Extending Lifetime of Battery-powered Coarse-Grained Reconfigurable Computing Platforms

Shouyi Yin, Peng Ouyang, Leibo Liu and Shaojun Wei

Institute of Microelectronics
Tsinghua University, Beijing, P.R.China
Email: yinsy@tsinghua.edu.cn

***Abstract*—The coarse-grained reconfigurable architecture (CGRA) is a promising platform for mobile computing. In this paper, how to prolong the lifetime of battery-powered reconfigurable computing platform is addressed. Considering the nonlinear characteristics of battery, a multi-objective optimization model is built for extending the lifetime of battery. Based on this model, a joint task-mapping and battery-scheduling method is proposed. The experimental results show that the proposed method achieves 26.22% improvement of battery runtime on average comparing to the state-of-the-art methods.**

I. Introduction

The coarse-grained reconfigurable architecture (CGRA) is characterized by the high parallel computation and flexibility of reconfiguration, and exhibits high area efficiency and energy efficiency [1]. It is becoming a promising platform for mobile computing [2]. As shown in Fig.1, a typical CGRA is composed of a host controller and a 2-D PE (processing element) array. Each PE can be configured to perform different word-level operations. The PEs are power-gated so that the unused PEs can be closed to reduce power consumption. For a complex application, it is usually divided into several tasks. Then these tasks are mapped onto the PE array consecutively. Different task mapping schemes on PE array bring different energy consumption.

In mobile computing, the platforms are usually powered by the batteries and extending the battery runtime is a primary concern. Reducing the energy consumption of the task execution will extend the battery runtime [3]. However, because of the nonlinear discharging characteristics of battery, the battery runtime is not only decided by the lower energy consumption but also the battery state [4] that can be denoted as the residual energy units of the battery. The nonlinear characteristics include the recovery effect and the ratio capacity effect. Based on the nonlinear characteristics, some battery-aware task scheduling methods have been proposed to improve the battery runtime, such as [5] [6]. However, in the mobile or embedded application, the amount of concurrent tasks is relatively small. And the tasks usually have strict dependencies. Therefore the task scheduling space is limited, which constrains the battery runtime to be fully extended.

This work is supported in part by the China Major S&T Project (No.2013ZX01033001-001-003), the International S&T Cooperation Project of China grant (No. 2012DFA11170), the Tsinghua Indigenous Research Project (No.20111080997), the NNSF of China grant (No.61274131) and the China 863 Program (No.2012AA012701).

978-3-9815370-2-4/DATE14/©2014 EDAA

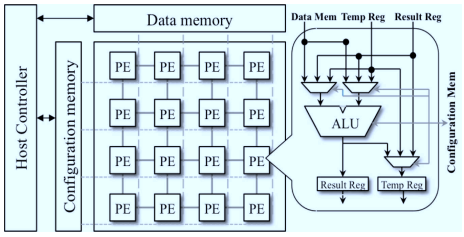


Fig. 1. The typical architecture of CGRA

On the other hand, the multi-cell structure is adopted in the practical battery for mobile devices. For example, there are usually three parallel connecting cells in tablet PC's (or PAD's) battery and each cell is capable to power the device individually. Taking the nonlinear characteristics into account, we are inspired that the battery-cell scheduling is a good way to avoid the dependency constraint of the tasks. The battery-cell scheduling means that we dynamically switch battery-cell which powers the mobile device in runtime and let the other cells stay in idle state. In literature, there are some previous work about multi-battery scheduling algorithms, such as [7] and [8], which can be also used in battery-cell scheduling. These algorithms leverage recovery effect in switching batteries so that the battery runtime can be extended. However these methods are not efficient for the reconfigurable computing platforms since the task mapping constrained by the features of the reconfigurable architectures are not considered well.

On the CGRA based mobile platforms, task mapping brings a new searching space so that the problem of the battery-cell scheduling should be redefined. Since the battery runtime is related to the energy consumption and battery state (i.e. residual battery capacity), we take these two factors as two objectives, and explore the co-optimization methodology to reduce the energy consumption and sustain the high battery state. We propose a joint optimization method which combines task mapping and battery-cell scheduling together. Comparing to the state-of-the-art methods, the proposed algorithm can extend 26.22% battery runtime averagely.

