EasyChair Preprint

№ 408

Energy-aware Task Scheduling Strategies for Multi-core Embeddded Systems

Chengming Zou, Liu Panwen and Xing Liu

August 8, 2018

Energy-aware Task Scheduling Strategies for Multi-core Embeddded Systems

Chengming Zou
Wuhan University of Technology
WuHan, China
zoucm@whut.edu.cn

Panwen Liu
Wuhan University of Technology
WuHan, China
liupanwen.bernie@whut.edu.cn

Xing Liu
Wuhan University of Technology
WuHan, China
liu.xing@whut.edu.cn

***Abstract*—In this paper, we propose two energy-aware scheduling algorithms—(1) Reinforcement learning-based multiprocessor scheduling (RL) algorithm and (2) Mathematical morphology multiprocessor scheduling (MMS) algorithm—for scheduling time-constrained Directed Acyclic Graph (DAG) tasks in an embedded multiprocessor system with Dynamic Voltage And Frequency Scaling (DVFS) and Dynamic Power Management (DPM) technology. Unlike other heuristic scheduling algorithms, the proposed reinforcement learning (RL) is a machine learning algorithm, rarely considered for energy-aware scheduling in DAG tasks. The MMS, inspired by Mathematical morphology that is often used in image processing, continuously adjusts the coded scheduling through a probe matrix to optimize energy consumption. In this paper the genetic algorithm (GA) is compared with these two proposed algorithms by rigorous simulation. The results demonstrate that our algorithms are more energy efficient. Compared with the GA algorithm, the RL and the MMS algorithm significantly improve the energy consumption reduction rate by an average of 13.37% and 72.92% respectively. In addition, MMS algorithm shows better performance in high-density and high-complexity DAG tasks.**

***Index Terms*—Energy optimization; Task scheduling; Multi-core; Embedded**

I. Introduction

With the rapid development of electronic technology, therere increasing demands for better performance of embedded systems in recent years. Applications such as image processing, high-definition television, and video games are being implemented in embedded system [1], [2]. Due to performance limitations of single-core processors nowadays, the embedded systems generally run on heterogeneous multi-core hardware platforms that include different processing units such as CPU, DSP, FPGA and so on [3]. This structure greatly improves the performance of the embedded system while brings a lot of energy consumption [4], [5], [6], [7]. As high energy consumption is heavily restricting the development of embedded systems, how to reduce the energy consumption of embedded systems has become a non-negligible problem[5]. In order to do this, the DVFS and DPM low-power technologies are widely used in heterogeneous multi-core processors, and have achieved excellent results [6], [9], [10].

In an embedded real-time system, ensuring the real-time characteristics of the task directly determines the baseline for system performance [3]. Therefore, both the time performance and energy cost should be considered in the energy optimization algorithm. Genetic Algorithm (GA), a classic heuristic algorithm, is widely applied to energy optimization in the embedded system. However, this algorithm has obvious disadvantages. For one thing , it never exploits the feedback information to adjust the search direction in the calculation. For another, it behaves poorly in local search. Therefore, the GA tends to be easily trapped in the local optimum, which inevitably limits its performance. In this paper, in hopes of overcoming these defects, we try a new machine learning approach—RL ,which is likely to show better performance, instead of improving the GA. The essence of the RL algorithm is to use continuous trial and error, with the aim of bringing good global search results, to find the optimal solution. Moreover , RL can adjust its search direction according to feedback from network search, which in turn makes it present a good local search performance. So the RL algorithm takes into account both global search and local search capabilities compared to the genetic algorithm. It will provide new inspiration for those scholars who try to find the optimal solution by using machine learning algorithms. In addition, in this paper, we propose a new optimization algorithm called MMS, inspired by mathematical morphology. Mathematical morphology has a wide range of applications in the images processing. It uses a structural element ”probe” to collect information from an image, and continuously moves the probe to examine the relationship among parts of the image [11], [12]. Using this idea, we encode the task scheduling scheme into a binary matrix and adjust the local encoding through a moving matrix probe, so that the entire encoding reaches a better state.

II. Related Work

The research on reducing the energy consumption of heterogeneous multi-cores in embedded systems has made considerable progress. The application of DVFS and DPM low-power technology further reduces the energy consumption of heterogeneous multi-core embedded systems [6], [13]. The DVFS technology provides several discrete voltage levels, and the processor can independently and dynamically adjust its own voltage supply to save energy [14], [15], [16]. DPM technology allows the idling processors to enter sleep mode to reduce power consumption [17]. The task scheduling of heterogeneous multi-cores in embedded systems can be divided into online scheduling and offline scheduling [8].

Online scheduling means that the processor does not know the scheduling scheme of the task in advance, and the processor performs real-time scheduling when the task arrives. Off-line scheduling means that the processor knows the scheduling scheme of the task beforehand and directly schedules the task when it arrives. This paper focuses on the study of offline scheduling where the DAG is used to describe precedence among tasks.

In heterogeneous multi-core embedded systems, there are many methods to minimize the energy consumption of time-constrained DAG tasks. The application of heuristic algorithms is very common. However, few people pay attention to the application of machine learning algorithms in this field. The literature [18], [19], [20], [21] uses the existing heuristic algorithm for energy optimization. The literature [18] improved the genetic algorithm and combined it with the retiming technology to reduce the energy consumption of Real-Time Streaming Applications. In the literature [19], Adaptive Genetic-Simulated Annealing algorithm that integrates the genetic algorithm and the annealing algorithm, was used to minimize the energy consumption and time of WMSN equipment. The literature [20] studied the influence of genetic algorithm parameters on energy consumption optimization of multi-core processors and improved the fitness function of genetic algorithms. The literature [21] compared the genetic algorithm and the shuffled frog leaping algorithm, and thought that the particle swarm optimization algorithm has a better optimization effect. In addition, the literature [22], [23], [24] proposes new ideas about minimizing energy consumption in multi-core. The literature [22]comprehensively considers the problem of task ordering and voltage scaling, and proposes a multiprocessor scheduling algorithm based on energy gradient. In the literature [23], a fine-grained integer linear programming (ILP) model was established for inter-processor communication-intensive scenarios, and a unified priority-based scheduling (UPS) algorithm was brought forward. The literature [24] takes into account the memory allocation problem of data, and proposes an energy optimization algorithm that considers data allocation.

