Energy-Aware Scheduling of Distributed Systems

Pragati Agrawal, *Student Member, IEEE*, and Shrisha Rao, *Senior Member, IEEE*

***Abstract*—Scheduling of tasks on a multi-machine system to reduce the makespan, while satisfying the precedence constraints between the tasks, is known to be an NP-hard problem. We propose a new formulation and show that energy-aware scheduling is a generalization of the minimum makespan scheduling problem. Taking the system graph and program graph as inputs, we propose three different algorithms for energy-aware scheduling, each of them having its own strengths and limitations. The first is a genetic algorithm (Plain GA) that searches for an energy reducing schedule. The second (CA + GA) uses cellular automata (CA) to generate low energy schedules, while using a genetic algorithm (GA) to find good rules for the CA. The third (EAH) is a heuristic which gives preference to high-efficiency machines in allocation. We have tested our algorithms on well-known program graphs and compared our results with other state-of-the-art scheduling algorithms, which confirms the efficacy of our approach. Our work also gives insight into the time-energy trade-offs in scheduling.**

***Note to Practitioners*—In today's world of large systems and energy shortages, the need for energy efficiency in individual machines is complemented by the need for energy awareness in the use of the complete system. One important aspect of this is for proper scheduling of tasks to minimize the energy consumption in carrying out a program of tasks over a system of machines. Our work is to find an energy-aware schedule for a given system that also satisfies the precedence constraints between tasks to be performed by the system. We propose three different algorithms for energy-aware scheduling and indicate their strengths and limitations. We show that the energy-aware scheduling problem is a generalization of the minimum makespan scheduling problem. The assumptions made by us in this paper are close to practical settings, as we consider both the power required for task execution as well as the power dissipation of machines when idle. Our model is generic and can describe many distributed systems from different domains. We have validated our algorithms with simulations of systems with different numbers of machines, with some standard program graphs. Our results support the intuition that there can be energy-minimal schedules that are not time-minimal, and give scope for further work on energy-time trade-offs.**

***Index Terms*—Cellular automata (CA), distributed systems, energy-aware scheduling, genetic algorithms (GA), learning algorithms, machine learning, makespan.**

Manuscript received March 21, 2013; revised November 28, 2013; accepted February 25, 2014. Date of publication March 25, 2014; date of current version October 02, 2014. This paper was recommended for publication by Associate Editor R. Fierro and Editor S. Sarma upon evaluation of the reviewers' comments. A part of this work (only the CA + GA algorithm) was presented at the Sixth Annual IEEE International Systems Conference (IEEE SysCon 2012), Vancouver, BC, Canada, March 2012.

The authors are with the International Institute of Information Technology–Bangalore, Bangalore, Karnataka 560100, India (e-mail: pragati.a.in@ieee.org; shrao@ieee.org).

This paper has supplementary downloadable multimedia material available at http://ieeexplore.ieee.org provided by the authors. The Supplementary Material contains a video. The material is 267 MB in size.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2014.2308955

I. Introduction

TRADITIONAL thinking on the subject of scheduling has been concerned almost entirely with reducing the *time* taken to complete tasks. However, from the perspective of sustainability, it is perhaps even more fruitful to ask how to schedule tasks on a system of connected but dissimilar machines in such a way that the total *energy* consumed is kept low. The latter problem has yet to be addressed in as great detail as the former; though there are some efforts to build energy-aware algorithms and protocols in particular domains (especially, computer processors and networks), there is a need for a general theory of energy-efficient system scheduling that is not specific to a particular domain. Using systems in an energy-efficient way, and understanding the basic limits of energy-aware scheduling, still needs a lot of research. Efficiency can only be achieved by proper scheduling of required tasks over many machines, but even with the simplest case of a two-machine system, scheduling a parallel program is an NP-complete problem [2]. Hence, obtaining optimum schedules for systems with many machines is a challenging open problem.

Calculation of a near-optimal schedule when multiple machines or resources are present is a very important problem in operations research and computer science. This problem is most commonly referred to as job shop scheduling [3], [4]. Energy-aware scheduling can be seen as a variant of job shop scheduling with the optimization criterion being the minimization of total energy usage of the system, rather than its *makespan* (time to completion of a set of tasks). Its applications potentially include scheduling tasks in industrial process systems, scheduling cargo airline flights to reduce fuel consumption, and assigning elementary tasks in a multiprocessor system. In a world that is increasingly looking for green solutions for the future, increasing energy efficiency by intelligent allocation of tasks is most welcome. One major field which is waiting for energy-efficient scheduling solutions is distributed computing. According to a report [5], servers operate most of the time at between 10% and 50% of their maximum utilization levels. Hence, a good scheduling algorithm can increase the energy efficiency significantly.

Apart from the computing industry, scheduling is a widely known and well-studied problem in the chemical and other industries. Jobs in such industrial settings can be divided into elementary tasks. Tasks have to follow precedence constraints while there is scope for parallelization of some tasks. This gives an excellent opportunity to increase overall efficiency by proper scheduling. Munawar *et al.* [6] propose an algorithm for cyclic scheduling of continuous multi-product plants operating in a hybrid flow-shop. An integrated multilevel, control-theoretic framework has also been proposed for effectively handling integration of planning, scheduling, and rescheduling by Munawar

and Gudi [7]. A survey of scheduling algorithms used in chemical industry plants is presented by Zhou *et al.* [8].

It is well known that energy-aware scheduling is of practical interest in data centers [9]–[11], chip-level scheduling [12]–[14], and grid computing [15]–[18]. In the chemical process industry, techniques such as cycle-time analysis [19] are commonly applied, but energy-aware scheduling as such has not been applied to a significant degree, though energy consumption is a primary concern for the industry. Thus, the formulations introduced in this paper are useful additions to the set of methods [20], [21] already applied in chemical engineering.

Better scheduling can increase the overall efficiency and profitability of any process industry [22]. There are a lot of scheduling products available for scheduling in plants, such as the Model Enterprise Optimal Single-Site Scheduler [23], the Aspen Plant Scheduler [24], etc. These schedulers take a lot of things into consideration, like costs associated with the use of equipment, inventories, changeovers, labor, energy, utilities, etc., as objectives as well as constraints. Industries with chemical plants, fabrication units, and other process plants, however, have energy as one of the most significant inputs, which current scheduling products do not directly take into account, partly on account of the lack of a suitable theoretical framework. (A review of schedule optimization techniques used in any batch processing plant is given by Mendez *et al.* [25].) Hence, an algorithmic framework which can be used to assess schedules according to energy requirements and related constraints of the system, can be of great use to these process plants, and in related scheduling products.

A. Background

From the historical perspective, makespan scheduling is an important problem and lots of work has been done in its regard in various domains, using many different approaches. Many heuristic-based methods have been employed for scheduling (for example, list scheduling [26] and critical-path-based heuristics [27]), but most of them are sequential in nature. Some limitations of sequential scheduling algorithms are their sensitivity to scheduling parameters, lack of scalability, and determinism. Due to these limitations, they are generally not able to reach close enough to the optimum solution. Parallel scheduling methods have given a new outlook to this problem, but there is yet a lot of space for research and development [28].

Many stochastic global search techniques have been used in the field of parallel scheduling. These stochastic global search techniques combined with some heuristics have been successfully applied for makespan scheduling. Some of the successful heuristics include genetic algorithms (GAs) [29], [30], neural networks [31], simulated annealing [32], [33], and ant colony optimization [34], [35]. Though such methods have often produced good results, they have the drawback of requiring large scheduling overheads. Scheduling overhead is the computational cost on the system for finding a schedule dynamically. Since a new schedule has to be designed for any new program graph while the system remains the same, some properties remain common among different schedules. Generally, stochastic search algorithms do not try to take advantage of this fact and instead search for a schedule from scratch.

To reduce the scheduling overhead by taking advantage of this property, the use of cellular automata (CA) [36], [37] was proposed by Seredynski and Zomaya [38] to find schedules with reduced makespan. Their work uses genetic algorithms to learn the rules for CA, and proposes sequential as well as parallel update rules for an irregular CA. Swiecicka *et al.* [39] extend the method by adding an artificial immune system (AIS) technique to the method proposed by Seredynski and Zomaya [38]. Swiecicka *et al.* [39] use linear CA in place of the irregular automata of Seredynski and Zomaya [38]. Another method which uses irregular CA is proposed by Ghafarian *et al.* [40], who use ant colony optimization techniques to learn the rules of their CA. The methods tried to date which have used CA for scheduling have only concentrated on optimizing for total execution time.