II. Battery model

We adopt the Markov battery model [9] as the power source for the CGRA. This model divides the whole discharging time into equal time slots, and use the Markov state to

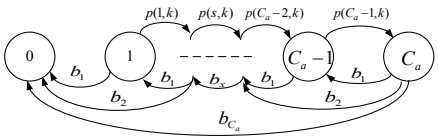


Fig. 2. The Markov battery model

TABLE I. PARAMETERS DEFINITION

B_n	the number of batteries.
C_t	the theoretic energy units of battery.
C_a	the actual energy units the battery provide($C_a < C_t$)
(s_{ij}, k_{ij})	the state of j battery cell is s_{ij} when k_{ij} energy units have been consumed in the i time slot.
s_{ij}	represents the state of the j battery in the Markov chain, $s_{ij} \in \{0, 1, 2....C_a\}$.
k_{ij}	the energy units the j battery has provided before the i time slot.
u	represents the discharging stage.
u_m	the total stages of discharging, $u \in \{1, 2...u_m\}$, u begins when c_u energy units have been drained out and ends when c_{u+1} energy units have been drained out. ($c_1 = 0, c_{u_m} = C_a$)
E_r	represents the energy units recovered during one time slot because of the recovery effect of battery.

represent the battery state. As Fig.2 shown, s and k denote the Markov state and energy consumption respectively, where $s \in \{0, 1.....C_a - 1, C_a\}$. If s increases, the battery state becomes high. Otherwise, the battery state becomes low. If s equals to the zero, the battery has been fully discharged; if s equals to the C_a, the battery has the full energy units. Let b_x be the probability of x energy units drained from the battery in each time slot. Let $p(s, k)$ be the recovery probability at the s state when k energy units has been drained. When the system is idle, the Markov process moves from low state to high state because of the recovery effect; when the system is busy, it moves from high state to low state as the energy units are drained from the battery. We extend the battery model to multi-cell battery model and present the parameters in Table I. In this model, $p(s_{ij}, k_{ij})$ is used as the recovery probability in the time slot. The expression is shown in (1).

$$p(s_{ij}, k_{ij}) = \begin{cases} A_S e^{-A_N(C_a - s_{ij}) - g(k_{ij})}, & 0 \le k_{ij} \le c_1 \\ A_S e^{-A_N(C_a - s_{ij}) - c_u g(k_{ij})}, & c_u \le k_{ji} \le c_{u+1} \end{cases} \quad (1)$$

In (1), A_S is a constant; A_N and $g(\cdot)$ denote the recovery capability of battery. These parameters can be calculated according to the statistical method [7]. A large value of A_N corresponds to a low recovery capability, while a small value of A_N denotes a high recovery capability. The battery state is related to its nonlinear characteristics, and maximizing s_{ij} is the objective. Besides, k_{ij} represents the energy units that have been consumed. It drives the Markov states to transit and is related to the energy consumption of the tasks execution. Hence, minimizing k_{ij} is the other objective.

III. PROBLEM FORMULATION

In this section, we illustrate the task mapping on the CGRA and build the task model according to the characteristics of CGRA, then formulate the multi-objective optimization problem of energy consumption and battery state.

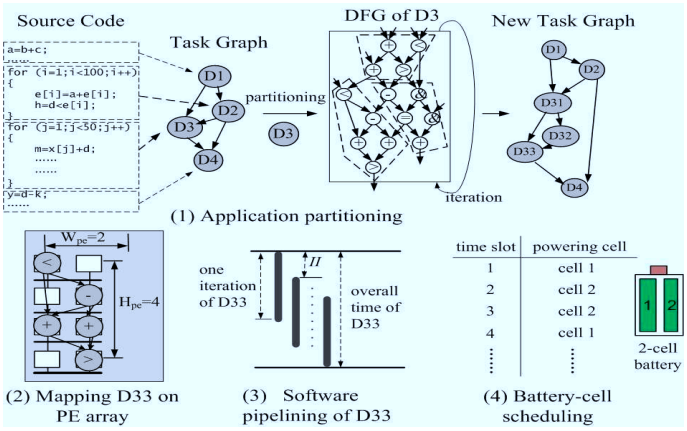


Fig. 3. Task mapping and battey-cell scheduling on CGRA

A. Tasks mapping and battery-cell scheduling on CGRA

In a complex application written in high-level programming language (e.g. C/C++), there are basically three categories of structures: sequential structure, selective structure and looping structure. In selective structure, each branch is composed of sequential and looping structures. Therefore, without loss of generality, we can divide a complex application into two kinds of tasks: sequential logics and loops. For example, as shown in Fig.3(1), $D1$ and $D4$ are sequential logics; $D2$ and $D3$ are loops. For simplification, we regard the sequential logic as loop with only one iteration so that we can use the same methodology to model the energy consumption of both types of tasks. If the size of a task is less than the size of PE array (e.g., $D1$, $D2$, $D4$), it can be mapped to the array without partitioning. Otherwise, it needs to be partitioned into small tasks. For instance, the loop $D3$ is partitioned to three small loops, $D31$, $D32$ and $D33$. Let $S_{iter}(D)$ be the number of operators in a task and $H_{iter}(D)$ be the number of operators in the critical path of the task. In task partitioning, the size of the task should be smaller than the size of PE array, and the critical path of the task should be smaller than the height of PE array. Hence, we have the constraints in equation (2) and (3), where the W_{pe} and H_{pe} denote the width and height of the array. Both sequential logics and loops are then mapped onto the PE array as the Fig.3(2) shown. Especially for loops, they can be executed in a pipeline way to improve the efficiency. As shown in Fig.3(3), II is the initiation interval between the iterations. In each time slot, a different battery-cell is selected to power the CGRA platform as shown in Fig.3(4). The alternative scheduling of battery cells is beneficial since each cell gets time to recover charge during idle periods.