III. Task Scheduling Model

A. Task Model

In a heterogeneous multi-core embedded system, a task consists of multiple jobs. Directed Acyclic Graph (DAG) is used to describe the precedence among jobs. And we apply the DVFS and DPM low-energy technologies to reduce energy consumption in model. Without loss of generality, this article describes the task model as a quadruple $S = \{P, V, T, E\}$.The heterogeneous multi-core processor contains m cores. $P = \{p_0, p_1, ..., p_{m-1}\}$ is used to represent the set of m cores,where $p_i (0 \leq i < m$ represents the i-th core. $V = \{v_0, v_1, ..., v_{k-1}\}$ represents a set of k voltage levels, where $v_i (0 \leq i < k$ represents the i-th voltage level. $T = \{T_0, T_1, ..., T_{n-1}\}$ represents a task set of n jobs,where $T_i (0 \leq i < n$ representing the i-th task. E represents an $n * n$ matrix, and $e_{ij} (0 \leq i, j < n)$ represents the element of the i-th row and the j-th column in E. If e_{ij} is non-negative, it indicates the traffic from job T_i to job T_j. Besides, the E also suggests whether there is precedence between jobs, where e_{ij} is positive or zero indicates that there is a precedence between jobs, and e_{ij} being -1 indicates that the precedence does not exist.

Algorithm 1 Generate scheduling scheme linked list

Input: $S = \{P, V, T, E\}$
Output: J , $T_{ordered}$

```
Assign a deep copy of E to E_copy
for T_i in T do
    if the job T_i has a zero in degree then
        if the job T_i is in list T_ordered then
            continue
        Add T_i to T_ordered
        for T_j in T do
            E_copy[i][j] ← −1
        i ← 0
for T_i in T_ordered do
    Randomly assign core p and voltage v to the job T_i
    J_i ← (p, v)
    Add J_i to J
return J , T_ordered
```

B. Scheduling Model

The scheduling scheme for the job T_i is described by a binary tuple $J_i = (p, v)$, which means that the job T_i is assigned to the core p and voltage v is supplied. Firstly, a task need to be preprocessed.Sort the jobs in the task by precedence shown in DAG to get an ordered list $T_{ordered} = \{T_0, T_1, ..., T_{n-1}\}$. Then, schedule each job in the task to generate an ordered list of scheduled tasks, denoted by $J = \{J_0, J_1, ..., J_{n-1}\}$.

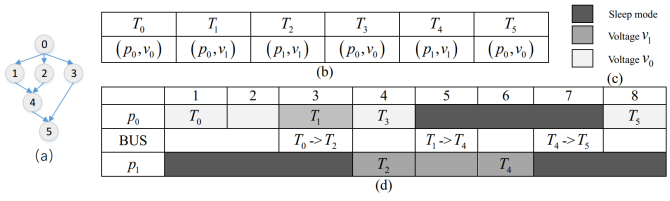

Fig. 1. Generate scheduling model.

Figure (1) shows an example where (a) is a DAG task, (b) is a corresponding task scheduling scheme list J, and (d) is the corresponding specific scheduling scheme. The following ALGORITHM 1 is the algorithm description.

C. Time Model

The direct preceding jobs of the job T_i in the DAG is $PreDAG(T_i) = \{T_j | e_{i,j} \neq -1\}$. The direct preceding job of the job T_i in the core is denoted $PreCore(T_i)$. $Start(T_i)$ and $End(T_i)$ indicate the start and end time of the job

Algorithm 2 Calculate task set execution time

Input: J

Output: t_{total} , $Finished$

1: $Finished \leftarrow \{(start_i = 0, exec_i = 0, end_i = 0)|T_i \in T_{ordered}\}$
2: **for** J_i in J **do**
3: $t\_comm_i \leftarrow max\{Finished[i].end_i + \frac{e_{i,j}}{B}|T_j \in PreDAG(T_i)\}$
4: $t\_core_i \leftarrow End(PreCore(T_i))$
5: $start_i \leftarrow max(t\_comm_i, t\_core_i)$
6: $exec_i \leftarrow \frac{Vol(T_i)}{S(J_i.p, J_i.v)}$
7: $end_i \leftarrow start_i + exec_i$
8: $Finished[i] \leftarrow (start_i, exec_i end_i)$
9: $t_{total} \leftarrow max\{Finished[i].end_i|T_i \in T_{ordered}\}$
10: **return** t_{total} , $Finished$

respectively. The time when all the communication data of job T_i's preceding jobs arrive is t_comm_i:

$$t\_comm_i = max\{End(T_j) + \frac{e_{i,j}}{B}|T_j \in PreDAG(T_i)\} \quad (1)$$

The end time of the direct preceding job of the job T_i in core is t_core_i:

$$t\_core_i = End(PreCore(T_i)) \quad (2)$$

The start time of the job T_i is the larger value between t_comm_i and t_core_i:

$$Start(T_i) = max(t\_comm_i, t\_core_i) \quad (3)$$

The execution time of the job T_i is $Exec(T_i)$. $Vol(T_i)$ denotes the job size. And the speed v at which the job T_i runs on the core p is $S(p, v)$:

$$Exec(T_i) = \frac{Vol(T_i)}{S(p, v)} \quad (4)$$

The end time of the job T_i is the sum of the start time and the execution time of the job T_i:

$$End(T_i) = Start(T_i) + Exec(T_i) \quad (5)$$

Therefore, the time at which the task T set starts to execute is 0, and the total time consumed for its execution is:

$$t_{total} = max\{End(T_i)|T_i \in T\} \quad (6)$$

The following ALGORITHM 2 is the algorithm description.Finished is a linked list that contains the start execution time $start_i$, execution time $exec_i$, and end time end_i of each job.

D. Energy consumption model

In this paper, the processor cores in the MPSoC can support Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DMP) technologies. when the voltage value v provided by core to a job, corresponding working frequency of the core is f. In addition, each core can independently perform voltage switching without affecting other cores. When the core is in sleep mode, it can maintain energy consumption at a lower level. When a scheduling scheme is determined, its total energy consumption E_{total} can be expressed as:

$$E_{total} = E_{dynamic} + E_{static} + E_{comm} + E_{sleep} + E_{tranV} + E_{tranS} \quad (7)$$

$E_{dynamic}$ indicates total dynamic energy consumption. E_{static} indicates static energy consumption. E_{comm} indicates total communication energy consumption. E_{sleep} indicates the power consumption of the core in sleep mode. E_{tranV} indicates energy consumption in switching between different running voltages. E_{tranS} indicates energy consumption in switching between modes of sleep and wake up in the cores.

Dynamic energy consumption is the energy consumed by a task when it is executed. The power model for a core running at voltage v is [25]:

$$P_{dymanic}(p, v) = C_{sw}(p) * f * v^2 \quad (8)$$

The characteristics of each core p are different. The capacitance value of the core p is $C_{sw}(p)$, and the frequency value corresponding to the voltage v is f. Then in the dynamic energy consumption model, the energy consumption of the core p at the voltage v during the running time t can be expressed as:

$$E_{dymanic} = P_{dymanic}(p, v) * t \quad (9)$$

Static voltages are generated by different leakage power sources, where the subthreshold leakage current and the reverse bias junction current contribute to most of energy consumption. Therefore, the static energy consumption can be described as [26]:

$$P_{static}(v) = I_{subn} * v + |V_{bs}| * I_j \quad (10)$$

$$E_{static} = P_{static}(v) * t \quad (11)$$

Where the I_{subn} represents the subthreshold current, the V_{bs} represents the body bias voltage, the I_j represents the reverse bias junction current, the v represents the voltage assigned to the job, and the t represents the time.