Energy-aware scheduling of distributed systems is a relatively seldom explored area though one of much current interest. Liu *et al.* [41] present a method for power-aware scheduling under timing-constraints, but their system assumes only one particular machine for one class of tasks, and hence does not require parallel scheduling of tasks. Artigues *et al.* [42] apply tree searches for finding schedules under energy constraints in industrial applications. Scheduling overhead is usually not a concern in industrial scheduling problems, hence the method presented in their paper does not optimize for the scheduling overhead. Wang *et al.* [43] propose a method for energy-efficient scheduling of a uni-machine system under thermal constraints. A system could fail if its peak temperature exceeds its thermal constraints; higher temperatures may also lead to higher leakage power consumption. Their method performs thermal management to minimize the energy consumption in dynamic voltage/speed scaling (DVS) machines. Chan *et al.* [44] propose a technique for scheduling for weighted flow time and energy graphs, which includes the scope for rejecting some tasks for overall optimality. Lee and Zomaya [45] propose a framework in which many factors can be taken into consideration for energy-efficient operation of large-scale distributed systems.

A recent anthology edited by Zomaya and Lee [46] covers various issues in energy-efficient computing, from chip-level energy reduction approaches such as DVFS, to larger analyses applicable to data centers and smart grids. However, this book does not study energy-aware scheduling as such in much depth, and only basic algorithms like Greedy-Min and Greedy-Max (mentioned later) are considered on occasion.

B. Our Work

This paper formulates the energy-aware scheduling problem considering task dependencies and idle power consumption of machines, shows that the same is a generalization of the makespan scheduling problem, and presents three methods to find a good schedule which reduces the total energy consumption by a system, given the system configuration and the tasks to be completed by it.

The inputs to our algorithms are the *system graph*, the *program graph*, and the power specifications of the machines. The

system graph specifies the connections between the machines of the system. The machines of the system are the units which perform the tasks. The program graph specifies the execution times and precedence relationships of the tasks. The power consumptions of all machines while working as well as while idle are also to be specified as inputs to our algorithms. The aim of our algorithms is to assign each task to a resource such that all the tasks are completed with a low overall energy expenditure.

The first algorithm which we present (Plain GA) is a genetic algorithm, thus is basically a search method. Genetic algorithms have been used for makespan scheduling [47], [48].

In the second algorithm $(\mathrm{CA} + \mathrm{GA})$, we use CA for calculating energy-aware schedules. CA [36], [37] are collections of cells on a grid of specified shape that evolve through a number of discrete time steps according to a set of rules based on the states of neighboring cells. CA form highly parallel and distributed systems of single, locally interacting units which are able to produce a global behavior. CA can be used to emulate the properties of real-life systems. Hence, schedules for real-life systems can be found, with some scheduling overheads, using CA.

To solve the scheduling problem using CA, the system graph and program graph are first mapped to the CA domain. With some initial random schedules and rules, CA evolve to give better schedules. We use a GA to search for better rules for the CA that computes schedules, as has been done previously by others [38], [39]. As the generations in the GA progress, we keep improving on the schedules to have lesser energy consumption. Unlike previous works which also use GAs for finding rules of the CA or schedules, we also preserve good schedules found in previous generations, and improve upon them if possible to get better energy efficiency over successive generations.

The third algorithm (EAH) is based on a heuristic which tries to assign tasks to the most efficient machines available at the time of execution of the tasks. It is very fast but as it is based on a heuristic, it faces the same limitations as do other heuristic algorithms elsewhere.

We have tested our algorithms on program graphs which are most commonly used by researchers in this field for testing their algorithms. We have formulated the energy-aware scheduling problem in a very generic manner. Our results give insights into the time-energy trade-offs found in many systems—such as how, as we increase the number of machines, the energy consumption may also increase while the makespan may decrease. The insights given by our results can enable system designers or users to choose the appropriate numbers of resources they want in their systems, or the correct schedules, depending on the desired balance between makespan and energy.

Thus, the salient features of our work are as follows.

1) The system model used here is very close to the behavior of practical systems, as we have considered both the power required for task execution as well as the power dissipation of machines when idle, which is not the case in prior works on energy-aware scheduling. The power consumptions of various machines are considered to be different and independent of other machines.
2) Our approach is generic as it can be applied on various types of systems, from large car manufacturing or chemical processing units to distributed computing systems.
3) Most works in scheduling deal with very few (2–4) machines. We have shown simulation experiments with eight machines and our approach is further extensible to even larger numbers of machines.
4) The system graph used in our simulations is a mesh connected graph which is common across almost all the contemporary and prior works in this field, but our method has the capability to deal with any general system graph which may or may not be mesh connected.
5) We present three algorithms for energy-aware computations of schedules. We compare the strengths, limitations, and computation costs of each algorithm. Any one of these algorithms can be used in a particular domain of application, depending upon the requirements of the same.

A formal definition of the problem considered in this paper is presented in Section II. The proposed methods are explained in Section III. Section IV illustrates the results observed by simulating the proposed method on standard program graphs. Finally, Section V describes the conclusion and scope for future work.

II. Energy-Aware Scheduling

The scheduling problem is generally modeled using a system graph and a program graph. The input to our algorithm are the system graph, the program graph, and the power specifications of the machines in the system.

The basic aim of our algorithm is to assign each task to a machine such that all the set of all tasks is completed with minimum possible energy expenditure; this is calculated by the *fitness function* which is a mapping from each schedule to the energy consumed by it.

The next section formally introduces the model used for scheduling in this paper.

A. System Model

Our model comprises of a system graph, a program graph, and machine specifications.

System Graph: A system is represented by an undirected unweighted graph, called the system graph. Here, $\mathcal{V}_c$ is the set of nodes of the system graph representing machines with their local memories. The cardinality $|\mathcal{V}_c| = N_c$ specifies the number of machines in the system. Edges represent channels between machines and defines a topology of the multi-machine system. In our simulations, we assumed the topology to be a fully connected mesh topology with $N_c = 2, 4, 8$.

Fig. 1 shows an example of a system graph with four nodes, with $\mathcal{V}_c = \{C_1, C_2, C_3, C_4\}$ as machines which are connected in a mesh topology with bidirectional links. The machines of the system are the actual units which perform tasks. The system graph specifies the connections between the machines of the system. It is assumed in our work that communication links themselves do not consume any power.

Program Graph: A parallel program is represented by a weighted directed acyclic graph $\mathcal{G}_p = (\mathcal{V}_p, \mathcal{E}_p)$, called a precedence task graph or a program graph. In the program graph:

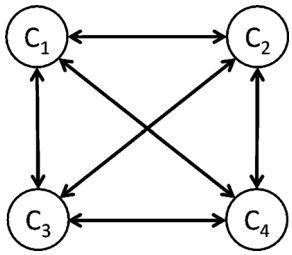

Fig. 1. System graph example: Mesh topology with four nodes.

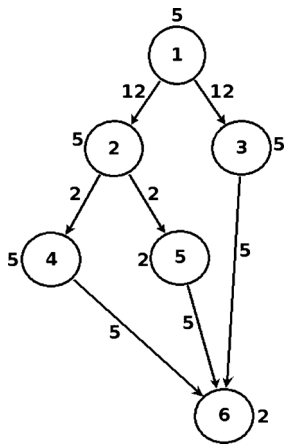

Fig. 2. Program graph example: Weights of nodes and edges are given by numbers along them. The precedence constraints are specified by the directions of the edges.

TABLE I
TERMS DEFINED FOR PROGRAM GRAPH

Symbol	Meaning
i	task i in program graph
$w(i)$	weight of node i
$l(i,j)$	weight of edge connecting i and j
$C(i)$	machine that is assigned task i

- $\mathcal{V}_p$ is the set of nodes of the graph where each node represents an elementary task. The cardinality $|\mathcal{V}_p| = N_p$ specifies the number of elementary tasks in the program.
- $\mathcal{E}_p$ is the set of edges which specifies the precedence between the tasks.

The program graph specifies the execution times and precedence relationships of the elementary tasks.

The weight of a node represents the execution cost of the corresponding task, and the weight on an edge shows the transfer cost between two tasks if they are located at different machines. If they are located on the same machine then the transfer cost is taken as zero. The weight $w(i)$ of node i describes the processing time needed to complete task i in a program graph. The weight $l(i,j)$ of an edge (i,j) describes the communication time between the pair of tasks i and j. If i and j are on then same machine then it is 0. $C(i)$ denotes the machine that is assigned task i in a given schedule. Fig. 2 shows a small example of a program graph with weights of nodes and edges.

Table I describes various parameters of a program graph.