$$S_{iter}(D) \le W_{pe} \times H_{pe} \quad (2)$$

$$H_{iter}(D) \le H_{pe} \quad (3)$$

B. Task model

The procedure of executing a task on CGRA includes four stages: configuring PE array, loading input data from memory, executing task and storing output data into memory.

1) Configuration: Let I_{conf} be the working current of configuration that represents the current consumed when the controller writes configuration words into PE's registers. For a given CGRA, I_{conf} is constant. Let T_{conf} be the configuration

time. It equals to the time of writing the configuration words to all PEs. Like I_{conf}, it is also a constant.

2) Data loading and storing: We denote the current of data loading and storing as I_{dl} and I_{ds}, denote the time of data loading and data storing as T_{dl} and T_{ds}, and $V_{in}(D)$ and $V_{out}(D)$ as the input data volume and output data volume respectively. The T_{dl} and T_{ds} are in direct proportion to $V_{in}(D)$ and $V_{out}(D)$ respectively. The equations (4) and (5) show the relations where t_{dl} and t_{ds} denote the time of loading and storing one data respectively.

$$T_{dl} = t_{dl} \times V_{in}(D) \tag{4}$$
$$T_{ds} = t_{ds} \times V_{out}(D) \tag{5}$$

3) Execution: In this stage, the working current is in direct proportion to the number of activated PEs (N_{pe}). Let I_{exe} be the working current of the PE array. Since different operations need different currents, let $i_{pe}(op)$ be the current of implementing a corresponding operation *op*. The current of executing task D is shown in (6). Let T_{exe} be the executing time of task D. If D is a sequential logic, T_{exe} equals to the critical path length of D. If D is a loop, considering the pipeline execution, T_{exe} is shown in (7) where $H_{iter}(D) \times t_{pe}$ is the time of executing one iteration, ν denotes the number of iterations and t_{pe} denotes the PE executing time.

$$I_{exe} = \sum_{N_{pe}} i_{pe}(op) \tag{6}$$
$$T_{exe} = H_{iter}(D) \times t_{pe} + \mathit{II} \times (\nu - 1) \tag{7}$$

Based on the above discussion, the total execution time $T(D)$ and the average working current $I(D)$ are shown in (8) and (9). If $\nu = 1$, $I(D)$ is the average working current of a sequential logic. Otherwise, it is the current of a loop.

$$T(D) = T_{conf} + T_{dl} + T_{exe} + T_{ds} \tag{8}$$
$$I(D) = \frac{T_{conf} \times I_{conf} + T_{exe} \times I_{exe}}{T(D)} + \frac{T_{dl} \times i_{dl} + T_{ds} \times i_{ds}}{T(D)} \times \nu \tag{9}$$

C. Optimization Problem

Let Q be the number of tasks derived from the application A_p, and let $D(q)$ denote the task q, where $q \in \{1, 2, 3.....Q\}$, and $D(q) = (I_q, t_q, tb_q)$. I_q, t_q, and tb_q denote the execution current, execution time and starting time of the task q respectively. The I_q equals to the $I(D)$ and the t_q equals to the $T(D)$. Let T be the maximum execution time for these Q tasks. The time constraint is shown in (10).

$$\sum_{q=1}^{Q} t_q \le T \quad \text{or} \quad tb_Q + t_Q \le T \tag{10}$$

Let o_{ijq} be the factor of battery-cell scheduling and task execution. It means that the j battery-cell is picked to power the task q in the i time slot, where $i \in \{1, 2, 3, ...T\}$, $o_{ijq} \in \{0, 1\}$. If $\sum_{q=1}^{Q} o_{ijq} = 0$, it means the j battery is idle in the i time slot, and it will recover energy with the specific recovery probability. Let E_c be the intrinsic energy consumption of the circuits, and let $E(D_i)$ be the amount of energy units needed to execute the task in the i time slot. In order to extend the battery runtime, we first minimize the energy consumption. Given Q and T, the first objective is shown in (11).

$$\begin{aligned} Minimize \quad & max(k_{Tj} - k_{1j}), \quad j \in 1....B_n \\ S.t. \quad & \begin{cases} k_{(i+1)j} = k_{ij} + \sum_{q=1}^{Q} o_{ijq}[E(D_i) + E_c] \\ o_{ijq} \in \{1, 0\}, \quad D_i \in \{D(1), D(2), ...D(Q)\} \end{cases} \end{aligned} \tag{11}$$

