Algorithms for Bi-objective Multiple-choice Hardware/Software Partitioning

Wenjun Shi[a], Jigang Wu[b,*], Siew-kei Lam[c], Thambipillai Srikanthan[c]

[a]*School of Computer Science and Software Engineering, Tianjin Polytechnic University, Tianjin 300387, China*
[b]*School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China*
[c]*School of Computer Engineering, Nanyang Technological University, Singapore 639798*

Abstract

This paper proposes three algorithms for multiple-choice hardware-software partitioning with the objectives: minimizing execution time and power consumption, while meeting the area constraint. A heuristic algorithm is proposed to rapidly generate an approximate solution. A tabu search algorithm is customized to refine the heuristic solution. Also, a dynamic programming algorithm is proposed to calculate the exact solution. Simulation results show that the heuristic method produces results that are very close to the exact ones, which can be further refined by tabu search to the solutions with an error of no more than 1.5% for all cases considered in this paper.

Keywords: Multiple choice, hardware/software partitioning, bi-objective, power consumption, algorithm

1. Introduction

Hardware/software (HW/SW) partitioning plays an essential role in embedded systems design to decide which components (tasks) in an application should be mapped to software and hardware. Hardware implementation is preferred when performance and power is a critical concern, however they are usually achievable with significant cost. On the other hand, software is relatively cheap and is preferred when programmability is important. However, pure software implementation are usually unable to meet the timing constraints and power budgets of real-time systems. Therefore, typical embedded systems incorporates a mix of hardware-software components to meet the various conflicting constraints of speed, power and cost.

The historical and primary work in the HW/SW partitioning problem can be found in [1]. Traditional approaches include hardware-oriented and software-oriented methods. Hardware-oriented approach starts

*Corresponding author
Email address: asjgwucn@outlook.com (Jigang Wu)

with a complete hardware solution and iteratively moves parts of the system to the software as long as the performance constraints are fulfilled [2][3]. The latter approach starts with a software program and iteratively moves pieces to hardware to improve speed until the performance of the final system meets the given constraint [4][5]. Many approaches emphasize on the algorithmic aspects since the HW/SW partitioning has been shown to be NP-hard for most cases[6]. Thus, simulated annealing algorithms [4][7], dynamic programming algorithms [8][9], integer programming approaches [10][11], genetic algorithms [12][13] and particle swarm optimization [14] are generally utilized to perform the system partitioning and hardware exploration. In addition, some heuristic approaches are introduced in [15][16] for HW/SW partitioning. The work in [17] compared the genetic search, simulated annealing, and tabu search algorithms for solving the partitioning problem and showed the superiority of the tabu search approach. The comparisons of genetic search and simulated annealing algorithms can be found in [18]. While most of these approaches can work well within their respective co-design environments, it is impossible to perform a comprehensive comparison of all the existing approaches due to the large incompatibility in their co-design environments and the lack of proper benchmarks [18]. In our previous work [19][20], the HW/SW partitioning problem is transformed into a variation of knapsack problem. A theoretical approach to find the global optimum of an NP-hard model was proposed in [21].

For a given application,we focus on its 'hot path' which consists of the executed components with high frequency. Hence, the HW/SW partitioning for the whole application can be approximately solved by partitioning the selected hot path. It is fact that larger hardware area may provide higher speed implementation for a given component due to the potential of parallel execution in hardware. Hence, each component of the application has multiple-choice in the different ways of hardware implementation, according to the different hardware areas assigned to it.

Due to the increasing complexity of the applications and growing importance in meeting multiple design constraints, there is a need for hardware-software partitioning algorithms that can produce multiple-objectives and multiple-choice solutions. The challenge in solving for multi-objective problem stem from the conflicting design goals/constraints, while the motivation of solving for multi-choice solutions arise from the need to consider many different but feasible hardware implementation options of a single task. This is a significantly challenging problem as each implementation option has different speed, power and cost attributes.

In our previous work [22], the computing model of multi-choice HW/SW partitioning problem is given and three algorithms are proposed to minimize the execution time of all the given components under the

limited area. This model is very limited as it only considers the execution time and hardware area. Power consumption which is an important design criteria in embedded systems today is ignored.

In this paper, we first formulate the multi-choice HW/SW partitioning problem for minimizing the execution time and power consumption under the constraint of hardware area. The introduction of an additional design objective (i.e. power) increases the complexity of the problem. In particular, this now becomes a bi-objective multi-choice HW/SW partitioning problem and we have reformulated the problem as a multi-objective knapsack problem. This enabled us to devise a heuristic approach to rapidly generate high quality solutions. The solutions can be further refined with tabu search. We have also devised an exact algorithm based on dynamic programming to solve the multi-objective multi-choice problem. Results show that the approximate methods can produce solutions which marginally differs from the exact solution.

