in non-increasing order of the traffic going on them, and then tries to map the still unmapped endpoints of each edge (i, j) so as to achieve the minimum increase in the objective function. We also appropriately modified this algorithm to obtain *GreedyMinTotal()* for OPT_2.

Algorithm 1. *GreedyMinMax*: for OPT_1

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m], s[m][m], e_o[n]$
Output: $M[m]$: Task Assignment

```
1: Initialize M[i] = −1 for i = {1, ..., m}
2: Sort (i, j) ∈ DE in non-increasing order of f[i] · s[i][j]
3: for all (sorted) (i, j) in DE do
4:    minmaxCost = ∞; minPath = (−1, −1) // Initialize minmaxCost and minPath
      for this iteration
5:    for all (α, β) such that (i, j) can be assigned to them do
6:       M[i] = α, M[j] = β // Temporarily assign (i, j) to (α → β)
7:       maxCost = maxCost(D, N, T, R, f, s, e_0, M)
8:       if maxCost < minmaxCost then
9:          minmaxCost = maxCost; minPath = (α, β) // Update minmaxCost and
            minPath
10:   if minmaxCost > 1 then
11:      declare failure. stop. // Checking for feasibility
12:   M[i] = minPath.α; M[j] = minPath.β
13: return M
```

Computational Complexity: Each invocation of *maxCost* takes $\theta(n(m+|DE|))$ time. During Algorithm 1, the sorting takes $\text{O}(|DE| \log(|DE|))$ time, and the main loops invokes Algorithm 2 for evaluating the *maxCost* $\text{O}(|DE|n^2)$ times. The total time complexity of the algorithm is $\text{O}(|DE|(\log(|DE|) + n^3(m + |DE|)))$.

Algorithm 2. *maxCost*: for determining the maximum fraction of energy spent at a node

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m], s[m][m], e_o[n], M[m]$
Output: *maxCost*: Maximum fraction of energy spent at any node

```
1: maxCost = 0 // Initialize max cost
2: for all k ∈ N do
3:    cost = 0 // Initialize node cost
4:    for all i ∈ DT do
5:       if M[i] == k then
6:          cost = cost + f[i] · T[i][k] // Increment computation cost
7:    for all (i, j) ∈ DE such that M[i] ≠ 1 AND M[j] ≠ 1 do
8:       cost = cost + f[i] · s[i][j] · R[M[i]][M[j]][k] // Increment communication cost
9:    if cost/e_0[k] > maxCost then
10:      maxCost = cost/e_0[k]
11: return maxCost
```

4.2 Worst-Case Analysis

Since both *GreedyMinMax* and *GreedyMinTotal* are heuristics, we explored the situations when they can give sub-optimal results. We introduce the notion of the *cost of an algorithm* for this purpose – the cost of *GreedyMinMax* is defined a the maximum fraction of energy spent in one round at any node in N, while the cost of *GreedyMinTotal* is the total energy spent by all the nodes in N in one round.

Theorem 1. *For any integer $\upsilon \geq 1$, there are problem instances for which the cost of* GreedyMinMax *(*GreedyMinTotal*) is arbitrarily close to $\upsilon \times OPT_1$ ($\upsilon \times OPT_2$).*

Proof. Consider a situation as illustrated in Fig. 2. $\mathcal{T}_{ax} = \mathcal{T}_{ay} = 0$, and the other tasks can only be placed on the nodes indicated by the arrows. Let us also assume that $f_a = 1$, $e_0^x = e_0^y = e_0$, and both nodes in N spend one unit of energy per unit of data transmitted on the link between them. Finally, $e_0 \gg \sigma \gg \varepsilon > 0$. The optimal solution, both for OPT_1 and OPT_2, is to place a on node x, thereby causing only the data on the (a, b_0) edge in DE to go on the network, costing σ units of energy to be spent by node x (and the entire system) in each round. The greedy algorithms, however, start with placing the costliest edge (a, b_0) in the best possible manner, co-locating a and b_0 on node y. This leads to $\upsilon \times (\sigma - \varepsilon)$ traffic to go over the $y \rightarrow x$ link. We thus get:

$$\text{OPT}_1 = \frac{1}{e_0}\sigma \tag{12}$$

$$\Rightarrow \text{cost}(GreedyMinMax) = \frac{1}{e_0}\upsilon \times (\sigma - \varepsilon) \approx \upsilon \times \text{OPT}_1 \tag{13}$$

$$\text{Similarly, OPT}_2 = 2\sigma \tag{14}$$

$$\Rightarrow \text{cost}(GreedyMinTotal) = 2\upsilon \times (\sigma - \varepsilon) \approx \upsilon \times \text{OPT}_2 \tag{15}$$

hence proving the theorem □

Theorem 2. *There are problem instances for which* GreedyMinMax *and* GreedyMinTotal *will terminate in failure although a feasible solution exists.*