Communication energy consumption is energy consumption generated by data transmission between jobs of different cores. The communication energy consumption of job T_i from job T_j can be expressed as:

$$E_{comm}(T_i, T_j) = P_{comm} * \frac{com(T_i, T_j)}{B} \quad (12)$$

The P_{comm} represents communication power, the $com(T_i, T_j)$ represents the amount of communication data from the job T_i to the job T_j , the B represents the bandwidth between two relative cores.

Sleep energy consumption is the energy consumed by the core after it switchs to sleep mode. The P_{sleep} represents the power of the core in sleep mode and the t represents the duration of sleep mode. Therefore, the sleep power consumption can be expressed as:

$$E_{sleep}(t) = P_{sleep} * t \quad (13)$$

Algorithm 3 Calculate the energy consumption of the task set

Input: $Finished$, J
Output: E_{total}

for J_i in J **do**
  $E_{total} \leftarrow E_{total} + E_{dynamic}(J_i.p, J_i.v, Finished[i].exec_i)$
  $E_{total} \leftarrow E_{total} + sumE_{comm(T_j,T_i)|T_j \in PreDAG(T_i)}$
  $T_j \leftarrow PreCore(T_i)$
  $FreeTime \leftarrow Finished[i].start_i - Finished[j].end_j$
  $SleepEnergy = 2 * E_{tranS} + E_{sleep}(FreeTime)$
  $TranEnergy = E_{tranV}(J_j.p, J_j.v, FreeTime) + E_{dynamic}(J_j.p, J_j.v, FreeTime)$
  if $SleepEnergy > TranEnergy$ **then**
    $E_{total} \leftarrow E_{total} + TranEnergy$
    $E_{total} \leftarrow E_{total} + E_{static}(J_j.v, FreeTime)$
  else
    $E_{total} \leftarrow E_{total} + SleepEnergy$
    $v_{sleep} \leftarrow$ voltage in sleep mode
    $E_{total} \leftarrow E_{total} + E_{static}(v_{sleep}, FreeTime)$
  $E_{total} \leftarrow E_{total} + E_{static}(J_i.v, Finished[i].exec_i)$
return E_{total}

The voltage switching energy consumption is the energy consumed by the voltage switching of the core in the running mode. The time cost when the core switches from voltage v_i to voltage v_j is [27]:

$$t_{tranV}(v_i, v_j) = \frac{2 * C_{DD}}{I_{MAX}} * |v_i - v_j| \tag{14}$$

where the C_{DD} represents the capacitance of the voltage converter , the T_{MAX} represents the maximum output current of the voltage converter. The energy consumption E_{tranV} in voltage switching consists of the energy consumption E_{tranDC} in the voltage converter and the energy consumption $E_{tranCPU}$ in the core during conversion:

$$E_{tranV}(v_i, v_j) = E_{tranDC}(v_i, v_j) + E_{tranCPU} \tag{15}$$

The $E_{tranDC}(v_i, v_j)$ and the $E_{tranCPU}$ during voltage switching from v_i to v_j can be expressed as:

$$E_{tranDC}(v_i, v_j) = \alpha * C_{DD} * |v_i^2 - v_j^2| \tag{16}$$

$$E_{tranCPU} = P_{tran} * t_{tran} \tag{17}$$

Here, the α represents the efficiency factor of the voltage converter, the P_{tran} represents the power consumption of the core in the voltage conversion process.

This paper studies the energy consumption with regard to six aspects mentioned above and establishes a more comprehensive energy consumption model. The following ALGORITHM 3 is the algorithm description.

For the sake of convenience, the calculation of the time $Time$ and energy consumption En of the scheduling scheme J is described below using the $CET(J)$ function,following the style of the python language:

$$En, Time = CET(J) \tag{18}$$

IV. Energy Optimization

In this chapter, we put forward an improved genetic algorithm to achieve energy optimization based on the model constructed in the previous chapter. Then, for the first time, we use reinforcement learning to optimize the energy consumption of the DAG task. At the same time, we formulate a new optimization algorithm based on the concept of mathematical morphology to achieve the optimization of energy consumption.

A. GA Algorithm

The GA simulates the evolution process of a biological population through iterations, and reserves better individuals during each iteration in relative strong probability. After a finite number of iterations, a better solution can be obtained. The GA is described in several parts including chromosome coding, crossover, mutation, selection, and fitness functions. We use the GA to find the optimal solution based on the tasks scheduling scheme $J = \{J_0, J_1, ..., J_{n-1}\}$ proposed in Section 3.1.

(1) Chromosome encoding and decoding. Firstly, we use the scheduling scheme to construct chromosomes. Take the scheduling scheme described in Section 3.2 as an example. As shown in the Table 1, a scheduling scheme of the task for multi-core processors with two cores and two voltage levels is J. The scheduling of any job T_i requires two bits and the chromosome encoding of the scheduling scheme J can be expressed as 000111100100. One scheduling scheme corresponds to a unique chromosome encoding and vice versa.

TABLE I
Task Scheduling

Task	T_0	T_1	T_2	T_3	T_4	T_5
J_i	p_0, v_0	p_0, v_1	p_1, v_1	p_0, v_0	p_1, v_1	p_0, v_0
Coding	00	01	11	00	11	00

(2)Chromosome crossings and mutations. Figure(2) shows crossover and mutations in any two sets of chromosome codes.

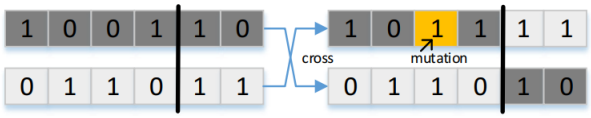

Fig. 2. Crossover.

(3)The fitness function is used to measure the quality of chromosome encoding. Suppose that the energy consumption and time of a chromosome ξ_i corresponding scheduling

scheme are respectively En_i and $Time_i$. The fitness function $fitness$ can be expressed as:

$$fitness(\xi_i) = \begin{cases} \frac{1}{En_i} & Time_i \leq Deadline \\ 0.00001 & Time_i > Deadline \end{cases} \quad (19)$$

If the schedule for a chromosome misses the deadline, the fitness value of that chromosome will not be discarded directly, which guarantee chromosome diversity. Those chromosomes whose schedules are finished before, where the lower the energy consumption of the chromosome, the greater the possibility therell be the chromosome might be passed down to the offspring.