TABLE II
POWER CONSUMPTION SPECIFICATIONS OF THE SYSTEM

Symbol	Meaning
$\mu(C_u)$	Power consumption in working state by machine C_u
$k\mu(C_u)$	Power consumption in idle state by machine C_u, here $0 \leq k \leq 1$
$\tau_c(C_u)$	Time taken in working state by machine C_u
$\tau_i(C_u)$	Time taken in idle state by machine C_u

Power Specifications: The power consumptions of each machine in working as well as idle states are to be specified as inputs. The power consumption in the working state of machine C_u is denoted by $\mu(C_u)$ and the power consumption in the idle state of machine C_u is given by $k\mu(C_u)$, where $0 \leq k \leq 1$. If $k = 0$ it means the machine is effectively switched off when not under load, and thus does not consume any power when idle. If $k = 1$ it means that in the idle state with no load also the machine consumes 100% of its full-load power. $\tau_c(C_u)$ is the time spent in the working state by machine C_u, and the time spent in the idle state by machine C_u is denoted by $\tau_i(C_u)$.

The total energy consumption of the system of all machines is denoted by E and the total time taken by the system to complete a set of tasks is denoted by T.

Fitness Function: The aim of scheduling is to assign each node of the program graph (i.e., a task) to a node in the system graph (i.e., to a machine). In this paper, we describe approaches to find schedules which reduce the total energy consumption of the system, given the power specifications (power requirements) of each machine of the system, the system graph $\mathcal{G}_c$, and the program graph $\mathcal{G}_p$.

Suppose the power consumption specifications of the system are as indicated in Table II.

The total energy consumption of the system is given by

$$E = \sum_{n=1}^{N_c} [\mu(C_n)\tau_c(C_n) + k\mu(C_n)\tau_i(C_n)]. \quad (1)$$

Hence, our aim is to find a schedule which minimizes the total energy E as given by (1). As previously discussed, the classical scheduling problem which is similar to energy-aware scheduling is minimum makespan scheduling, where T rather than E is minimized. In the following section, we describe how the makespan scheduling problem is related to energy-aware scheduling.

B. Minimum Makespan Scheduling Versus Energy-Aware Scheduling

Makespan scheduling aims to minimize the total time in which all tasks are finished. It has been studied extensively by many researchers for several decades. If the power specifications of all machines in a system are the same, then the minimum-energy problem can be reduced to that of calculating the minimum total execution time for a given program of tasks, since the minimum time then corresponds to minimum energy. The proof of this claim is presented next.

Considering the power consumptions of all the machines to be the same, i.e., $\mu(C_1) = \mu(C_2) = \mu(C_3) = \cdots = \mu(C_{N_c}) = \mu(C)$, (1) becomes

$$E' = \sum_{n=1}^{N_c} [\mu(C)\tau_c(C_n) + k\mu(C)\tau_i(C_n)] \quad (2)$$

$$= \mu(C) \sum_{n=1}^{N_c} [\tau_c(C_n) + k\tau_i(C_n)]. \quad (3)$$

Now, let the total time be T, where

$$T = \tau_c(C_n) + \tau_i(C_n). \quad (4)$$

Substituting $\tau_i(C_n) = T - \tau_c(C_n)$ in (2), we get

$$E' = \mu(C) \sum_{n=1}^{N_c} [\tau_c(C_n) + k[T - \tau_c(C_n)]] \quad (5)$$

$$= \mu(C) \sum_{n=1}^{N_c} [(1-k)\tau_c(C_n) + kT] \quad (6)$$

$$= \mu(C) \left[(1-k) \sum_{n=1}^{N_c} \tau_c(C_n) + N_c kT \right]. \quad (7)$$

The term $\sum_{n=1}^{N_c} \tau_c(C_n)$ signifies the sum of the times taken for all tasks. Since the number of tasks and the weights associated with them are constant, their sum is also constant. Hence, we replace $\sum_{n=1}^{N_c} \tau_c(C_n)$ in (4) by a constant η

$$E' = \mu(C)[(1-k)\eta + N_c kT] \quad (8)$$

$$= \mu(C)(1-k)\eta + \mu(C)N_c kT. \quad (9)$$

The first term in (9) is constant, and in the second term all variables except T are fixed. Hence, (9) can only be optimized over the variable T which signifies the total makespan of the schedule, reducing the energy-aware scheduling problem to the makespan scheduling problem.

Therefore, if the power specifications of all system machines are the same, then the minimum-energy problem reduces to that of calculating the minimum total execution time for a given program of tasks, since the minimum time corresponds to minimum power. Our proposed techniques consider the general case where machines have different energy requirements, hence provide a solution with energy awareness. The next section explains our proposed methods in detail.

III. Proposed Methods for Energy-Aware Scheduling

We propose three methods for energy-aware scheduling. Each of these has its own strengths. In this section, we explain these methods in detail. We first explain the genetic algorithm method.

A. Genetic Algorithm for Energy-Aware Scheduling (Plain GA)

A GA [49]–[51] is a heuristic for search, mimicking the natural evolution process. In a GA, a population of strings encoding candidate solutions evolves towards better solutions, using mutation and crossover functions. The evolution usually starts from a random population of individuals and happens over multiple generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations of candidate solutions is completed, or a satisfactory fitness level is reached for the population.

GAs have been used for makespan scheduling with good success [47], [48]. We present a method which instead uses this power of GAs to find energy-reducing schedules.

We first map the energy-aware scheduling problem to the GA domain. Since a GA finds the strings which have good fitness values, we model schedules as strings and the energy values of the strings by the fitness function. Hence, a string in GA is a vector of length equal to the number of nodes in the program graph. Each element in that vector represents the machine corresponding to that node. The parents for mutation and crossover are chosen on the basis of the energies of the schedules in the current generation. A schedule is more likely to be chosen as a parent if its rank (in terms of lower energy consumption) is better than others. The mutation rule uses a Gaussian distribution with zero mean and a variance, which reduces the number of generations. The crossover is random, in the sense that it arbitrarily chooses the portions from the first and second parent. The search space of a GA depends upon the number of machines as well as the number of elementary tasks in the program graph. The total number of possible solutions in a GA-based method is ${N_p}^{N_c}$.

In the next section, we explain our second proposed algorithm which uses a cellular automaton and a GA to find energy-aware schedules.

B. Cellular Automata Method for Energy-Aware Scheduling ($CA + GA$)

We use the framework of CA to solve the stated scheduling problem. But to solve the scheduling problem using CA, the system graph and program graph have to be first mapped to the CA domain. An elementary task of the program is mapped to a cell in the CA space. The state of the cell specifies the machine to which the task is assigned. Initially, a random assignment of tasks is created. Then, according to the rules and the neighborhood, the CA evolves successively to reach a state which gives a better schedule. We use a GA to find the CA rules which, in turn, generate better schedules. In the process of finding better rules, we also keep improving upon the schedules across GA generations.

The detailed architecture of this method is explained in Section III-B1. The algorithm for energy-aware scheduling is described in Section III-B2. The selection of the neighborhood and the rules are critical to finding a good solution with minimal scheduling overhead. Section III-B3 describes the cellular automaton algorithm and the method for selecting the