Proof. Consider the situation as illustrated in Fig. 2. However, in this case, assume that $e_0 = \sigma \gg \varepsilon > 0$. The optimal solution (given by the MIP formulation) will still place task a on node y, while the greedy algorithms will try to place it on node y. Note that for $\upsilon \geq 2$, this will lead to an infeasible solution, as the nodes end up spending $> e_0$ energy and the heuristics will declare failure. □

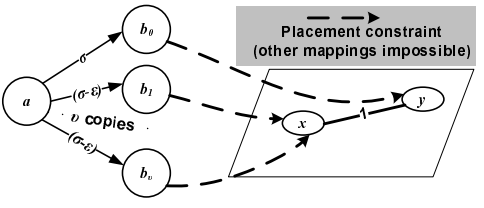
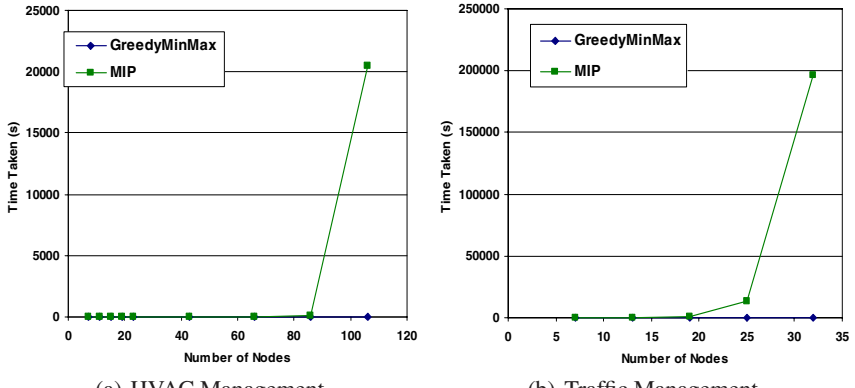


Fig. 2. Scenario for worst case performance of *GreedyMinMax* and *GreedyMinTotal*

5 Evaluation

The worst-case performance bounds discussed above apply to cases where *arbitrary* task graphs and constraints are permitted. However, for realistic sensor network applications, the relationships between the tasks are not completely arbitrary. For evaluating the relative performance of our heuristics in realistic applications, we applied them on the task graphs of the building environment management (HVAC) and traffic management applications discussed in [4]. We used our algorithms to map their tasks onto various simulated target deployments based on real-world scenarios.

In our experiments, we assumed that all nodes started with a sufficiently high initial energy level e_0. The routing energy cost matrix $\mathcal{R}$ was obtained by using a shortest path algorithm on the network, assuming equal energy spent by all nodes on a route, and all data items were assumed to be of unit size ($s_{ij} = 1$). The task execution energy matrix



(a) HVAC Management (b) Traffic Management

Fig. 3. Time taken to compute task-mapping for minimizing maximum energy spent

$\mathcal{T}$ was set up to represent placement constraints: $\mathcal{T}_{ik} = 0$ when task i could be placed on node k, ∞ when it could not. The tasks which performed sensing and actuating were tied to a node with the relevant capabilities. Finally, the f_i for each task was computed as follows: For sensing tasks, f_i was set to 10, and for all other tasks j, f_j was set to the sum of the firing rates of tasks on the other ends of the incoming edges. This represented the fact that task j fires whenever there is data available for it.

We ran our experiments on a PC with dual quad-core Xeon processors running at 2GHz, with 16GB of RAM. We implemented our greedy algorithm in Java, and solved the MIPS using the `lp_solve` linear programming toolkit. We recorded the average time taken for task placement over 500 runs for each data point. The time taken for computing task placements for both the applications for OPT_1 is shown in Fig. 3. For all instances of the traffic application, and the for *all* experiments with OPT_1, the solution given by the greedy algorithm was the same as the one given by the MIP. Our experiments clearly show that the greedy algorithms take much less time that the MIP formulation to determine the mappings. The speedups were similar with OPT_2. This showcases the efficacy of the algorithms in solving the task-mapping problem for complex real-world WSN applications.