In (11), $(k_{Tj} - k_{1j})$ denotes the energy consumption of the j battery during time T. We use $\min(\max(\cdot))$ function to ensure that the total energy consumption of all the battery-cells is minimal. Let $k_{(i+1)j}$ be the energy units that have been consumed before the $(i+1)$ time slot. It equals to the sum of k_{ij} and the energy consumption in the i time slot. The energy consumption in the i slot includes the energy units $E(D_i)$ and the intrinsic circuit energy consumption E_c, where the D_i denotes the task number in the time slot i. These two items are depended by the operation o_{ijq}. Besides, we present the second objective in (12) to keep the battery in high state, where $\omega_1 = \sum_{q=1}^{Q} O_{ijq}[E(D_i) + E_c]$, $\omega_2 = \min((s_{ij} + E_r), C_a)$ with $p(s_{ij}, k_{ij})$.

$$\begin{aligned} Maximize \quad & min(s_{(T+1)j}), \quad j \in 1....B_n \\ S.t. \quad & s_{(i+1)j} = \begin{cases} s_{ij} - \omega_1, & \text{if } \sum_{q=1}^{Q} o_{ijq} = 1 \\ \omega_2, & \text{if } \sum_{q=1}^{Q} o_{ijq} = 0 \end{cases} \end{aligned} \tag{12}$$

In (12), $s_{(T+1)j}$ denotes the final state of the j battery after T. We use the $\max(\min(\cdot))$ function to ensure that all the battery-cells sustain the high state after all the Q tasks are executed. If $\sum_{q=1}^{Q} o_{ijq} = 1$, the j battery is selected to power the tasks, and the battery state $s_{(i+1)j}$ in the $(i+1)$ time slot equals that the current state subtract the energy consumption in the i time slot. If $\sum_{q=1}^{Q} o_{ijq} = 0$, the j battery is idle, and the E_r is added to the current state with the recovery probability. Since C_a is the actual maximum energy units the battery can provide, we use the function $\min(\cdot)$ to ensure that the total energy units are no more than C_a.

IV. Joint Task-Mapping and Battery-Scheduling Method

A. System model transformation

We transform the stochastic problem into the deterministic problem to seek the high efficient solution. And let $E_r^{ave}(s_{ij}^{'}, k_{ij}) \triangleq E_r p(s_{ij}, k_{ij})$ be the average recovery energy units of the j battery cell in the i time slot. $s_{ij}^{'}$ is a new description of the battery state, and is presented in (13), where $\omega_3(i) = \min((s_{ij} + (1 - \sum_{q=1}^{Q} o_{ijq}) E_r^{ave}(s_{ij}^{'}, k_{ij}) - \sum_{q=1}^{Q} o_{ijq}[E(D_i) + E_c]), C_a)$. By means of $E_r^{ave}(s_{ij}^{'}, k_{ij})$, we equivalently transform the system model.

$$s_{ij}^{'} = \begin{cases} s_{ij}, & i = 1 \\ \omega_3(i), & i = 2, 3....T \end{cases} \tag{13}$$

Proposition 1: For the j battery in the i time slot, given the initial battery state (s_{1j}, k_{1j}) and O_{ijq}, the $s_{ij}^{'}$ gives a low bound of the average of s_{ij}, namely $s_{ij}^{'} \le ave\{s_{ij}\}$, $1 \le i \le T+1$, $ave\{\cdot\}$ is a function of average.

Proof: In the i time slot, if $\sum_{q=1}^{Q} o_{ijq} = 1$, we have $ave\{s_{(i+1)j}\} - ave\{s_{ij}\} = s^{'}_{(i+1)j} - s^{'}_{ij}$, if the $\sum_{q=1}^{Q} o_{ijl} = 0$, we have the relation as (14) shown

$$ave\{s_{(i+1)j}\} - ave\{s_{ij}\} = E_r \sum s_{ij} p_{re}(s_{ij}, k_{ij}) p(s_{ij}, k_{ij}) \tag{14}$$

where $p_{re}(s_{ij}, k_{ij})$ denotes the probability of battery state (s_{ij}, k_{ij}) in the i time slot. Due to the Jensen' inequality, we have (15) below.

$$\begin{aligned} ave\{s_{(i+1)j}\} &= ave\{s_{ij}\} + E_r \sum s_{ij} p_{re}(s_{ij}, k_{ij}) p(s_{ij}, k_{ij}) \\ &\geq ave\{s_{ij}\} + E_r p(ave\{s_{ij}\}, k_{ij}) \end{aligned} \tag{15}$$

Besides, we have

$$\begin{aligned} s^{'}_{(i+1)j} &= s^{'}_{ij} + E_r^{ave}(s^{'}_{ij}, k_{ij}) \\ &= s^{'}_{ij} + E_r p(s^{'}_{ij}, k_{ij})) \end{aligned} \tag{16}$$