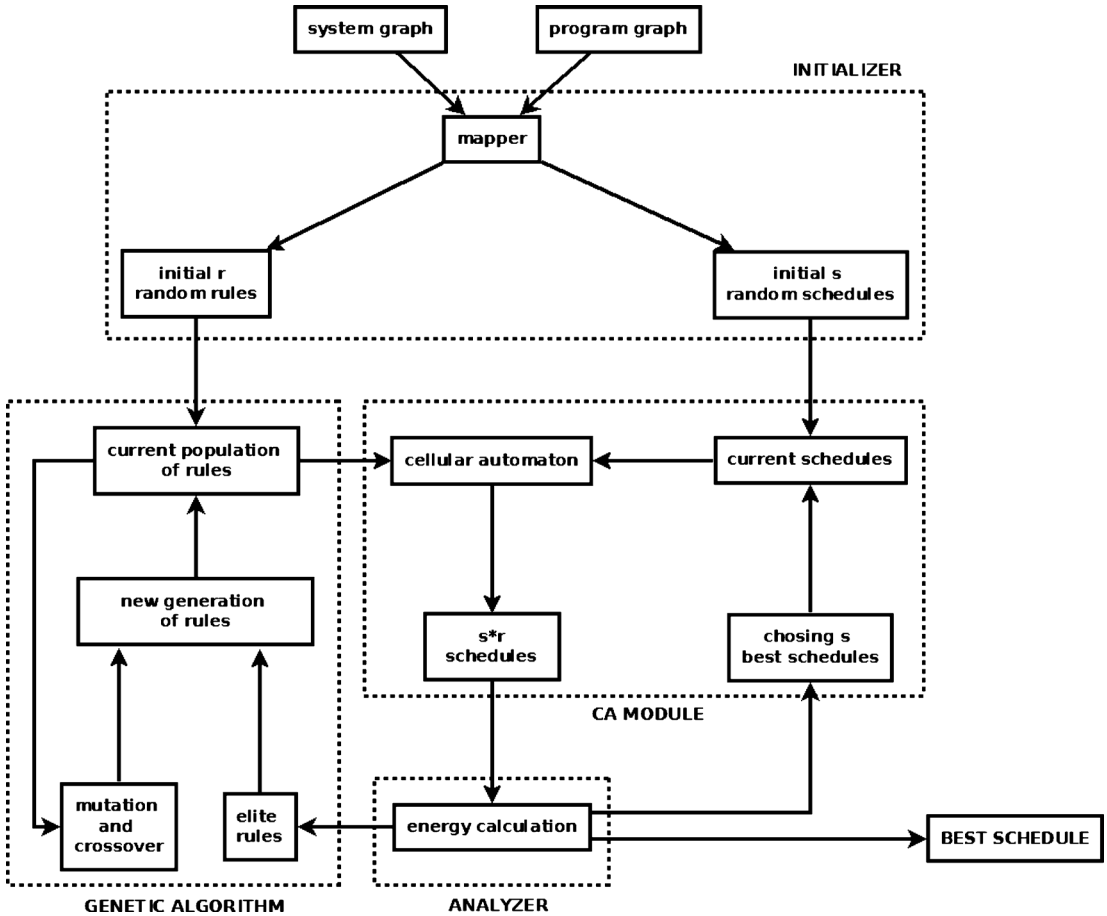

Fig. 3. Schematic for the CA + GA scheduling method.

neighborhood. The learning of rules and the finding of a good schedule are explained in Section III-B4.

1) Architecture: There are four major building blocks in our architecture. The block diagram in Fig. 3 presents the architecture of our method. The four blocks named initializer, CA module, genetic algorithm, and analyzer, represented by dotted lines, are as described next.

- INITIALIZER: This takes the program graph and the system graph as inputs. A MAPPER in the INITIALIZER maps the system graph with the program graph, which enables the INITIALIZER to generate s initial random schedules and also r initial random rules for the CA MODULE. When the algorithm starts for the first time, these schedules and rules are used to initialize the current set of schedules and the current set of rules, respectively.
- CA MODULE: The inputs to the CA MODULE are schedules and rules, which are taken from the current schedules and current population of rules, respectively. The CA algorithm evolves upon the current schedules using the current set of rules. There are s schedules, each of which is acted on by r rules, so there are $s \times r$ output schedules. (For details of how the cellular automaton evolves, refer Section III-B3.)
- GENETIC ALGORITHM: The work of the GA is to search for good rules in the space of all rules. It takes rules and energy values of schedules as inputs, and based on these energy values it selects elite rules and applies mutation and crossover functions to the rest of the rules and produces a better generation of rules. For the next iteration, these elite rules and the rules generated by GA are used to populate the current set of rules. Section III-B4 explains the further details of our GA.
- ANALYZER: The output of the CA module, i.e., the set of $s \times r$ schedules, is taken as input by the ANALYZER, and for each schedule the energy is calculated, and schedules are sorted by their energy values. The best s schedules are given to the CA module for the next iteration, and the best r rules are given to the genetic algorithm. Also, the best schedule (σ^*) is taken as output from this block when the stopping criterion for the GA is reached.

2) Algorithm: Notation used in Algorithm 1:

- s is the total number of initial schedules;
- r is the total number of initial rules;
- σ_m is the mth schedule;
- ρ_n is the nth rule;
- $\xi()$ is the energy function for schedules;
- $\xi(\sigma_m)$ is the energy consumed by the mth schedule;
- ν is the current generation;
- g is maximum number of generations;
- $\varsigma_{m,n}$ is a schedule got after applying CA to the mth schedule with the nth rule;
- σ^* is the schedule with minimum energy;

Algorithm 1: The CA+GA Algorithm

input : Initial schedules (σ_m), rules (ρ_n)
output: Best schedule (σ^*)

$\rho_n \leftarrow \rho_n, 1 \leq n \leq r$;
$\sigma_m \leftarrow \sigma_m, 1 \leq m \leq s$;
$\xi(\sigma_m) \leftarrow 0$;
heapsort($\xi(\sigma_m)$);
$\xi(\sigma_q) \leftarrow min(\xi(\sigma_m))$;
$\sigma^* \leftarrow \sigma_q$;
$\nu \leftarrow 1$;
for $n = 1$ *to* r **do**
  for $m = 1$ *to* s **do**
    $\varsigma_{m,n} \leftarrow$ CA(ρ_n, σ_m);
    $\xi(\varsigma_{m,n})$;
  end
end
heapsort($\xi(\varsigma_{m,n})$);
$\sigma_m \leftarrow \varsigma_{m,n}, 1 \leq m \leq s$;
if *($\nu = g$) || ($min(\xi(\varsigma_{m,n}))$ has not decreased d consecutive times)* **then**
  heapsort($\xi(\sigma_m)$);
  $\xi(\sigma^{g*}) \leftarrow min(\xi(\sigma_m))$;
  if $\xi(\sigma^{g*}) < \xi(\sigma^*)$ **then**
    $\sigma^* \leftarrow \sigma^{g*}$;
  end
else
  $\varrho_n \leftarrow$GA(ρ_n);
  $\nu \leftarrow \nu + 1$;
  $\rho_n \leftarrow \varrho_n, 1 \leq n \leq r$;
  goto step 6;
end

- ϱ_n is the updated rule after applying GA to rule ρ_n;

Algorithm 1 indicates the $\mathrm{CA} + \mathrm{GA}$ algorithm, and may be read in correlation with our architecture described previously.

3) Cellular Automaton Module: The architecture of the CA used in the proposed method is linear and irregular, which means that the neighborhood of a cell need not necessarily consist of the geometric neighbors of the cell. This creates the scope to choose neighbors that are more relevant according to the program graph. Our proposed method uses a neighborhood of size 4, which includes two parents (nodes in a program graph) and two children (nodes in a program graph) of the task. If a node i has more than two parents (respectively, children) then 2 parents (respectively, children) are selected from all available parents of the node. If a node has less than 2 parents (respectively, children) then dummy nodes are assigned to it as neighbors. The process of assigning nodes as neighbors is explained next.

1) Two dummy parent nodes with state -1 are assigned to the entry nodes (i.e., nodes with no parent node).
2) Two dummy children nodes with state -1 are assigned to the exit nodes (i.e., nodes with no child node).
3) If a node has only one parent then a dummy parent node is added with the same state as the non-dummy parent node.
4) If a node has only one parent then a dummy child node is added with the same state as the non-dummy child node.
5) If a node i has more than two parents (respectively, children) then the two parents (respectively, children) connected with highest edge weights ($l(j, i)$) to the node i are selected as the parents (respectively, children).

We choose the parents (respectively, the children) with highest edge weights in the CA neighborhood, since these edges affect the objective function more than others. Once the scheduling problem is mapped to the CA domain, we proceed to find good schedules. Such good schedules are given by the evolved state of the automaton, which is obtained by applying elite rules to good initial schedules. Since both the rules and the schedules are crucial, we search for both of these. We have designed a procedure in which we keep improving the state of the CA as well as the rules simultaneously to find the near-optimum schedule. We use a GA to search for the good rules. The process is explained next.

4) Genetic Algorithm for Rule Selection: We use a GA to search for good rules and in the process of doing so we also find the near-optimal schedule for our scheduling problem. The application of GAs to finding CA rules was first discussed by Das *et al.* [52]. As the aim is to search for good CA update rules, they are treated as individuals, creatures, or phenotypes in our GA setting. The fitness of a rule is given by the energy efficiency of the schedules obtained by applying the rule to some initial states of automata. Since rules can be represented as number strings, the reproduction is carried out by mutation and crossover of these strings.

The parents for mutation and crossover are chosen on the basis of average energies of the rules in the current generation. A rule is more likely to be chosen as a parent if its rank (in terms of lesser energy consumption) is better than others. The mutation rule uses a Gaussian distribution with zero mean and a variance which reduces over generations. The crossover is also random in the sense that it arbitrarily chooses the portions from first and second parents.

Next, we explain the heuristic-based approach to energy-aware scheduling.