B. RL Algorithm

1) Q-Learning algorithm principle: RL is a kind of machine learning algorithm, it can iterate the learning process through interaction with the environment, and finally realize its calculation goal. RL contains five basic components $< agent, state, action, policy, reward >$. These five components form a complete Markov model [28]. The Q-Learning algorithm is a very common method in RL. The Q-Learning system includes a state set S and an action set A. The order of round of interaction between the agent and the environment is t. When the agent selects an action a_t in the current state $s_t \in S$ through the policy, it will cause that the current state of the agent to move to $s_{t+1} \in S$ and obtain a reward r_t . The state-action mapping can be established and represented by the Q-value function $Q^{\pi}(s_t, a_t)$ which means cumulative rewards for all limited rounds of the action a_t taken under state s_t . Therefore, the agent can establish the mapping relation between the action and the state according to the Q-value function, thereby maximizing the reward. According to the state-action mapping, the Q-table can be established and updated in each round by the following formula:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma max(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)) \quad (20)$$

2) Implementation of Q-Learning Algorithm: The Q-Learning algorithm is used in this paper to solve the problem of energy consumption optimisation under time constraints. The energy optimization problem of DAG task scheduling in heterogeneous multi-core processors is an NP-hard problem [30]. The DVFS and DMP technologies are considered in the task scheduling model, which makes the problem more complicated. In this paper, we do not pay attention to the convergence of the algorithm, instead the computational process.

(1)The establishment of state and action models. The A is an action set, according to the task model established in Section 3.1:

$$A = \{(p, v) | p \in P, v \in V\} \quad (21)$$

The contains all the combination of the elements in the set P and the elements in the set V , so the elements of the set are known. The S is the state set, according to the scheduling model established in Section 3.1:

$$S = \{(J, i) | J_i \in J\} \quad (22)$$

Algorithm 4 Calculate task set execution time

Input: J
Output: $bestEn$, $time$

procedure $INITIALIZATION()$
  Initialize Q-table $Q[S, A]$ by 0
  Initialize $bestEn$, $time$ using (18)
  Initialize a state element $s \leftarrow (J, i)$
  Initialize a action element $a \leftarrow (p, v)$
  return s,a
procedure $TRANSFER\_STATE(s, a, bestEn)$
  $s\_$ ←Calculate using (23)
  r ←Calculate using (24)
  If r equal to 1 ,update $bestEn$,$time$ using (18)
  return s_,r
procedure $RL\_Brain()$
  $s, a \leftarrow INITIALIZATION()$
  $episodes$ ←Set cycle times
  for t between 0 to $episodes$ **do**
    $s\_, r \leftarrow TRANSFER\_STATE(s, a, bestEn)$
    $a\_ \leftarrow a_i$,where $a_i \in A$ with maximum $Q[s, a]$ value
    Update Q-table $Q[S, A]$ using (20)
    $s \leftarrow s\_$,$a \leftarrow a\_$
return $bestEn$, $time$

Each element in the set S consists of a complete schedule J and a pointer i, with the pointer pointing to a job in the task. Due to the uncertainty of J , the content of the set S is unknown but limited, and the set S is automatically filled during program execution. The interactive process between the agent and the environment at the t-th time can be described as: After the agent selects an action $a_t = (p, v)$, the agent will switch from the current state $s_t = (J, i)$ to the new state s_{t+1}. The rule here is to replace the contents of the i-th element J_i in the set J with (p, v) to generate a new set J' . And the s_{t+1} expressed as:

$$s_{t+1} = (J', (i+1) mod\, n) \quad (23)$$

(2)Reward mechanism. The energy and time cost corresponding to the current state s_t are respectively En_t and $Time_t$ respectively. The corresponding time and energy consumption of s_{t+1} are En_{t+1} and $Time_t$:$En_t, Time_t = CET(J)$, $En_{t+1}, Time_{t+1} = CET(J')$. And $r_t = 1$ only when $En_{t+1} > En_t$ and $Time_{t+1} < Deadline$, or $r_t = 0.1$.

$$r_t = \begin{cases} 1 & En_{t+1} > En_t \, and \, Time_{t+1} < Deadline \\ 0.1 & others \end{cases} \quad (24)$$

The Q-learning algorithm is described as follows ALGORITHM 4.

C. MMS Algorithm

1) The principle of MMS algorithm: The idea of the MMS algorithm comes from mathematics morphology in image processing. According to the encoding of the task scheduling

scheme J in Section 4.1, the encoding of the d bits is structured. Then the encoding is folded e times, divided into a number of units of c bits each, where c and e meet formula $c * e = d$. Therefore, the encoding matrix EM as shown in the left Figure3 can be structured after the coding process, and each encoding matrix matches a unique scheduling scheme J. We then select a $g * h$ matrix probe PM, where $g \in (0, c)$, $h \in (0, e)$ is the height and width of the probe, respectively.

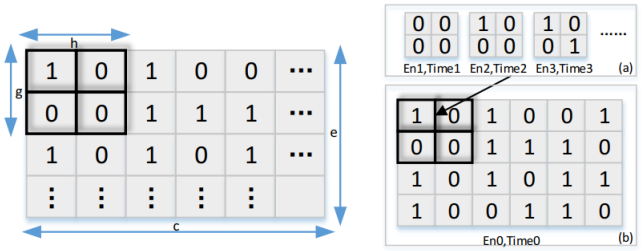

Fig. 3. Example of MMS algorithm.

The probe starts from the upper left corner of the coding matrix, and moves from left to right. After moving to the rightmost side, the probe moves down one step. It repeats the left-to-right process until the probe moves to the bottom right corner of the matrix, completing one round of traversal. As shown in the right Figure3(b) is a 6*4 coding matrix, the probe is a 2*2 matrix, 0 and 1 can be arranged in the 2*2 matrix to get 16 groups of arrangements, as Figure3(a) shows. In each round of traversal, the probe uses the CET function to calculate the energy consumption En0 and timeTime0 of the current encoding matrix before each movement. Then we successively replace the area pointed by the probe with the 16 groups in the right figure, and calculate the energy consumption and time corresponding to the replacement, as is shown in Figure3(a). Remove the replacement from the groups whose running time exceeds the deadline, and select the replacement group with the lowest energy consumption and less than En0 from the remained groups to replace the original code, then move the probe backward. Repeating the above operation above can make the energy consumption decrease continuously and eventually stabilize near certain level. In order to prevent the MMS algorithm from falling into a local optimum trap, when the energy consumption is stable near a certain value, the code at different positions in the coding matrix is randomly selected to be inverted to make the algorithm stable near another energy consumption value. The minimum among these values is the optimized result.