Since the initial battery state (s_{1j}, k_{1j}) is given, we have $s^{'}_{2j} = ave\{s_{2j}\}$, and according to (15) and (16), we have $s^{'}_{ij} \leq ave\{s_{ij}\}$ when $i \geq 3$. □

According to **Proposition 1**, the stochastic optimization problem in (12) is approximately transformed as a deterministic optimization problem in (17).

$$\begin{aligned} Maximize \quad & min(s^{'}_{(T+1)j}), \quad j \in 1....B_n \\ S.t. \quad & s^{'}_{ij} = \begin{cases} s_{ij}, & i = 1 \\ \omega_3(i), & i = 2, 3..T \end{cases} \end{aligned} \tag{17}$$

B. Heuristic algorithm

This heuristic algorithm includes **Algorithm 1** and **Algorithm 2**. We first rank the equation (11) as the primary objective and equation (17) as the secondary objective, and denote $\delta_q (= I_q)$ as the energy efficient factor of task q. Here δ_q is an indicator for tasks assignment and is used to generates the energy efficient array $E_f(m)$. $E_f(m)$ is an index array of the descending sort of δ_q. Since the battery has the ratio capacity effect, the large discharging current and short time duration results in low discharging efficiency. Hence, we define a cost function, $\zeta = \sum_{q=1}^{Q} (I(D_q) * T(D_q))/(\sum_{q=1}^{Q} T(D_q))^2$, to evaluate the partitioning performance. The smaller ζ means that the task partitioning is more energy-efficient.

Algorithm 1 targets partitioning a large task into some small ones. The Fig.4 illustrates the procedure of task partitioning. It first computes the initial connection array (A_r) whose size is $num(A_p) \times num(A_p)$. The $num(A_p)$ denotes the number of nodes in a task. The array element denotes the connection between the node r_1 (depicted by the row number) and node r_2 (depicted by the column number). If $A_r(r_1, r_2) = 1$, it means r_1 is the start point. If $A_r(r_1, r_2) = -1$, it means r_1 is the end point. If $A_r(r_1, r_2) = 0$, it means there is no connection between r_1 and r_2. Then it implement the IBFS (Improved Breadth First Search) method on the A_r to search the optimal cut. If the edge of (r_1, r_2) is cut, the $A_r(r_1, r_2)$ is set to 0. For example, as shown in Fig.4, $A_r(1, 3)$ is set to 0 when the edge of (1, 3) is cut. According to the cuts, the tasks are generated by merging the nodes that are interconnected. When these tasks meet size and time constraints in (2)-(3) and (10) (denoted by $C_{ons}(S, T) == 1$), the ζ is computed. If ζ is smaller than that in the previous search ($\zeta^{'}$), the IBFS is continued until the nodes are all searched. Otherwise, it backtracks to the last change and the IBFS is continued until all the nodes are searched. Finally, we will get the task groups $\phi_{A \to G}$ which is most energy-efficient.

We compute the I_q of each generated task and then we get the efficient array, $E_f(m)$, by sorting the I_qs. According to the efficient array, the **Algorithm 2** first implements the battery-cell scheduling. We traverse the tasks according to the order in $E_f(m)$. And in each time slot, we select the battery-cell which has been drained the least energy units to power the current task. When $m = Q$, the initial scheduling scheme of o_{ijq} is obtained. The initial scheme of o_{ijq} determines the order of battery-cells to be used. Considering the slack of $T - \sum_{q=1}^{Q} t_q$, we can move the idle time to improve charge recovery further. Then we transform the objective (17) into (18).

$$\min(\max(\sum_{i=1}^{T} \sum_{q=1}^{Q} o_{ijq} E(D_i) - \sum_{i=1}^{T} (1 - \sum_{q=1}^{Q} {}_{ijq}) E_r^{ave}(s^{'}_{ij}, k_{ij}))) \tag{18}$$