6 Related Work

Parallel and Distributed Computing: The task mapping problem [5] is a well studied problem is parallel and distributed computing. In [6], the authors have covered a wide range of mapping problems in such systems and approaches to solve them to minimize system latency in cases where tasks do not have placement constraints. In [7], the authors present a genetic algorithm for placing tasks on a parallel processor, with an extension for the case where not all tasks can be run on all nodes, by way of assigning each node to a class, and associating a class number with each task. Their algorithm is designed to work for a range of metrics, and they focus on the *minimize total execution time* metric in the paper. However, unlike us, they assume full control over routing.

Wireless Sensor Networks: Task placement on sensor networks has also been addressed recently. Efforts such as [8] approach the task-mapping problem for WSNs from a protocol-centric point of view, whereas we take a high-level perspective of the problem. [9] proposes a greedy solution to the *service placement problem*, which is applicable to our context of compiling macroprograms. Similar to our case, their application also has task placement constraints, where certain tasks can be placed only on certain nodes. However, they focus only on the specific goal of minimizing the total energy spent for *trees*. The work in [10] solves the generic role assignment problem, where task placements are specified using *roles*. Their algorithm allows ILP solutions of role assignment onto the nodes of the target system, based on a global optimization criteria represented in terms of the number of nodes with a particular role. Unlike their case, our heuristics are meant for solving an offline version of the problem, and the optimization goals more tied to the energy-consumption at the nodes.

7 Concluding Remarks

In this paper, we formalized the task-mapping problem as it arises in the context of designing applications for wireless sensor networks using data-driven macroprogramming. We provided mathematical formulations for two energy-related optimization goals – minimizing the maximum fraction of energy consumed in a node and minimizing the total energy consumed in the sensor network. We used our modeling framework to provide mathematical formulations to solve these two problem instances, and demonstrated linearization techniques to convert them into mixed-integer programs (MIP). We also provided greedy heuristics for the above problem scenarios, and provided worst-case performance bounds for the same. In spite of the worst-case performance possible for specially crafted problem instances, our heuristics were shown to out-perform the MIP formulation by several orders of magnitudes of time for real-world WSN applications, while not compromising in the quality of the solutions. We acknowledge that later in the life of the WSN applications, distributed protocols will be needed to re-assign the tasks in view of changing operating circumstances. However, our techniques (and other technique based on our models) will provide good initial task placements. Our immediate future work is to reduce the complexity of the greedy approaches, as well as to explore better polynomial time approximation algorithms. Additionally, we are working on integrating our algorithms into the compiler [4] of a pre-existing data-driven macroprogramming framework.

References

1. Bakshi, A., Prasanna, V.K., Reich, J., Larner, D.: The Abstract Task Graph: A methodology for architecture-independent programming of networked sensor systems. In: Workshop on End-to-end Sense-and-respond Systems (EESR) (2005)
2. Hsieh, T.T.: Using sensor networks for highway and traffic applications. IEEE Potentials 23(2) (2004)
3. Dermibas, M.: Wireless sensor networks for monitoring of large public buildings. Technical report, University at Buffalo (2005)
4. Pathak, A., Mottola, L., Bakshi, A., Picco, G.P., Prasanna, V.K.: A compilation framework for macroprogramming networked sensors. In: Int. Conf. on Distributed Computing on Sensor Systems (DCOSS) (2007)
5. Bokhari, S.H.: On the mapping problem. IEEE Transactions on Computers (March 1981)
6. El-Rewini, H., Lewis, T.G., Ali, H.H.: Task scheduling in parallel and distributed systems. Prentice-Hall, Inc., Upper Saddle River (1994)
7. Ravikumar, C., Gupta, A.: Genetic algorithm for mapping tasks onto a reconfigurable parallel processor. In: IEE Proceedings on Computers and Digital Techniques (March 1995)
8. Low, K.H., Leow, W.K., Ang Jr., M.H.: Autonomic mobile sensor network with self-coordinated task allocation and execution. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 36(3), 315–327 (2006)
9. Abrams, Z., Liu, J.: Greedy is good: On service tree placement for in-network stream processing. In: ICDCS 2006: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, Washington, DC, USA, p. 72. IEEE Computer Society, Los Alamitos (2006)
10. Frank, C., Römer, K.: Solving generic role assignment exactly. In: IPDPS (2006)