2) Implementation of the MMS Algorithm: A scheduling scheme J is initialized at random, corresponding to only one encoding matrix EM of $c * e$. The mutual transformation of the two can be expressed as:

$$EM = Reshape(J, c, e) \tag{25}$$

$$J = Restore(EM) \tag{26}$$

$RM = RM_0, RM_1, ..., RM_u$represents a set of $u(u = A_{g*h}^{g*h})$ elements. The elements in the set are permutations of 0 and 1 in the probe matrix PM, where RM_i denotes the i-th permutation. The coordinate of the lower right corner of the region pointed by the probe PM in the encoding matrix EM is denoted by (x, y), and the coordinate at the upper left of the region pointed by the probe is $(x - g + 1, y - h + 1)$. Before the probe moves, a new coding matrix EM_i is obtained by replacing the area pointed by the probe with RM_i , expressed as:

$$EM_i = EM^{(x,y)} \oplus RM_i \tag{27}$$

The $\oplus$ symbol indicates a local replacement operation. Calculate the time and energy consumption of the original EM matrix and the newly generated coding matrix EM_i: $En, Time = CET(Restore(EM))$,$En_i, Time_i = CET(Restore(EM_i))$. Find a code matrix EM_i whose time and energy consumption meets the following requirements from the newly generated coding matrix EM_i:

$$\begin{cases} Time_i < Deadline \\ En_i < En \\ En_i = max(En_j | j \in [0, u)) \end{cases} \tag{28}$$

Then replace the original encoding matrix with this encoding matrix EM_i:

$$EM = EM_i \tag{29}$$

Then the probe continues to move. Suppose the probe moves s_row steps to the right and s_col steps down. Here is a description of the MMS algorithm in ALGORITHM 5.

V. Experimental Results And Discussion

We compared the GA, RL and MMS algorithm simulation experiments results. The three algorithms are written in python. The simulation environment is in Ubuntu 16.04.3LTS, Intel E7-4820 v3/1.9GHz. The experiment simulates the voltage and frequency level at which the processor operates [22] , shown in Table 2:

TABLE II
Voltage and Frequency

Voltage(v)	Frequency(MHz)
0.75	150
1.0	400
1.3	600
1.6	800
1.8	1000

In section 4.2.2, the capacitance of the dynamic capacitor is C_{sw} = 12nF [31]. In the static energy consumption calculation formula, the subthreshold current is I_{subn}=250uA [32], the body bias voltage is V_{bs}=0.4V [31], and the reverse bias junction current is I_j=4.8*10-10A [26]. In the communication energy consumption calculation, the bandwidth is B=10 Mb/s and the communication power is P_{comm}=0.147w [31]. In the sleep energy consumption calculation, the sleep state power is P_{sleep}=2.4w [27]. In the dynamic voltage switching energy consumption calculation, the efficiency factor of the voltage converter is α=0.9 [32], the capacitance of the voltage converter is C_{DD}=12pF [31], and the power consumption

Algorithm 5 MMS algorithm

Input: J
Output: $bestEn$, $time$

1: $EM = Reshape(J, c, e)$
2: $En, Time \leftarrow CET(J)$
3: Record En with $Localoptimum\_EN$
4: $episodes$ ←Set cycle times
5: **for** t between 0 to $episodes$ **do**
6: **if** $Localoptimum\_EN \leq En$ **then**
7: **if** $bestEn > En$ **then** $bestEn \leftarrow En$,$time \leftarrow Time$
8: Randomly Inverting $\frac{c*e}{2}$-bit EM coding
9: $En, Time \leftarrow CET(Restore(EM))$
10: $Localoptimum\_EN \leftarrow En$
11: $x \leftarrow g-1$,$y \leftarrow h-1$
12: **while** $x < e$ **do**
13: **while** $y < c$ **do**
14: **for** RM_i in RM **do**
15: $EM_i \leftarrow EM^{(x,y)} \oplus RM_i$
16: **if** $Time > Deadline$ **then**
17: **if** $Time_i < Time$ **then**
18: $EM \leftarrow EM_i$
19: $En \leftarrow En_i$,$Time \leftarrow Time_i$
20: **else**If$En_i < En$
21: $EM \leftarrow EM_i$
22: $En \leftarrow En_i$,$Time \leftarrow Time_i$
23: $J' \leftarrow Restore(EM_i)$
24: $y \leftarrow y + s\_row$
25: $x \leftarrow x + s\_col$
26: **return** $bestEn$, $time$

during the voltage conversion process is P_{TRAN}=1w[24]. The average power consumption in switching between sleep and wake-up mode is $E_{transS} = 3.5 * 10 - 6J$.

The experiment randomly generates 40 tasks based on the number of jobs and the number of the DAG edges. The attributes of each task included the number of tasks, the number of the DAG edges, the number of cores, and the deadline. In each group of experiments, the number of tasks denoted tn is a random number between 10 and 100. The number of allowed edges denoted en in the DAG graph is closely related to variate tn. According to the calculation, the number of edges in the DAG graph is within the range $en \in [tn-1, tn*(tn-1)/2]$. In order to ensure the authenticity of the real situation, the number of DAG edges is chosen randomly in the range $[tn, tn + 20]$ in the experiment [33]. According to the number of selected jobs and the number of DAG edges, we will randomly generate DAGs. The number of cores is 2, 4, and 8 in this experiment. The completion time of a task is closely related to the number of jobs. Considering the fact that the more processing cores there are, the less time constraint is required for the completion of task execution, to ensure that the task has sufficient time to complete, we allocate an average of 2us, 1.5us, and 1us of running time to each job in the case of 2, 4, and 8 cores respectively. And thus the task sets the deadline for running time at tn*2us, tn*1.5us and tn*1us respectively. We use energy drop rate to describe the optimisation effect of the algorithm on energy consumption. The drop rate of energy consumption (DREC) is the proportion of the drop rate of current energy consumption of the task to energy consumption of the task performed in a single core at the highest frequency. As shown in the following table, Save (%) represents the energy consumption drop rate, and Time(measured in u.s.)indicates the time consumed by task execution. The units are (us). GA, RL, and MMS represent the three algorithms. The experimental results are shown in the Table 3.