C. Efficiency-Based Allocation Heuristic (EAH)

The two methods we have until now are search algorithms. Search algorithms just try to find the best possible solution in a given solution space. The other types of algorithms generally used for scheduling are heuristic-based algorithms. Such algorithms have some heuristic at their core. The heuristic proceeds towards locally optimal results and it is hoped that with the combinations of locally optimal decisions, a globally optimal or at least a good result can be obtained. Heuristic-based algorithms are very fast and have steady performances, but generally reach some subpar solution. Search-based algorithms are slower but on the other hand do not get stuck in local minima, though they do not guarantee a good solution every time they are executed. Their success can only be measured statistically.

The first-in–first-out (FIFO) heuristic is used for scheduling in many single-machine as well as multi-machine systems. We extend the same idea by also incorporating knowledge of differing power requirements of different machines. As soon as a task is ready to be executed, we check which machines have an empty input queue. From the set of free machines, we choose the one with lowest working power, and we assign the task to that machine. If no machine is free, then we assign the task to the machine with least working power.

We now present the simulation experiments and their results which substantiate the usefulness of our proposed methods.

IV. Results

A number of simulations with standard program graphs have been conducted. These graphs are tree15, g18, gauss18, and g40 which are also used extensively in the literature (see, e.g., [38]).

Our algorithms as well as the simulation setup allow for any number of machines, which can have different working state power consumptions and idle state power consumptions, to be used in the simulated systems. However, most existing scheduling algorithms consider fewer than 8-machine systems. The standard graphs which we have used are tested with 8-machine or smaller systems in previous published work. Hence, for illustrative purposes, we have shown results for 2-, 4-, and 8-machine systems, to make it easier to compare our algorithms with others. Similar results also obtain with other numbers of machines.

As has been previously discussed, prior state-of-the-art scheduling algorithms compute good schedules for minimizing makespan rather than energy. Though our algorithms are energy-aware and work to reduce the energy rather than time, if we take the working power and idle power as identical then they provide the minimum makespan schedules as well. Hence, for the sake of comparison with other systems, we have also calculated the schedules for makespan as well.

In the simulations reported in this section for the CA + GA algorithm (Algorithm 1), we assume that the cellular automaton works asynchronously [38]. This means at a given instant of time, only one cell updates its state. So, a single step of the CA, i.e., a rule to be applied once on all cells, takes on the order of the number of tasks to be completed. In the simulation, for learning the rules we fixed the population size of the GA at 20 and the maximum number of generations at 100. These tuning parameters are kept the same for all approaches presented.

For the GA scheduling algorithm also, the GA parameters remain the same.

The schedules obtained from the CA + GA and GA algorithms just tell us which task will be performed on which machine, but not the sequence of task execution. We have used a FIFO approach for scheduling of tasks assigned to the same machine, i.e., the task which reaches the machine earlier is executed first.

In the following section, we first explain the different program graphs which we have used. We then analyze different aspects of our proposed methods based on the simulation results. After this we present the comparison of our methods with one another and also with other published methods. In the end, we discuss trade-offs and indicate the advantages of our proposed algorithms based on experiments.

A. Program Graphs

We have used four standard program graphs for simulation experiments: tree15, g18, gauss18, and g40. Tree15 is a binary tree with 15 nodes. All the working costs and communication costs in this program graph are the same, and can be taken to be unity. The program graph for graph tree15 is shown in Fig. 4.

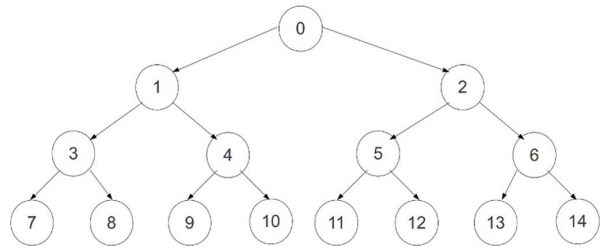

Fig. 4. Program graph tree15.

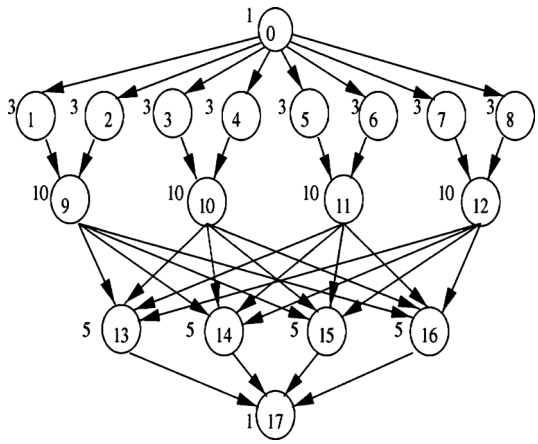

Fig. 5. Program graph g18.

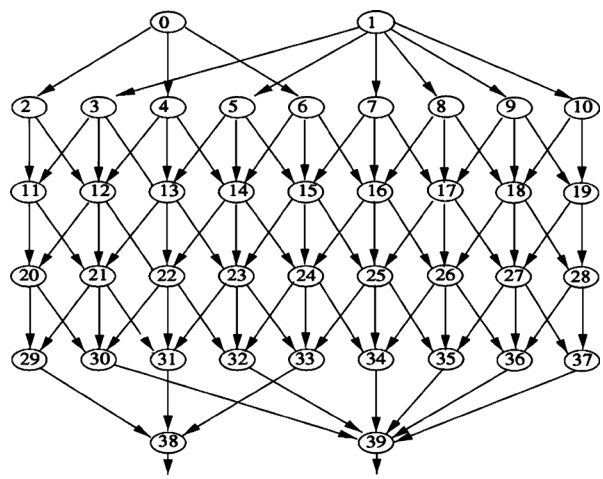

Fig. 6. Program graph g40.

The program graph g18 is displayed in Fig. 5. It has 18 tasks with different computation costs as shown, and the communication cost for all links equal unity.

Fig. 6 displays the program graph g40 which is also considered in simulation. This is a directed acyclic graph which has 40 nodes. The computation and communication costs of tasks are equal to 4 and 1, respectively.

Fig. 7 displays the program graph gauss18. This is a directed acyclic graph with 18 nodes. The computation and communication costs of tasks are as indicated in the figure.

B. System Analysis: Effects of Varying k and C

In this section, we present our analyses of our experiments in which we have simulated the program graphs on different

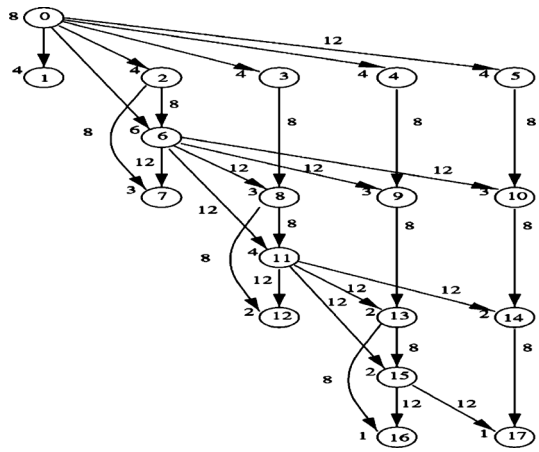

Fig. 7. Program graph gauss18.

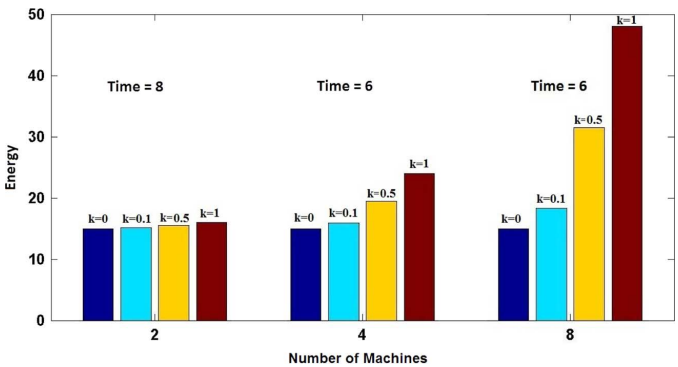

Fig. 8. Comparison of energy values with different idle times for 2, 4, and 8 number of machines for program graph tree15.