Algorithm 1 Task partitioning

Input:
A_p, W_{pe}, H_{pe}, T, $\zeta^{'} = 0$, $A_r(r_1, r_2)$(pre-computed);
Output: optimal partition of A_p: $\phi_{A \to G}$

```
1: for r_1 = 1; r_1 <= num(A_p); r_1 ++ do
2:   for r_2 = 1; r_2 <= num(A_p); r_2 ++ do
3:     if A_r(r_1, r_2)!=0 then
4:       tmp = A_r(r_1, r_2), l_1 = 0, l_2 = 0, = 0;
5:       A_r(r_1, r_2)=0, A_r^' =refresh(A_r), flag = 1;
6:       while l_1 <= num(A_p) do
7:         while l_2 <= num(A_p) do
8:           if A^'(l_1, l_2)= 0&&A^'(l_1, l_2) ∩ φ_{A→G} = null then
9:             Node(l_1, l_2) ⟶ φ_{A→G}(q++);
10:          else
11:            if A^'(l_1, l_2)= 0 then
12:              Node(l_1, l_2) ⟶ φ_{A→G};
13:            end if
14:          end if
15:        end while
16:      end while
17:      if C_ons(S, T) ≠ 1 in φ_{A→G} then
18:        flag = 0.
19:      end if
20:      if flag == 1 then
21:        compute the ζ.
22:        if ζ > ζ^' then
23:          A_r(r_1, r_2) = tmp.
24:        else
25:          ζ^' = ζ, A_r = A_r^'.
26:        end if
27:      else
28:        A_r(r_1, r_2) = tmp.
29:      end if
30:    end if
31:  end for
32: end for
33: return φ_{A→G}.
```

Since $\sum_{i=1}^{T} \sum_{q=1}^{Q} o_{ijq} E(D_i)$ is determined by the initial scheme of o_{ijq}, the problem in (18) is equivalent to maximize the $\sum_{i=1}^{T} (1 - \sum_{q=1}^{Q} o_{ijq}) E_r^{ave}(s^{'}_{ij}, k_{ij})$. As shown in the steps between 17 and 24 in the **Algorithm 2**, the idle time slots that satisfy $\sum_{q=1}^{Q} o_{ijq} = 0$ are distributed. According to the equation (1), the $E_r^{ave}(s^{'}_{ij}, k_{ij})$ increases with $s^{'}_{ij}$. Hence, we allocate the idle time slots at the beginning of tasks where

Algorithm 2 Battery cell scheduling

Input:
$\phi_{A\to G}$, Q, $i=1$, $m=1$, $l=1$;
energy efficient array: $E_f(m)$, $m \in \{1, 2...Q\}$;
Output: the final scheduling scheme of o_{ijq};

1: sample the battery status: k_j, $j \in 1...B_n$;
2: sort the k_j in an ascending order to generate battery index $Bs(j)$;
3: **for** $j=1; j <= B_n; j++$ **do**
4: **if** battery $Bs(j)$ is idle **then**
5: choose the battery $Bs(j)$; break;
6: **end if**
7: **end for**
8: **if** $m > Q$ **then**
9: go to step 17;
10: **else**
11: **if** $i < t_{E_f(m)}$ **then**
12: $q = m$; $o_{ijq} = 1$; $i++$; go to step 1;
13: **else**
14: $m++$; go to step 1;
15: **end if**
16: **end if**
17: acquire the initial scheduling scheme of o_{ijq};
18: insert one idle time slot at the beginning of task interval; refresh all o_{ijq};
19: **if** all the idle time slots are inserted **then**
20: go to step 27;
21: **end if**
22: **if** $E_r^{ave}(s_{ij}^{'}, k_{ij}) + s_{ij}^{'} < C_a$ **then**
23: go to step 18;
24: **else**
25: move to next task interval, goto step 18;
26: **end if**
27: **return** The final scheduling scheme of o_{ijq};

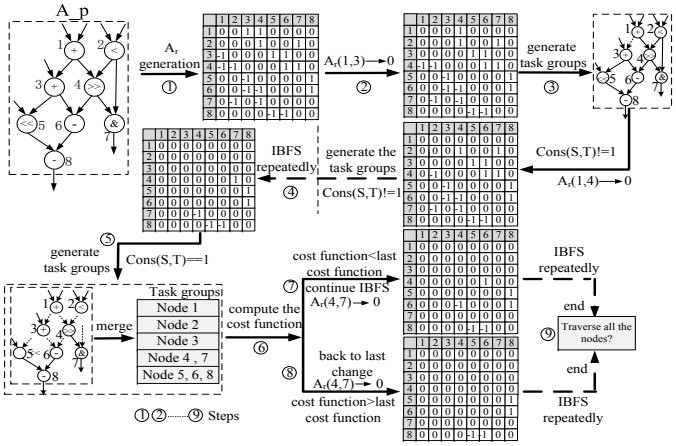


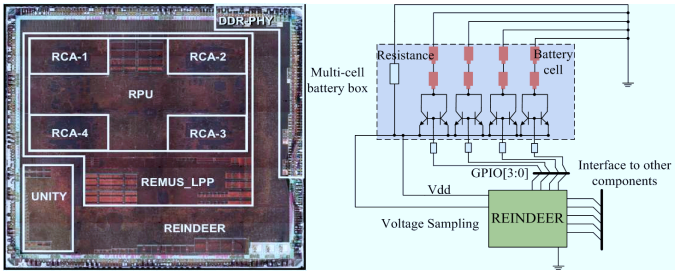
Fig. 4. Task partition illustration (assume $W_{pe} = H_{pe} = 2$)

$s_{ij}^{'}$ is high. Since C_a is the actual energy units the battery can provide, if $E_r^{ave}(s_{ij}^{'}, k_{ij}) + s_{ij}^{'} > C_a$, no energy unit can be recovered. When we distribute the idle time slots, the condition $E_r^{ave}(s_{ij}^{'}, k_{ij}) + s_{ij}^{'} < C_a$ needs to be met.