Our algorithms can be used in the emerging applications for electric vehicles. We have previously demonstrated the need for hardware-software partitioning of the Extended Kalman Filter for motor control in electric vehicle applications [23]. While it is possible to implement the entire motor control loop in reconfigurable hardware, a typical design will require certain amount of existing and certified motor control software. Our investigation revealed that the communication latencies between the processor and hardware accelerator will impact the hardware implementation choice. In addition, performance and power consumption are essential design constraints in electric vehicle applications.

The rest of the paper is organized as follows: In section 2, we present the computing model and formally describe the bi-objective multi-choice HW/SW partitioning problem. In section 3, we introduce the proposed algorithms. In section 4, we show the experimental results. Finally, we conclude our work in the last section.

2. Computing Models and Formulations

This section describes the bi-objective multi-choice HW/SW partitioning problem, which is the focus of the paper. A hot path of an application consists of a sequence of n blocks, $\mathcal{B} = \{B_1, B_2, \cdots, B_n\}$, where each block can be implemented in software or hardware. Note that the blocks are part of the hot path, they cannot be implemented in parallel. However each block can be realized in hardware in different ways by exploiting intra-block parallelism, whereby each implementation option may incur a different hardware area. The goal of HW/SW partitioning is to decide which blocks should be implemented in hardware and which in software to meet certain design objectives based on the given area constraint.

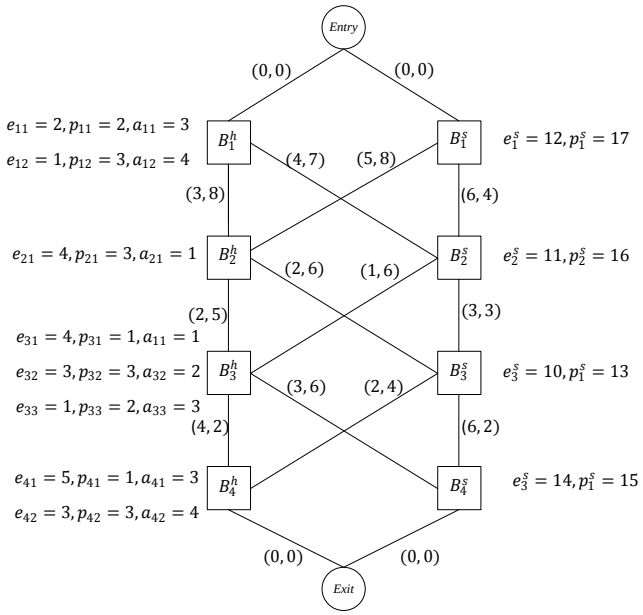


Figure 1: Computing model.

Figure 1 shows an simple example of the computing model. The following notations are utilized to formulate the partitioning problem.

- e_i^s (p_i^s) denotes the execution time (power consumption) for implementing B_i in software, $1 \leq i \leq n$.
- r_i denotes the number of the implementation ways of B_i in hardware, $1 \leq i \leq n$.
- a_{ij} denotes the area penalty of implementing B_i in hardware with the way j, where $1 \leq j \leq r_i$ and $1 \leq i \leq n$.
- e_{ij} (p_{ij}) denotes the execution time (power consumption) of B_i in hardware with area a_{ij}, $1 \leq i \leq n$, $1 \leq j \leq r_i$.
- t_i^{sh} (t_i^{ss}) denotes the communication time between B_i and B_{i+1} if B_i is assigned to software and B_{i+1} is assigned to hardware (software), $1 \leq i < n$.
- t_i^{hh} (t_i^{hs}) denotes the communication time between B_i and B_{i+1} if B_i is assigned to hardware and B_{i+1} is assigned to hardware (software), $1 \leq i < n$.
- p_i^{sh} (p_i^{ss}) denotes the communication power consumption between B_i and B_{i+1} if B_i is assigned to software and B_{i+1} is assigned to hardware (software), $1 \leq i < n$.

- p_i^{hh} (p_i^{hs}) denotes the communication power consumption between B_i and B_{i+1} if B_i is assigned to hardware and B_{i+1} is assigned to hardware (software), $1 \leq i < n$.