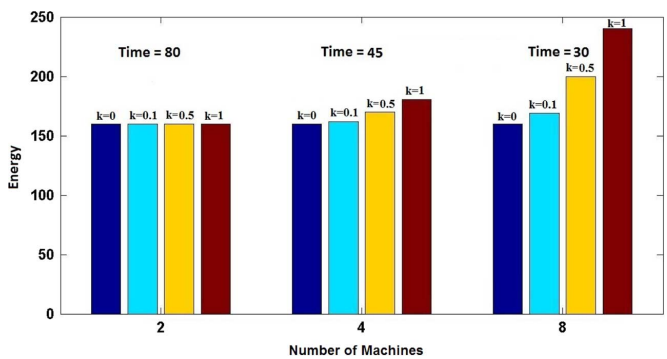

Fig. 9. Comparison of energy values with different idle times for 2, 4, and 8 number of machines for program graph g40.

system graphs. These system graphs have different numbers of machines ($N_c = 2, 4, 8$) and different idle power to working power ratios ($k = 0, 0.1, 0.5, 1$). The energy and makespan values shown in this section are the best ones we have got from our energy-aware algorithms. The focus of the analyses is how k and N_c are related irrespective of the scheduling algorithm used. The comparison of energy values with different values of k for 2, 4, and 8 machines for program graphs tree15 and g40 is shown in Figs. 8 and 9, respectively.

We see that with an increase in the number of machines, the energy required also increases if tasks are less—because in that case most of the machines remain idle and the idle power consumption contributes to increase in the total energy E, though the makespan time is decreased because tasks are executed in parallel on different machines.

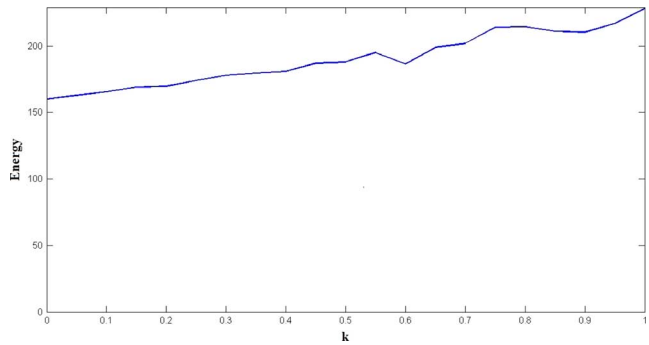

Fig. 10. The effect of varying k on energy consumption for graph g40.

The effect of any increase in idle power consumption of machines on the net system energy consumption is pretty straightforward. As the idle power consumption of machines increases, the energy consumption of the system increases nearly linearly for any system with a larger number of machines than called for by the parallelism of the program graph. Fig. 10 shows the variation in energy consumption with variation in k for a 4-machine system on $g40$ graph.

Thus, we see that as the number of machines increases, the idle power specification plays an important role in deciding the energy consumption of the schedule.

C. Comparative Performance of Algorithms

As there is relatively less literature on energy-aware scheduling *per se*, we first compare our scheduling algorithms with other algorithms which optimize on makespan. As previously noted, the energy-aware scheduling becomes makespan scheduling if all machines have the same power specification.

Using a 2-machine system, we show the best makespan calculated by our proposed as well as other algorithms, along with the time taken to calculate the schedules in Table III; similar relations are seen for other numbers of machines as well. Each entry in this table has two elements separated by a comma. The first element indicates the makespan time and the second represents the scheduling overhead time of the algorithm in seconds.

We used an Intel i5 processor computer for these simulations. In Tables III–V, EAH refers to the Efficiency-Based Allocation Heuristic explained in Section III-C. In the same table, GA refers to the GA-based energy-aware scheduling algorithm explained in Section III-A. The Greedy-Min algorithm is a standard algorithm [46] which tries to first finish the jobs which can be finished sooner (i.e., it first executes the jobs with lower node weights), while honoring the program graph dependencies. In the Greedy-Max algorithm [46] the preference is to finish those jobs first which take longer times to finish (i.e., the jobs with higher node weights are the first to be executed) while adhering to the program graph dependencies.

Clearly, the CA + GA algorithm gives the best results for makespan scheduling. We have not mentioned the results of other state-of-the-art algorithms in Table III since the makespan

TABLE III
MAKESPAN COMPARISON

Graphs	CA+GA	Plain GA	EAH	Greedy-Min	Greedy-Max
tree15	9, 3.76	9, 0.32	12, 0.002	12, 0.015	12, 0.01
g18	46, 4.97	46, 0.43	58, 0.005	76, 0.05	76, 0.04
gauss18	44, 6.53	44, 0.66	102, 0.005	74, 0.035	74, 0.03
g40	80, 20.26	80, 0.96	112, 0.01	89, 0.197	89, 0.11

TABLE IV
ENERGY COMPARISON, WITH C = 4

Graphs	CA+GA	Plain GA	EAH	Greedy-Min	Greedy-Max
tree15	36.4, 5.02	36.8, 0.428	38.8, 0.015	41.6, 0.063	41.6, 0.028
g18	201.2, 7.58	201.6, 0.31	223.2, 0.004	311.6, 0.048	311.6, 0.038
gauss18	163.2, 8.735	193.6, 0.359	291.2, 0.003	280.8, 0.029	280.8, 0.029
g40	356, 19.5	374.4, 1.007	406.4, 0.009	436.8, 0.134	436.8, 0.14

TABLE V
ENERGY COMPARISON, WITH C = 8

Graphs	CA+GA	Plain GA	EAH	Greedy-Min	Greedy-Max
tree15	76.8, 87.76	84, 0.71	76.8, 0.013	104, 0.048	104, 0.028
g18	380.8, 96.33	412.8, 0.36	380.8, 0.006	652.8, 0.102	652.8, 0.07
gauss18	454.4, 62.11	462.4, 1.23	834.4, 0.01	820.8, 0.086	820.8, 0.104
g40	687.2, 204.2	761.6, 0.8288	786.4, 0.008	931.2, 0.372	931.2, 0.398

values provided by them are the best possible for these graphs, and we have also got the same makespans. This indicates that our algorithms perform as well as other good published algorithms even for makespan scheduling.

Tables IV and V show comparisons of the energy requirements of all the algorithms considered, with Table IV being for a 4-machine system, and Table V for an 8-machine system.

As suggested by the relative dearth of learning algorithms in the field of scheduling, it is not straightforward to apply learning algorithms to scheduling. Learning algorithms can learn to derive good schedules; but such learning, and the schedules obtained, are very specific to particular program graphs. It is generally better to store the good schedules for each program graph rather than trying to learn them, so learning schedules is generally not a good option. With CA+GA we do not learn schedules, but use the GA to learn CA rules that generate good schedules. These CA rules are specific to the system graph, but not to the program graph (the same rules can be used to generate schedules for different program graphs on a given system); i.e., they can in a sense capture the properties of the system graph.

Among our proposed algorithms, CA + GA and Plain GA are search algorithms, hence can give different outputs on different runs, while EAH (like the published Greedy-Min and Greedy-Max) is a heuristic-based algorithm which gives the same output each time. The output of search algorithms is often better than heuristic-based algorithms—simple heuristics are not good enough to find the best schedules, as they mostly come to some significantly suboptimal results. Hence, search algorithms are necessary to find better results. Clearly, as can be seen in the results, the Plain GA algorithm is as good as the CA+GA algorithm, and it is much faster. But there are certain advantages of the CA + GA algorithm over others.

- **Better Convergence**: In our experiments we have seen that CA + GA gives very good results much more frequently than the Plain GA.
- **Fixed Search Space**: The search space of the CA + GA algorithm only depends on the number of machines and the size of the neighborhood of the CA. On the contrary, the search space of GA is exponential on the number of nodes of the program graph. The bigger the search space, the lower the probability of reaching close to the global optimum.
- **Learning**: The biggest motivation behind using CA is its ability to learn the characteristics of the system graph. A rule learned for one program graph on a given system can be applied successfully to other program graphs on the same system.

TABLE VI
LEARNING SIMULATION

Graphs	tree15	g18	gauss18	g40
tree15	34.95	32.19	52.78	28.63
g18	33.47	31.98	50.55	29.03
gauss18	35.42	30.52	49.02	27.61
g40	29.43	22.72	37.32	28.03

We conducted experiments which support this claim of learning ability of CA. We obtained a CA rule learned for a given system and program graph, which we then applied on random schedules of other program graphs to get the evolved schedules. We noted that the evolved schedules were on an average much better than the initial random schedules. The percentage improvement for all the graphs is reported in Table VI. The percentage improvement is calculated as

$$\text{impr} = \frac{1}{I}\sum_{i=1}^{I}\frac{E_i(\text{initial}) - E_i(\text{evolved})}{E_i(\text{initial})} \times 100. \qquad (10)$$

Here, I is the number of initial random schedules which we have taken as 200 in our experiments. The first column of Table VI indicates the program graph on which the rule

was learned. Each row records the percentage improvement in the schedule computed using a rule learned with the program graph mentioned in the first column of that row, over the initial (random) schedule. We consider a 4-machine system with power consumptions 1, 2, 3, and 4 respectively, with k taken as 0.2.