TABLE III
RANDOM TASKS

Num	Task/Edge/core	Deadline(us)	Save(%)			Time(us)		
			GA	RL	MMS	GA	RL	MMS
1	88/105/8	88.0	37.51	40.59	81.91	47.43	45.73	87.50
2	86/99/8	86.0	37.29	39.61	81.96	45.61	46.07	85.42
3	13/17/2	26.0	56.73	68.05	67.58	24.80	25.68	25.75
4	25/25/2	45.0	48.01	58.88	80.56	31.48	39.70	49.83
5	32/35/8	32.0	43.10	51.83	36.62	27.23	27.60	23.26
6	59/61/2	118.0	43.15	45.23	79.62	63.03	66.80	118.00
7	65/70/8	65.0	35.34	42.76	82.41	41.93	35.60	64.17
8	63/64/8	63.0	37.12	43.76	81.74	30.70	33.07	62.92
9	31/32/8	31.0	50.53	53.78	26.94	25.52	27.55	17.98
10	58/76/4	87.0	41.69	48.39	82.63	50.33	56.86	86.67
11	68/78/2	136.0	38.74	41.33	82.64	65.28	61.48	135.33
12	14/26/8	14.0	50.75	50.35	60.76	13.42	13.10	13.92
13	68/83/8	68.0	40.04	41.42	82.19	39.77	41.40	67.67
14	56/58/2	112.0	43.72	48.17	82.24	56.20	71.03	111.58
15	43/51/2	86.0	44.42	51.67	82.35	46.43	56.43	85.25
16	21/35/2	42.0	51.30	59.68	76.92	38.05	33.81	41.67
17	29/37/2	58.0	46.02	55.25	79.28	34.59	35.21	58.00
18	77/86/2	154.0	38.09	43.74	82.64	78.16	77.52	151.33
19	23/40/2	46.0	47.50	54.28	77.00	35.89	35.04	45.67
20	46/65/4	69.0	49.88	54.30	80.46	37.68	42.08	68.67
21	10/28/4	15.0	46.61	61.76	61.06	13.50	11.58	14.93
22	76/89/8	76.0	35.94	44.08	77.22	35.28	47.09	75.95
23	28/29/8	28.0	44.65	51.32	74.18	21.87	23.70	27.75
24	17/36/2	34.0	57.81	62.12	69.50	27.93	32.83	33.58
25	37/54/2	74.0	51.25	47.46	79.97	52.48	43.08	73.25
26	37/41/4	55.5	45.40	54.64	81.94	54.06	44.63	54.75
27	63/67/2	126.0	40.08	42.51	82.64	56.08	66.38	124.00
28	57/73/8	57.0	45.20	43.56	79.78	46.20	34.06	56.42
29	24/39/4	36.0	47.17	59.87	71.12	27.65	35.02	35.93
30	69/83/2	138.0	38.94	47.89	81.72	71.98	75.20	137.75
31	90/94/2	180.0	35.67	43.42	82.64	77.13	98.91	174.67
32	61/80/2	122.0	41.95	44.39	82.27	49.92	70.06	120.92
33	67/72/4	100.5	42.50	43.42	82.37	44.73	52.86	100.42
34	74/89/2	148.0	38.19	44.95	82.64	77.18	71.18	144.00
35	15/30/8	15.0	44.58	58.44	52.65	14.82	14.19	14.94
36	59/64/4	88.5	43.02	47.42	82.64	43.83	58.55	84.67
37	45/51/4	67.5	45.88	49.05	82.08	37.65	48.65	65.83
38	76/87/8	76.0	36.72	44.18	81.51	41.23	50.60	75.58
39	82/99/4	123.0	37.42	43.34	82.64	58.53	55.82	118.67
40	29/49/4	43.5	50.39	57.57	67.92	32.41	26.80	43.43
Average		99.5	43.76	49.61	75.67	42.95	45.82	74.45

On line 12 and 28 in the table above, the DREC of the RL algorithm is lower than that of the GA algorithm; on the 5th and 8th line, the DREC of the MMS algorithm is lower than the GA algorithm. The most likely reason here is that the RL algorithm and the MMS algorithm have fallen in the local optimum trap. However, on other lines, the DREC of the RL

algorithm and the MMS algorithm is obviously greater than that of the GA algorithm. The MMS algorithm has a much lower DREC than the RL algorithm except on the 5th and 5th rows. From the perspective of time, the time consumption of the tasks calculated by the RL and MMS algorithms on most of the lines is larger than the GA algorithm under the time constraint. Based on the average energy consumption reduction rate listed in the table above, the performance of the three algorithms in energy consumption optimization is evident. Compared with the GA algorithm, the DREC of the RL algorithm and the MMS algorithm improves by 13.37% and 72.92%, respectively. However, it can be clearly seen that the DREC of the MMS algorithm is far higher than that of the RL algorithm. Under the time constraint, RL algorithm and MMS algorithm can make fuller use of time than GA algorithm. In particular, the time consumption calculated by MMS algorithm is almost close to the constraint time.

In order to compare the performance of the algorithm under different numbers of cores, we perform 10 groups of experiments on the number of 2, 4, and 8 cores, and sorts the experimental results according to the number of tasks. The number of tasks, the number of DAG edges, and deadline time are all generated according to the rules above-mentioned. The experimental results are shown in the Table 4.

In the table above, we observe through comparison of energy consumption and time consumption that the DREC calculated by the three algorithms is always MMS>RL>GA under the conditions of 2-core, 4-core and 8-core. When their deadlines are is the same, the time consumption of the task calculated by the three algorithms is always MMS<RL<GA. In other words, the time utilization rate of the three algorithms is ordered MMS>RL>GA. With regard to the relation between the number of jobs and the DREC, the DREC of GA and RL decreases with the number of tasks increasing under the conditions of 2-core, 4-core, and 8-core; on the contrary, the DREC of the MMS algorithm increases when the number of jobs decreases. Therefore , MMS ,compared with RL and GA, is more suitable for energy optimization under complex task conditions and has excellent results. Comparing the DREC of the three algorithms under different conditions for the number of cores, the, we find that, except for a few cases in the above table, when the same task is executed under the conditions of 2-core, 4-core and 8-core, the DREC calculated by three algorithms decreases with the number of cores increasing. This shows that an increased number of cores doesnt necessarily reduce energy consumption, it may increase energy consumption instead. In order to observe the effects of the deadline on the DREC and time consumption of the three algorithms, in this chapter we select a task with 60 jobs and 69 DAG edges (represented as task (60,69).), and then randomly generate a DAG.And then we select several deadline 20us to 130us with same intervals. In the case of 2 cores and 4 cores respectively, the three algorithms use these deadlines as conditions to optimize the energy consumption of the selected task. As shown in the figure below, Figure(4) shows the relationship between the deadline and the DREC,