The communication time and communication power consumption of each hardware block with its neighbor(s) is the same regardless of how that are implemented in hardware. If we regard the software realization way as a special case in hardware, then the block B_i has r_i+1 different ways of implementation. Also, we set a_{i0} to 0, e_{i0} to e_i^s and p_{i0} to p_i^s, respectively, corresponding to the software implementation of block B_i. Without loss of generality, we assume $a_{ij} < a_{ij'}$ and $e_{ij} \geq e_{ij'}$ for $0 \leq j < j' \leq r_i$ and $p_{ij} < p_{i0}$ for $1 \leq j \leq r_i$ for each block B_i, throughout this paper. This assumption implies that hardware implementation with larger area, will lead to faster execution. As for the power consumption,we assume that the power consumption of any hardware implementation way is less than it in software, due to the diversity of the system structures.

Note that power consumption is not taken into account in the computing model reported in [22]. We now extend it. B_i^s indicates that block B_i is implemented in software and B_i^h indicates that block B_i is implemented in hardware. The pair of ordinal numbers associated with the lines connecting the two blocks denote the communication time and power consumption respectively.

In Figure 1, there are 4 blocks to be partitioned. Now we take B_1 as an example to explain the proposed model. B_1 has three implemented ways, one is in software and the other two are in hardware. If B_1 is realized in software, its execution time e_1^s is 12 and the execution power consumption p_1^s is 17. When B_2 is just implemented in software, the communication time t_1^{ss} and power consumption p_1^{ss} are 6 and 4, respectively. If B_2 is implemented in hardware, the communication time t_1^{sh} and power consumption p_1^{sh} are 5 and 8, respectively. If B_1 selects the first way in hardware, its execution time e_{11} becomes 2, the execution power consumption p_{11} turn into 2, and the area penalty of implementing B_1 with the way 1, i.e., a_{11}, is 3. When B_2 is just implemented in software, the communication time t_1^{hs} and power consumption p_1^{hs} are 4 and 7, respectively. If B_2 is implemented in hardware, the communication time t_1^{hh} and power consumption p_1^{hh} are 3 and 8, respectively. If B_1 selects the second way in hardware, its execution time e_{12} becomes 1, the execution power consumption p_{12} turns into 3, the area penalty a_{12} of implementing B_1 with the way 1 is 4. It is worthy to note that the communication time and power consumption are the same with the first way in hardware at this time.

Each block can be realized in several ways (implementation options), but only one way will be eventually selected. The binary variable $x_{ij} \in \{1, 0\}$, where $x_{ij} = 1$ ($x_{ij} = 0$) denotes that B_i is (not) implemented in j-th way for $j = 0, 1, \cdots, r_i$. We can conclude that $\sum_{j=0}^{r_i} x_{ij} \equiv 1$ because only one way is

allowed to implement block B_i, and

$$x_{i0} = 1 - \sum_{j=1}^{r_i} x_{ij}. \qquad (1)$$

$(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ indicates a feasible solution of the bi-objective multiple choice HW/SW partitioning problem, where $\mathbf{x}_i = (x_{i0}, x_{i1}, \cdots, x_{ir_i})$ reflects the implementation way of block B_i. $E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ indicates the corresponding execution time of the solution, that includes the inherent communication overhead. $P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ indicates the corresponding power consumption. $T_i(\mathbf{x}_i, \mathbf{x}_{i+1})$ indicates the communication time and $P_i(\mathbf{x}_i, \mathbf{x}_{i+1})$ indicates the communication power consumption between B_i and B_{i+1}, where $1 \leq i \leq n-1$. So the execution time can be formalized as:

$$E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \sum_{i=1}^{n} (e_i^s x_{i0} + \sum_{j=1}^{r_i} e_{ij} x_{ij}) + \sum_{i=1}^{n-1} T_i(\mathbf{x}_i, \mathbf{x}_{i+1}),$$

and $T_i(\mathbf{x}_i, \mathbf{x}_{i+1})$ is calculated by:

$$\begin{aligned} T_i(\mathbf{x}_i, \mathbf{x}_{i+1}) = {} & x_{i0} \cdot x_{i+1,0} \cdot t_i^{ss} + x_{i0} \cdot (1 - x_{i+1,0}) \cdot t_i^{sh} \\ & + (1 - x_{i0}) \cdot x_{i+1,0} \cdot t_i^{hs} + (1 - x_{i0}) \cdot (1 - x_{i+1,0}) \cdot t_i^{hh}. \end{aligned}$$