As noted previously, a CA rule learned with a program graph can be gainfully applied to another program graph. For instance, we see from Table VI that the rule learned using program graph g18, when applied to program graph gauss18, gives a 50.55% improvement.

The learning capability of CA indicates that the algorithm using $\mathrm{CA} + \mathrm{GA}$ is not just a random global search for better rules, but constitutes searching in a reasonable manner. Hence the $\mathrm{CA} + \mathrm{GA}$ algorithm has the power of learning which helps it converge to better solutions, while remaining a search algorithm, which helps it to avoid getting stuck in locally good results.

Hence, all the proposed algorithms have certain advantages over others, and we may sum up our recommendations based upon their properties.

- **EAH**: It is a very simple and fast algorithm and needs little memory and computational power compared with other approaches. It is to be preferred if getting the best possible schedule is not a high priority but the schedule is to be calculated fast (or at runtime), or if the system calculating the schedule has limited processing power.
- **Plain GA**: It is a fast search algorithm. It is to be preferred if sufficient processing power is available and good schedules, though not necessarily the best ones, are to be calculated quickly.
- $\mathbf{CA + GA}$: It gives the best schedules, and is to be preferred if the scheduling overhead can be overlooked (as in industrial settings) in order to obtain schedules that are likely to be the best obtainable.

Consequently, a user or designer can choose from any of the proposed algorithms based on the requirements. We find the $\mathrm{CA} + \mathrm{GA}$ algorithm to give the best schedules amongst the proposed algorithms. The next section presents some more insights in the $\mathrm{CA} + \mathrm{GA}$ algorithm using some simulation data.

D. Insights Into the $CA + GA$ Algorithm

The proposed $\mathrm{CA} + \mathrm{GA}$ algorithm is quite complex. Looking at some graphs giving simulation results can help us understand more about the algorithm. We have analyzed the change in the fitness function for GA across different generations. The fitness function of a rule is the average of energies of the output schedules which are evolved using that rule. The plot for graph tree15 is shown in Fig. 11, which show the best, worst and mean values of the fitness function (energy function) of different rules across the generations. Similar graphs for g18 and g40 are shown in Figs. 12 and 13. In case of g18 and g40, the mean values of fitness function are decreasing with generations, which shows that better rules have been found as the generations are progressing.

On the contrary, in Fig. 11, it is seen that the minimum values of fitness function are better in earlier generations rather than later ones. This can be explained from the fact that the input schedules on which the rules are tested are different in different generations, hence, even if the elite rules of previous generations are carried forward to later generations, their fitness values may have increased. Though this gives us an indication that we might not be improving upon the rules over generations, it does not adversely affect the final result. The reason is that we store the best schedules from across generations and give the one with lowest energy as the output schedule, regardless of the generation in which that schedule was evolved. In the above mentioned simulation results, we have chosen the value of k to be 0.1, which means that the idle power of the machines is taken as 0.1 times their working power.

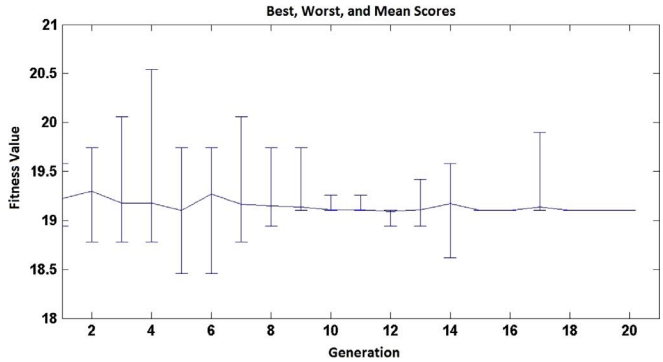

Fig. 11. Plot for program graph tree15 showing the best, worst, and mean values of the fitness function across generations.

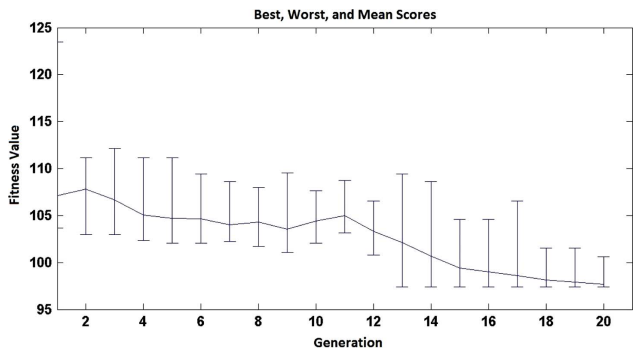

Fig. 12. Plot for program graph g18 showing the best, worst, and mean values of the fitness function across generations.

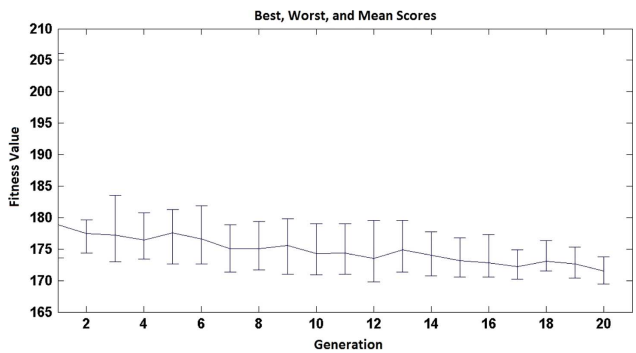

Fig. 13. Plot for program graph g40 showing the best, worst, and mean values of the fitness function across generations.

V. Conclusion

In this paper, we have presented three algorithms for energy-aware scheduling of tasks on a system with multiple,

non-identical machines that have non-zero idle power requirements. Our approach is quite generic, in the sense that it can be used to model and analyze many systems in various domains. Its main strength lies in the facts that it can deal with systems with various numbers of machines, and that it generates a schedule which is according to the power specifications of the machines in that system. All three algorithms give results comparable to or better than present state-of-the-art methods.

The cellular-automata-based algorithm which we call CA + GA is seen to be the best of the three proposed algorithms though it has the maximum scheduling overhead. Generally, the computational complexity for a scheduling algorithm (including the Plain GA and EAH) grows geometrically with the number of tasks, but this is not the case with the CA + GA. GAs are known to be useful in finding good CA rules. The CA + GA method not only finds good rules over the generations of GA, but also finds good schedules across generations. The method has been tested by simulation experiments on various program graphs with different system graphs. The results show that the presented approach is effectively fast and gives good solutions. The EAH and Plain GA scheduling algorithms are very quick, and in certain cases these two algorithms can be preferred over the CA + GA.

The scope of providing different power specifications for different machines opens up many possibilities. It allows us to derive varying schedules for different systems of machines with non-identical power requirements. This in turn allows for the evaluation of different system specifications given a program graph specification, and also allows for consideration of trade-offs between time and energy—it is not unreasonable to expect that in many systems, scheduling would be for minimizing the energy rather than for makespan, or else, that some combination of constraints on both makespan and energy would enter the picture in the choice of a schedule.

There are enormous numbers of applications which await good energy-aware scheduling algorithms. Many of them are also time-bound. The calculation of time-bounded energy-aware schedules can be seen as a possible next step. Of course, in addition to time and energy, we may look ahead to develop a framework which can accommodate any constraint over the system.

With the availability of different power sources at different prices, it is desirable that a system be not just energy-aware but also cost-aware. It is relatively straightforward to extend our present work to a cost-aware setting with the inclusion of cost factors.

Thus, there is a lot of scope for further enhancement and hence for moving forward with our approach. The proposed methods open up very promising possibilities in developing generalized but sophisticated energy-aware scheduling applications that address important real-world concerns.

REFERENCES

[1] P. Agrawal and S. Rao, "Energy-aware scheduling of distributed systems using cellular automata," in *Proc. 6th Annu. IEEE Int. Syst. Conf. (SysCon'12)*, Vancouver, BC, Canada, Mar. 2012, doi:10.1109/SysCon.2012.6189481.

[2] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.

[3] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *INFORMS J. Comput.*, vol. 3, no. 2, pp. 149–156, 1991.

[4] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. New York, NY, USA: Springer, 2012.

[5] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[6] S. A. Munawar, M. Bhushan, R. D. Gudi, and A. M. Belliappa, "Cyclic scheduling of continuous multiproduct plants in a hybrid flowshop facility," *Ind. Eng. Chem. Res.*, vol. 42, no. 23, pp. 5861–5882, 2003.

[7] S. A. Munawar and R. D. Gudi, "A multilevel, control-theoretic framework for integration of planning, scheduling, and rescheduling," *Ind. Eng. Chem. Res.*, vol. 44, no. 11, pp. 4001–4021, 2005.