TABLE IV
IN CASE OF 2-CORE,4-CORE,8-CORE

Task/Edge /Deadline	Save(%)			Time(us)		
	GA	RL	MMS	GA	RL	MMS
2 core						
10/13/20.0	74.98	68.24	77.39	16.83	18.93	19.92
11/24/22.0	60.34	68.07	58.32	19.68	20.55	21.93
13/17/26.0	56.73	71.09	67.58	24.80	25.92	25.75
13/14/26.0	54.47	68.56	69.89	17.17	16.52	25.57
32/35/64.0	43.35	50.60	78.81	41.37	38.67	63.83
33/52/66.0	43.15	63.11	79.99	39.57	51.10	65.83
59/61/118.0	43.15	47.28	79.62	63.03	57.92	118.00
63/80/126.0	37.92	46.66	81.89	67.01	74.56	125.42
84/103/168.0	37.76	41.72	82.64	74.14	77.34	161.33
88/105/176.0	37.48	42.15	82.64	86.63	81.98	173.33
Average	48.93	56.75	75.87	45.02	46.35	80.09
4 core						
10/13/15.0	63.46	71.01	76.60	10.75	12.83	14.83
11/24/16.5	56.62	59.12	60.53	14.70	14.70	16.43
13/17/19.5	52.34	68.68	53.78	18.84	17.58	19.40
13/14/19.5	55.20	67.46	66.40	19.37	17.98	19.08
32/35/48.0	48.39	53.94	75.06	38.20	44.97	47.75
33/52/49.5	54.30	48.48	77.88	44.05	37.17	49.42
59/61/88.5	37.38	44.35	82.26	43.87	46.12	85.58
63/80/94.5	39.54	44.79	82.64	60.43	57.73	89.33
84/103/126.0	38.51	44.79	82.64	65.03	70.38	108.67
88/105/132.0	38.98	43.32	82.64	62.23	59.75	120.00
Average	48.47	54.59	74.04	37.75	37.92	57.05
8 core						
10/13/10.0	51.10	62.04	66.82	9.43	9.43	9.92
11/24/11.0	40.78	53.14	53.00	10.33	10.85	10.88
13/17/13.0	47.21	56.88	63.78	12.43	12.18	12.92
13/14/13.0	49.70	61.92	64.67	10.63	12.58	12.83
32/35/32.0	43.10	51.85	36.62	27.23	31.40	23.26
33/52/33.0	45.82	48.87	69.53	24.20	31.08	32.90
59/61/59.0	39.44	46.42	79.70	33.88	30.43	58.83
63/80/63.0	38.29	43.79	78.89	36.89	43.95	62.42
84/103/84.0	40.67	44.94	79.50	44.54	45.30	83.70
88/105/88.0	37.51	43.07	81.91	47.43	50.50	87.50
Average	43.36	51.29	67.44	25.70	27.77	39.52

where GA(2) represents the curve of the GA algorithm under the condition of 2 cores, and the same to the others. Figure(5) shows the relationship between the time consumption of the task and the deadline. Save (%) represents the DREC, time (us) represents time consumption, and deadline (us) is the given deadline.

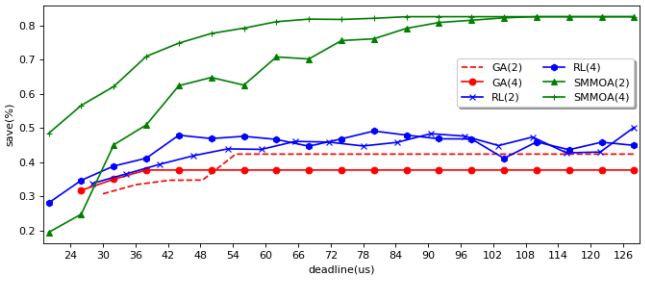

Fig. 4. Deadline and DREC.

The greater the number of cores is, the more in parallelism the task will be performed, and the less time the task will cost, and vice versa. It can be seen from Figure(4) that the three

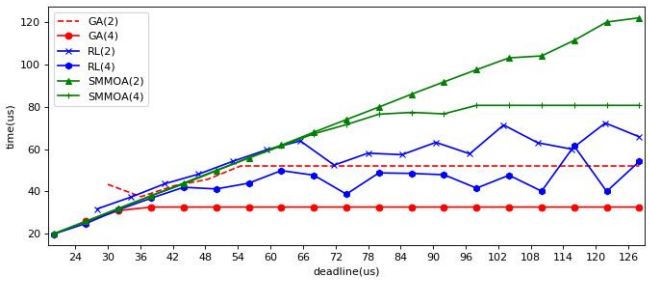

Fig. 5. Deadline and time consumption of tasks.

energy curves GA(2), GA(4), and RL(2) only bend downwards after the deadline value on the axis of abscissa is 30us, 26us, 28us on the axis of abscissa. However, curves MMS(2) and MMS(4) does so in the whole picture. Especially, MMS(2) in particular shows that the MMS algorithm can still calculate optimal results for task (60, 69) under such extreme conditions as 2-core and 20-us deadline, which indicates that is has a very pretty good effect on high-density task calculation. Therefore, in the optimization of energy consumption for high-density tasks, the performance of the three algorithms is MMS>RL>GA. In addition, it can be observed from the Figure(5) that as the deadline time increases, the DREC of the curve corresponding to the MMS algorithm will continuously increase and eventually stabilize, and the DREC of the curve corresponding to the RL algorithm has a small increase initially, then stabilizes in a certain area, and the DREC of the curve corresponding to the GA algorithm becomes a smooth straight line and no longer increase after an initial small increase. The trend of the time consumption curve of the three algorithms is roughly consistent with that shown in Figure(4). Therefore, it can be drawn that the MMS algorithm can make full use of time and try to find the optimal DREC before the deadline; the RL algorithm will have different performance in DREC for different deadline, but it cannot form a trend of stable and healthy development; while the DREC of the GA algorithm does not substantially change within the deadline. The deadline duration utilisation of the three algorithms in the deadline is MMS>RL>GA.

VI. Conclusion And Future Work

In this paper, we proposed two algorithms to minimize the energy consumption in heterogeneous multi-core embedded systems. Appropriate energy models and scheduling models were established as a result. In the scheduling model, the DVFS and DPM low-energy technologies were applied to reduce energy consumption, where tasks can be assigned to different cores and provide voltages according to energy demand. Meanwhile,the cores will switch into a sleep mode to further reduce energy consumption when the core is standing idle. In the energy model, the five main aspects in energy consumptions including the dynamic energy consumption, the sleep energy consumption, the communication energy consumption, the dynamic voltage switching energy consumption, and the sleep mode switching voltage were taken into consideration, which altogether improved the simulation's authenticity. Unlike the application of heuristics in this field, we proposed a RL algorithm. The RL algorithm is a kind of machine learning algorithm. Compared with GA algorithm, the RL algorithm can use feedback information to adjust its own search direction, greatly improving the search efficiency. In addition, we also proposed a new heuristic algorithm, a mathematical morphology optimization algorithm. This algorithm can adjust the interrelationship between local codes through probes, causing unexpectedly positive. In the experiment, we implemented the GA, RL and MMS algorithm. And we compared the performance of these algorithms in many ways. The calculation result of machine learning algorithm is largely dependent on the configuration of parameters and structure. Therefore, in the next step, we will strive to adjust parameters and structure of the RL algorithm so as to obtain good experimental results.