The power consumption can be formalized as:

$$P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \sum_{i=1}^{n} (p_i^s x_{i0} + \sum_{j=1}^{r_i} p_{ij} x_{ij}) + \sum_{i=1}^{n-1} P_i(\mathbf{x}_i, \mathbf{x}_{i+1}), \qquad \text{and}$$

$$\begin{aligned} P_i(\mathbf{x}_i, \mathbf{x}_{i+1}) = {} & x_{i0} \cdot x_{i+1,0} \cdot p_i^{ss} + x_{i0} \cdot (1 - x_{i+1,0}) \cdot p_i^{sh} \\ & + (1 - x_{i0}) \cdot x_{i+1,0} \cdot p_i^{hs} + (1 - x_{i0}) \cdot (1 - x_{i+1,0}) \cdot p_i^{hh}. \end{aligned}$$

From (1), we conclude

$$\begin{aligned} E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) &= \sum_{i=1}^{n} (e_i^s (1 - \sum_{j=1}^{r_i} x_{ij}) + \sum_{j=1}^{r_i} e_{ij} x_{ij}) + \sum_{i=1}^{n-1} T_i(\mathbf{x}_i, \mathbf{x}_{i+1}) \\ &= \sum_{i=1}^{n} e_i^s - \sum_{i=1}^{n} \sum_{j=1}^{r_i} (e_i^s - e_{ij}) x_{ij} + \sum_{i=1}^{n-1} T_i(\mathbf{x}_i, \mathbf{x}_{i+1}). \qquad (2) \end{aligned}$$

$$P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \sum_{i=1}^{n}(p_i^s(1 - \sum_{j=1}^{r_i} x_{ij}) + \sum_{j=1}^{r_i} p_{ij}x_{ij}) + \sum_{i=1}^{n-1} P_i(\mathbf{x}_i, \mathbf{x}_{i+1})$$
$$= \sum_{i=1}^{n} p_i^s - \sum_{i=1}^{n}\sum_{j=1}^{r_i}(p_i^s - p_{ij})x_{ij} + \sum_{i=1}^{n-1} P_i(\mathbf{x}_i, \mathbf{x}_{i+1}). \qquad (3)$$

Given available hardware area A, the bi-objective multiple-choice HW/SW partitioning problem discussed in this paper can be modeled as the following non-linear minimization problem $\mathcal{P}$:

$$\mathcal{P}: \begin{cases} \text{minimize} & E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \\ & P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \\ \text{subject to} & \sum\limits_{i=1}^{n}\sum\limits_{j=0}^{r_i} a_{ij}x_{ij} \le A, \\ & \sum\limits_{j=0}^{r_i} x_{ij} = 1 \ \text{ and } \ x_{ij} \in \{0,1\}, \\ & i = 1, 2, \cdots, n, \ \ j = 0, 1, \cdots, r_i. \end{cases}$$

Like the general multi-objective programming, the two objective functions discussed this paper are also conflicting. This is because when total time is minimal, the total power consumption is not necessarily minimal, vice versa. Deeper reason lies in the fact that the execution time for the same block decreases with the area augmentation, but the power does not necessarily follow the same pattern.

First, we introduce the multi-objective knapsack problem. Consider an optimization problem with t objective functions $V_1, \ldots, V_t$, and let I be an instance of this problem. If S is a feasible solution, then $V_k(S)$ denotes the value of S with respect to the k-th objective function, $1 \le k \le t$. Without loss of generality we assume that all objectives are maximization criteria. A feasible solution S_1 weakly dominates a feasible solution S_2 if

$$V_k(S_1) \ge V_k(S_2), 1 \le k \le t.$$

A weakly domination solution S_1 even dominates solution S_2 if at least one of the inequalities is strict. A feasible solution S is *efficient* or *Pareto optimal* if there is no other feasible solution which dominates S. The set $\mathcal{R}$ of efficient solutions for I is called *Pareto frontier*.

The multi-objective multi-choice knapsack problem (MOMCKP) is obtained from the classical multi-choice knapsack problem by introducing t profit values instead of one for every item. More precisely, an

instance I of MOMCKP consists of n classes of items $S = \{1, 2, \ldots, n\}$ and a knapsack with capacity C. Every class has at least one item, and every item in a class is different. Each item in class i has t profits p_{ik} and a weight w_{ik}, where $k = 1, 2, \ldots, t$. The MOMCKP is aimed at finding a subset of all items which maximizes t profits $\sum_{i \in S'} p_{ik}, k = 1, 2, \ldots, t$.