V. EXPERIMENT

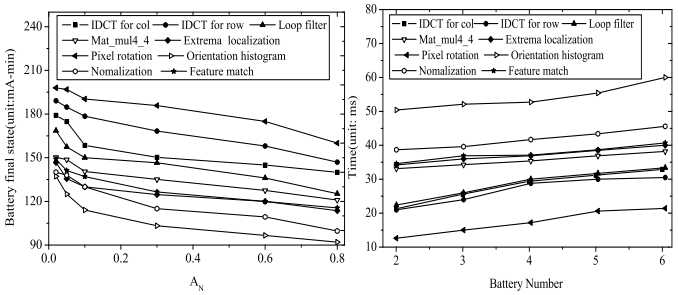
A. Experimental setup

We use a low power reconfigurable processor (REINDEER [10]) as the experimental platform. It contains 4 reconfigurable architectures (RCAs). Each RCA is a 8×8 PE array. Fig.5(a) shows the die photograph. The clock frequency is 100MHz; I_{conf} is 20 mA; T_{conf} is 4 cycles; I_{dl} and I_{ds} are 12mA; t_{dl} and t_{ds} are 2 cycles; t_{pe} is 1 cycle. Besides, we build



(a) Die photograph (b) Multi-cell battery platform

Fig. 5. The experimental platform



(a) A_N effect ($B_n = 2$) (b) Time complexity

Fig. 6. A_N effect and time complexity

an improved multi-cell battery which supports the battery-cell scheduling. As shown in the Fig.5(b), the multiple li-ion cells are organized in a parallel way and can power the REINDEER platform individually. The capacity of each battery-cell is 1200mA-h. The battery-cell switching circuits are made up of the low power CMOS transistors. According to the scheduling algorithm, the processor sends commands to the IO ports (GPIO [3:0]) to select the proper battery-cell.

We adopt a set of benchmarks collected from multimedia (e.g. H.264 decoding) and computer vision (e.g. SIFT: Scale-Invariant Feature Transform) applications for both simulation and physical verification. In previous works, there is no joint optimization method considering task-mapping and battery-cell scheduling simultaneously. For example, Jawad's method [6] only considers battery-aware task scheduling on FPGAs. Peng [7] and Mandal [8] focus on the multi-cell battery scheduling without considering application partition. Jonghee [3] focuses on the task mapping without considering the characteristics of multi-cell battery. Hence, we combine these works to construct two methods (Peng [7]+Jonghee [3] and Mandal [8]+Jonghee [3]) for comparison. These combined methods include both application partition and battery-cell scheduling. In simulation, we examine the time complexity of the proposed method and test the effects of different A_N that implies the variation of nonlinear effect. Fig.6 shows the simulation results. In the physical verification, we evaluate the performance under different benchmarks and different number of battery-cells. Fig.7 and Fig.8 show the results of physical verification.

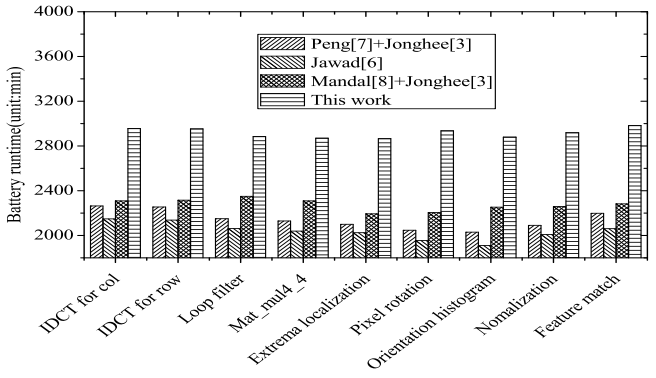


Fig. 7. Battery runtime for different benchmark examples ($B_n = 2$)