[8] X. Zhou, C. Chen, A. Xue, and M. Tian, "Review of optimization methods for scheduling of chemical processes based on the time representation," in *Proc. 7th World Congr. Intell. Control Autom. (WCICA'08)*, Jun. 2008, pp. 9105–9110.

[9] Y. Li, Y. Liu, and D. Qian, "A heuristic energy-aware scheduling algorithm for heterogeneous clusters," in *Proc. 15th Int. Conf. Parallel Distrib. Syst. (ICPADS'09)*, Dec. 2009, pp. 407–413, doi:10.1109/ICPADS.2009.33.

[10] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011, doi:10.1109/TPDS.2010.208.

[11] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proc. 18th ACM Symp. Oper. Syst. Principles (SOSP'01)*, 2001, pp. 103–116, doi:10.1145/502034.502045.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. 1st USENIX Conf. Operating Syst. Design Implementation (OSDI'94)*, Berkeley, CA, USA, 1994, pp. 13–23.

[13] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, Sep. 2005, doi:10.1145/1108956.1108957.

[14] J. Huang, C. Buckl, A. Raabe, and A. Knoll, "Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems," in *Proc. 19th Int. Euromicro Conf. Parallel, Distrib., Network-Based Process. (PDP'11)*, Feb. 2011, pp. 447–454, doi:10.1109/PDP.2011.10.

[15] A. Y. Zomaya and Y. C. Lee, "Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids," in *Energy-Efficient Distributed Computing Systems*, A. Y. Zomaya and Y. C. Lee, Eds. New York, NY, USA: Wiley, 2012, pp. 189–214.

[16] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogeneous computing systems," in *Proc. ACM/IEEE 13th Int. Conf. Grid Comput. (GRID'12)*, Sep. 2012, pp. 122–129, doi:10.1109/Grid.2012.32.

[17] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters," in *Proc. 7th IEEE Int. Symp. Cluster Comput. Grid (CCGRID'07)*, May 2007, pp. 541–548, doi:10.1109/CCGRID.2007.85.

[18] J. Kolodziej, S. Khan, and F. Xhafa, "Genetic algorithms for energy-aware scheduling in computational grids," in *Proc. Int. Conf. P2P, Parallel, Grid, Cloud, Internet Comput. (3PGCIC)*, Oct. 2011, pp. 17–24, doi:10.1109/3PGCIC.2011.13.

[19] *Handbook of Production Scheduling*, ser. International Series in Operations Research & Management Science, J. W. Herrmann, Ed. New York, NY, USA: Springer, 2006, vol. 89.

[20] L. T. Biegler, I. E. Grossmann, and A. W. Westerberg, *Systematic Methods of Chemical Process Design*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1997.

[21] J. Kallrath, "Planning and scheduling in the process industry," *OR Spectrum*, vol. 24, no. 3, pp. 219–250, Aug. 2002, doi:10.1007/s00291-002-0101-7.

[22] M. L. Pinedo, *Planning and Scheduling in Manufacturing and Services*, 2nd ed. New York, NY, USA: Springer, 2009.

[23] Process Systems Enterprise, 2013. [Online]. Available: http://www.psenterprise.com

[24] AspenTech, 2013. [Online]. Available: http://www.aspentech.com

[25] C. A. Mendez, J. Cerda, I. E. Grossmann, I. Harjunkoski, and M. Fahl, "State-of-the-art review of optimization methods for short-term scheduling of batch processes," *Compute. Chem. Eng.*, vol. 30, no. 67, pp. 913–946, 2006.

[26] T. Yang and A. Gerasoulis, "List scheduling with and without communication delays," *Parallel Comput.*, vol. 19, no. 12, pp. 1321–1344, Dec. 1993.

[27] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *Eur. J. Oper. Res.*, vol. 174, no. 1, pp. 23–37, Oct. 2006.
[28] I. Ahmad and Y.-K. Kwok, "On parallelizing multiprocessor scheduling problem," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 4, pp. 414–432, Apr. 1999.
[29] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113–120, Feb. 1994.
[30] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, Oct. 2008.
[31] F. Y.-P. Simon and T. Takefuji, "Integer linear programming neural networks for job-shop scheduling," in *Proc. IEEE Int. Conf. Neural Netw.*, Jul. 1988, vol. 2, pp. 341–348.
[32] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Oper. Res.*, vol. 40, no. 1, pp. 113–125, Jan. 1992.
[33] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *Eur. J. Oper. Res.*, vol. 149, no. 2, pp. 268–281, 2003.
[34] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.
[35] K.-L. Huang and C.-J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1030–1046, Apr. 2008.
[36] J. L. Schiff, *Cellular Automata: A Discrete View of the World*. New York, NY, USA: Wiley, 2007.
[37] E. W. Weisstein, "Cellular automaton," 2014. [Online]. Available: http://mathworld.wolfram.com/CellularAutomaton.html
[38] F. Seredynski and A. Zomaya, "Sequential and parallel cellular automata-based scheduling algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 10, pp. 1009–1023, Oct. 2002.
[39] A. Swiecicka, F. Seredynski, and A. Zomaya, "Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 3, pp. 253–262, Mar. 2006.
[40] T. Ghafarian, H. Deldari, and M.-R. Akbarzadeh-T, "Multiprocessor scheduling with evolving cellular automata based on ant colony optimization," in *Proc. 14th Int. CSI Comput. Conf. (CSICC'09)*, Oct. 2009, pp. 431–436.
[41] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," in *Proc. Design Autom. Conf.*, 2001, pp. 840–845.
[42] C. Artigues, P. Lopez, and A. Hait, "Scheduling under energy constraints," in *Int. Conf. Ind. Eng. Syst. Manage. (IESM'09)*, 2009.
[43] S. Wang, J.-J. Chen, Z. Shi, and L. Thiele, "Energy-efficient speed scheduling for real-time tasks under thermal constraints," in *Proc. 15th IEEE Int. Conf. Embedded and Real-Time Comput. Syst. Appl. (RTCSA'09)*, Aug. 2009, pp. 201–209.
[44] S.-H. Chan, T.-W. Lam, and L.-K. Lee, "Scheduling for weighted flow time and energy with rejection penalty," in *Proc. 28th Int. Symp. Theor. Aspects Comput. Sci. (STACS'11)*, T. Schwentick and C. Dürr, Eds., 2011, vol. 9, Leibniz International Proc. Informatics (LIPIcs), Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 392–403.
[45] Y. C. Lee and A. Zomaya, "Energy efficient resource allocation in large scale distributed systems," in *Proc. 9th Int. Symp. Distrib. Comput. Appl. Business Eng. Sci. (DCABES'10)*, Aug. 2010, pp. 580–583.
[46] *Energy-Efficient Distributed Computing Systems*, A. Y. Zomaya and Y. C. Lee, Eds. New York, NY, USA: Wiley, 2012.
[47] Y.-K. Kwok and I. Ahmad, "Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm," *J. Parallel Distrib. Comput.*, vol. 47, no. 1, pp. 58–77, 1997.
[48] A. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: Comparative studies and performance issues," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 795–812, Aug. 1999.
[49] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.
[50] L. D. Davis and M. Mitchell, *Handbook of Genetic Algorithms*. New York, NY, USA: Van Nostrand, 1991.
[51] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, Jun. 1994.
[52] R. Das, M. Mitchell, and J. P. Crutchfield, "A genetic algorithm discovers particle-based computation in cellular automata," in *Proc. Int. Conf. Evol. Comput. 3rd Conf. Parallel Problem Solving From Nature*, London, U.K., 1994, vol. 866, Lecture Notes In Computer Science, pp. 344–353.

Pragati Agrawal (S'14) received the B.Eng. degree (Hons.) in computer science and engineering, from the Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, India, in 2009. She is currently working towards the Ph.D. degree at the International Institute of Information Technology–Bangalore, Bangalore, India.

Her research interests include green computing, distributed systems, energy efficiency, and artificial intelligence.

Ms. Agrawal is a Student Member of the ACM.

Shrisha Rao (M'08–SM'13) received the M.S. degree in logic and computation from Carnegie Mellon University, Pittsburgh, PA, USA, and the Ph.D. degree in computer science from the University of Iowa, Iowa City, IA, USA.

He is an Associate Professor at IIIT-Bangalore, a graduate school of information technology in Bangalore, India. His research interests are in distributed computing, specifically algorithms and approaches for concurrent and distributed systems, and include solar energy and microgrids, cloud computing, energy-aware computing ("green IT"), and demand-side resource management.

Dr. Rao is a member of the IEEE Computer Society, the ACM, the American Mathematical Society, and the Computer Society of India.