The problem $\mathcal{P}$ is an extended bi-objective multi-choice knapsack problem. Bi-objective multi-choice knapsack problem is a special case of multi-objective knapsack problems which considers two objectives. As the single objective case has been proven to be NP-complete, we can conclude that $\mathcal{P}$ is also NP-complete because the number of pareto-optimal solutions can grow exponentially with the number of items in the knapsack. Generally only one partitioning scheme, i.e. one feasible solution of the Pareto frontier is required.

A solution of a bi-objective knapsack problem is called supported if it can be found through weighted sum scalarization (*wss*), i.e. if it is a convex combination of the two objective functions. Otherwise it is non-supported [24]. This supported solution is one solution of the Pareto frontier. Through this *wss* method, the bi-objective problem is transformed into a single objective problem. We can formulate the $\mathcal{P}$ by following single objective multi-choice knapsack problem which has been shown to be NP-hard:

$$\mathcal{P}' : \begin{cases} \text{minimize} & \alpha E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \\ & +(1-\alpha) P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \\ \text{subject to} & \sum_{i=1}^{n} \sum_{j=0}^{r_i} a_{ij} x_{ij} \leq A, \\ & \sum_{j=0}^{r_i} x_{ij} = 1 \ \text{ and } \ x_{ij} \in \{0, 1\}, \\ & i = 1, 2, \cdots, n, \ \ j = 0, 1, \cdots, r_i, \\ & 0 \leq \alpha \leq 1. \end{cases}$$

Let $B(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \alpha E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) + (1-\alpha) P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$. Thus α is the weight of execution time, and $(1-\alpha)$ is the weight of power consumption. The solution of $\mathcal{P}'$ is the supported solution under this weight trade-off. If execution time is of higher concern, we can set $\alpha > 1 - \alpha$, that is $\alpha > 0.5$. In a similar way, if the user attaches more importance to power consumption, we can set $\alpha < 0.5$. When the two objectives are of equal importance, $\alpha = 0.5$.

3. Proposed Algorithms

3.1. Heuristic Algorithm

The partitioning problem $\mathcal{P}'$ is closely related to multiple-choice knapsack problem (MCKP). Let block B_i in the problem $\mathcal{P}'$ correspond to the item i in MCKP. Also a_{ij} and A in $\mathcal{P}'$ correspond to w_{ij} and C in MCKP, respectively. The problem $\mathcal{P}'$can be reduced to MCKP when communication time and power consumption are not taken into account. This is because, when all communication time and power consumption values are set to 0, the two objective functions can be reduced from (2) and (3) to:

$$E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \sum_{i=1}^{n} e_i^s - \sum_{i=1}^{n} \sum_{j=1}^{r_i} (e_i^s - e_{ij}) x_{ij}.$$

$$P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) = \sum_{i=1}^{n} p_i^s - \sum_{i=1}^{n} \sum_{j=1}^{r_i} (p_i^s - p_{ij}) x_{ij}.$$

As *min* $E(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ *and* $P(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ are equivalent to

$$max \sum_{i=1}^{n} \sum_{j=1}^{r_i} (e_i^s - e_{ij}) x_{ij} \text{ and } \sum_{i=1}^{n} \sum_{j=1}^{r_i} (p_i^s - p_{ij}) x_{ij},$$

the problem $\mathcal{P}'$ can be reduced to the following multiple-choice knapsack problem,

$$\begin{cases} \text{maximize} & \alpha \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{r_i} (e_i^s - e_{ij}) \cdot x_{ij} \\ & +(1-\alpha) \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{r_i} (p_i^s - p_{ij}) x_{ij} \\ \text{subject to} & \sum\limits_{i=1}^{n} \sum\limits_{j=0}^{r_i} a_{ij} x_{ij} \leq A, \\ & \sum\limits_{j=0}^{r_i} x_{ij} = 1 \text{ and } x_{ij} \in \{0,1\}, \\ & i = 1, 2, \cdots, n, \ \ j = 0, 1, \cdots, r_i, \\ & 0 \leq \alpha \leq 1. \end{cases}$$

The objective function of $\mathcal{P}'$ is nonlinear while it is not for MCKP. This is the only difference between them. So the problem $\mathcal{P}'$ is also NP-hard.