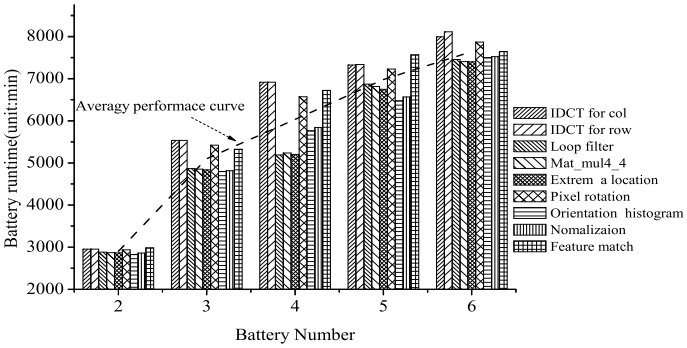


Fig. 8. Battery runtime for the different number of battery-cell

B. Results Comparison

As shown in the Fig.6(a), with A_N ranges from the 0.02 to 0.8, the final battery state (i.e. the residual energy) decreases. This is because the recovery effect is more significant when the A_N tends to 0. These results show that the proposed method is effective no matter the battery types (with different A_N) are. Especially for the battery with low A_N, the improvement of battery runtime is more obvious. The Fig.6(b) shows the time complexity of proposed heuristic algorithm. In **Algorithm 1**, the IBFS method costs the most time. Since the IBFS focuses on the connection point pairs, the time complexity is $O(N^2)$, where N is the number of nodes in DFG need to be partitioned. The core part of **Algorithm 2** is the traversing of battery-cells. Therefore the time complexity is $O(M^2/2)$ where M is the number of battery-cells. Since the number of battery-cells is usually small, the total time increases steadily as shown in Fig.6(b) which fully proves this heuristic solution is efficient.

In the Fig.7, for all benchmark examples, the proposed method achieves the average 26.6%, 30.1%, 21.97% improvement on the battery runtime over the method Peng [7]+Jonghee [3], Jawad [6] and Mandal [8]+Jonghee [3] respectively. The average improvement is 26.2%. These results strongly show that the joint task-mapping and battery-scheduling method is better than either single optimization method. The improvement is in three aspects. First, the task model of our method considers CGRA's special operation model including configuration, data loading/storing and execution which is more precise than other methods. Second, task partitioning in our method is energy-aware which uses time averaging working current as partitioning metric. Third, the battery-cell scheduling leverages the recovery effect greatly resulting in high residual battery capacity.

In the Fig.8, with the number of battery-cells varies from two to six, the battery runtime increases largely. The extension of battery runtime is mainly benefited from the increasing of battery capacity and the contribution of our method. With the increase of the number of battery-cells, the battery achieves more performance improvement since the battery-cell scheduling provides more idle time for each cell to recover the energy units. As shown in the Fig.8, the average performance curve increases quickly when the battery-cell number increases from two to four, and the curve tends to increase slowly when the number of battery-cells increases from four to six. This is because the impact of recovery effect decays with time. Although each cell has more idle time when the number of battery-cells increases, the energy cannot be recovered with the same ratio as the idle time increases.

VI. Conclusions

In this work, considering the energy consumption and battery state as a whole, we propose an efficient way of extending the battery runtime on the multi-cell battery powered CGRA. We propose a joint task-mapping and battery-scheduling method based on energy consumption and battery state model. The experimental results show that the proposed method outperforms the state-of-the-art methods in terms of improving battery runtime.

References

[1] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. M. Rabaey, "A 1-v heterogeneous reconfigurable dsp ic for wireless baseband digital signal processing," *Solid-State Circuits, IEEE Journal of*, vol. 35, no. 11, pp. 1697–1704, 2000.

[2] H.-E. Kim, J.-S. Yoon, K.-D. Hwang, Y.-J. Kim, J.-S. Park, and L.-S. Kim, "A reconfigurable heterogeneous multimedia processor for ic-stacking on si-interposer," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 4, pp. 589–604, 2012.

[3] J. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures," *Very Large Scale Integration Systems, IEEE Transactions on*, vol. 17, no. 11, pp. 1565–1578, 2009.

[4] J. Luo and N. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 444–449.

[5] V. Rao, N. Navet, G. Singhal, A. Kumar, and G. Visweswaran, "Battery aware dynamic scheduling for periodic task graphs," in *Parallel and Distributed Processing Symposium, 2006*. IEEE, 2006, pp. 8–pp.

[6] J. Khan and R. Vemuri, "Energy management for battery-powered reconfigurable computing platforms," *Very Large Scale Integration Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 135–147, 2006.

[7] P. Ouyang, Y. Shouyi, L. Leibo, and W. Shaojun, "Multi-battery scheduling for battery-powered dvs systems," *IEICE transactions on communications*, vol. 95, no. 7, pp. 2278–2285, 2012.

[8] S. Mandal, P. Bhojwani, S. Mohanty, and R. Mahapatra, "Intellbatt: Towards smarter battery design," in *Design Automation Conference, 2008*, 2008, pp. 872–877.

[9] C. Chiasserini and R. Rao, "Improving battery performance by using traffic shaping techniques," *Selected Areas in Communications, IEEE Journal on*, vol. 19, no. 7, pp. 1385–1394, 2001.

[10] M. Zhu, L. Liu, S. Yin, Y. Wang, W. Wang, and S. Wei, "A reconfigurable multi-processor soc for media applications," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 2011–2014.