References

[1] M. B. Rutzig, A. C. S. Beck, and L. Carro, "Adaptive and dynamic reconfigurable multiprocessor system to improve software productivity," IET Computers & Digital Techniques, vol. 9, pp. 63-72, 2015.

[2] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-Core Processors," IEEE Trans. Computers, vol. 66, pp. 1878-1889, 2017.

[3] Y. Li, J. Niu, M. Atiquzzaman, and X. Long, "Energy-aware scheduling on heterogeneous multi-core systems with guaranteed probability," Journal of Parallel and Distributed Computing, vol. 103, pp. 64-76,2017.

[4] A. Ilic, F. Pratas, and L. Sousa, "Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-Cores," IEEE Trans. Computers, vol. 66, pp. 52-58, 2017.

[5] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors," IEEE Transactions on Parallel and Distributed Systems, vol. 19, pp. 1540-1552, 2008.

[6] P. P. Nair, A. Sarkar, N. M. Harsha, M. Gandhi, P. P. Chakrabarti, and S. Ghose, "ERfair Scheduler with Processor Suspension for Real-Time Multiprocessor Embedded Systems," ACM Trans. Des. Autom. Electron. Syst., vol. 22, pp. 1-25, 2016.

[7] X. Li, N. Xie, and X. Tian, "Dynamic Voltage-Frequency and Workload Joint Scaling Power Management for Energy Harvesting Multi-Core WSN Node SoC," Sensors, vol. 17, p. 310, 2017.

[8] G. Zeng, Y. Matsubara, H. Tomiyama, and H. Takada, "Energy-aware task migration for multiprocessor real-time systems," Future Generation Computer Systems, vol. 56, pp. 220-228, 2016.

[9] M. K. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques," Real-Time Systems, vol. 47, pp. 143-162, Mar 2011.

[10] S. Akram, J. B. Sartor, and L. Eeckhout, "DEP+BURST: Online DVFS Performance Prediction for Energy-Efficient Managed Language Execution," IEEE Trans. Computers, vol. 66, pp. 601-615, 2017.

[11] A. Bouchet, J. I. Pastore, M. Brun, and V. L. Ballarin, "Compensatory fuzzy mathematical morphology," Signal, Image and Video Processing, vol. 11, pp. 1065-1072, 2017.

[12] J. P. Serra, "Image analysis and mathematical morphology " Academic Press, 1982.

[13] A. Colin, A. Kandhalu, and R. Rajkumar, "Energy-Efficient Allocation of Real-Time Applications onto Single-ISA Heterogeneous Multi-Core Processors," Journal of Signal Processing Systems, vol. 84, pp. 91-110, 2016.

[14] X. Lin, Y. Z. Wang, Q. Xie, and M. Pedram, "Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment" IEEE Transactions on Services Computing, vol. 8, pp. 175-186, Mar-Apr 2015.

[15] Z. M. Zhang and J. M. Chang, "A Cool Scheduler for Multi-Core Systems Exploiting Program Phases," IEEE Transactions on Computers, vol. 63, pp. 1061-1073, May 2014.

[16] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P.-O. ?stberg, and E. Elmroth, "Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling," Future Generation Comp. Syst., vol. 81, pp. 114-128, 2018.
[17] Y. Kim, K. S. Lee, and C. G. Lee, "Energy Efficient Real-Time Scheduling Using DPM on Mobile Sensors with a Uniform Multi-Cores," Sensors, vol. 17, p. 22, Dec 2017.
[18] Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, and E. H.-M. Sha, "Overhead-aware energy optimization for real-time streaming applications on multiprocessor System-on-Chip," ACM Trans. Des. Autom. Electron. Syst., vol. 16, pp. 1-32, 2011.
[19] X. Liu, H. Y. Zhou, J. W. Xiang, S. W. Xiong, K. M. Hou, C. de Vaulx, et al., "Energy and Delay Optimization of Heterogeneous Multicore Wireless Multimedia Sensor Nodes by Adaptive Genetic-Simulated Annealing Algorithm," Wireless Communications & Mobile Computing, p. 13, 2018.
[20] N. Kumar and D. P. Vidyarthi, "A GA based energy aware scheduler for DVFS enabled multicore systems," Computing, vol. 99, pp. 955-977, Oct 2017.
[21] W. Z. Zhang, H. C. Xie, B. R. Cao, and A. M. K. Cheng, "Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm," Mathematical Problems in Engineering, p. 9, 2014.
[22] L. K. Goh, B. Veeravalli, and S. Viswanathan, "Design of Fast and Efficient Energy-Aware Gradient-Based Scheduling Algorithms for Heterogeneous Embedded Multiprocessor Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 20, pp. 1-12, Jan 2009.
[23] L. Yang, W. Liu, W. Jiang, M. Li, J. Yi, and E. H. M. Sha, "Application Mapping and Scheduling for Network-on-Chip-Based Multiprocessor System-on-Chip With Fine-Grain Communication Optimization," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, pp. 3027-3040, 2016.
[24] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-Aware Data Allocation and Task Scheduling on Heterogeneous Multiprocessor Systems With Time Constraints," IEEE Transactions on Emerging Topics in Computing, vol. 2, pp. 134-148, 2014.
[25] C. Y. Yang, J. J. Chen, and T. W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in Design, Automation and Test in Europe Conference and Exhibition, Vols 1 and 2, Proceedings, N. Wehn and L. Benini, Eds., ed Los Alamitos: IEEE Computer Soc, 2005, pp. 468-473.
[26] S. M. Martin, K. Flautner, T. Mudge, D. Blaauw, Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. New York: IEEE, 2002.
[27] B. C. Mochocki, X. B. S. Hu, and G. Quan, "A unified approach to variable voltage scheduling for nonideal DVS processors," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, pp. 1370-1377, Sep 2004.
[28] J. Andreas, D. Klein, and S. Levine, "Modular Multitask Reinforcement Learning with Policy Sketches," 2017.
[29] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, D. Silver, et al., "Successor Features for Transfer in Reinforcement Learning," 2017.
[30] P. Ramesh and U. Ramachandraiah, "E-Token Energy-Aware Proportionate Sharing Scheduling Algorithm for Multiprocessor Systems," Wireless Communications & Mobile Computing, vol. 2017, pp. 1-6, 2017.
[31] AMD, "Mobile AMD Athlon 4 processor model 6 CPGA data sheet. Advanced Micro Devices," Tech, 2001.
[32] BURD, "Energy-effcient processor system design," Department of Electrical Engineering and Computer Sciences, 2001.
[33] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on, 1998, pp. 97-101.