We adopt the approach for calculating communication time in [22]. The communication time profit for moving B_i from software to hardware, denoted as δ_i, is defined as

$$\delta_i = ct\_sw(B_i) - ct\_hw(B_i),$$

where, $ct\_sw(B_i)$ ($ct\_hw(B_i)$) indicates the communication time of B_i to its neighbor(s) when B_i is assigned to software (hardware). The power consumption due to communication profit can be calculated

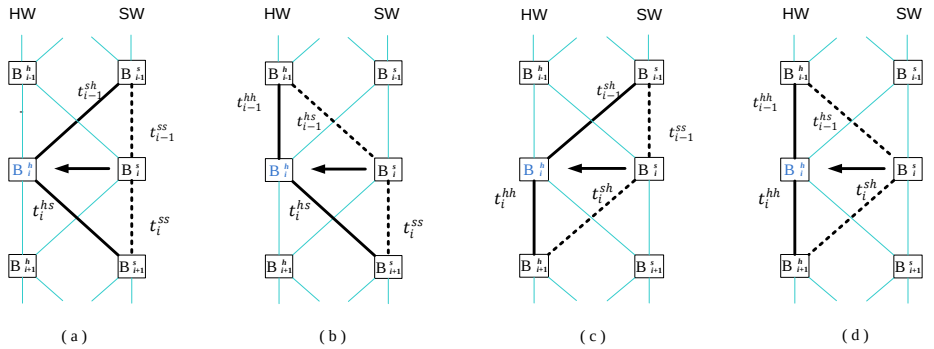


Figure 2: Calculation of δ_i and η_i.

in a similar way as the communication time. The communication power consumption profit for moving B_i from software to hardware, denoted as η_i, is defined as

$$\eta_i = cp\_sw(B_i) - cp\_hw(B_i),$$

where, $cp\_sw(B_i)$ ($cp\_hw(B_i)$) indicates the communication power consumption of B_i to its neighbor(s) when B_i is assigned to software (hardware). Figure 2 shows all the cases for the communication time when B_i is moved from software to hardware.Specifically, δ_i and η_i can be calculated as follows:

- case (a): $\delta_i = t_{i-1}^{ss} + t_i^{ss} - t_{i-1}^{sh} - t_i^{hs}$, $\eta_i = p_{i-1}^{ss} + p_i^{ss} - p_{i-1}^{sh} - p_i^{hs}$;
- case (b): $\delta_i = t_{i-1}^{hs} + t_i^{ss} - t_{i-1}^{hh} - t_i^{hs}$, $\eta_i = p_{i-1}^{hs} + p_i^{ss} - p_{i-1}^{hh} - p_i^{hs}$;
- case (c): $\delta_i = t_{i-1}^{ss} + t_i^{sh} - t_{i-1}^{sh} - t_i^{hh}$, $\eta_i = p_{i-1}^{ss} + p_i^{sh} - p_{i-1}^{sh} - p_i^{hh}$;
- case (d): $\delta_i = t_{i-1}^{hs} + t_i^{sh} - t_{i-1}^{hh} - t_i^{hh}$, $\eta_i = p_{i-1}^{hs} + p_i^{sh} - p_{i-1}^{hh} - p_i^{hh}$.

As each block has $r_i + 1$ different ways to be implemented, we evaluate each way via profit-to-area ratio. If the block B_i is in its k^{th} way and is being considered to be changed to j^{th} way, where $j > k$, the profit-to-area ratio for this case, denoted as $\lambda_i(k, j)$ calculated by

$$\frac{\alpha(e_{ik} - e_{ij} + \delta(i,k)) + (1-\alpha)(p_{ik} - p_{ij} + \eta(i,k))}{a_{ij} - a_{ik}} \tag{4}$$

where $\delta(i,k) = \begin{cases} \delta_i & \text{if } k = 0, \\ 0 & \text{otherwise.} \end{cases}$ *and* $\eta(i,k) = \begin{cases} \eta_i & \text{if } k = 0, \\ 0 & \text{otherwise.} \end{cases}$

When $k \neq 0$, $\delta = 0, \eta = 0$, the implementation way of block B_i changes within hardware. The communication time and the communication power consumption are not changed.

Our heuristic algorithm is based on finding the largest profit-to-area ratio in each iteration. Initially we assume all blocks are realized in software. δ_is and η_is are computed as shown in figure 2(a). The profit-to-area ratios are calculated for all blocks together with their different implementation ways. Then the block with the maximum profit-to-area ratio is selected. The selected block will be considered to be assigned to hardware if there is sufficient hardware area to implement it. All neighboring blocks of the selected block update the corresponding communication profits, calculated as shown in figure 2(b), 2(c) and 2(d). Selecting the block with maximum profit-to-area ratio and updating the communication profits are repeated on the remaining blocks with different ways of implementations, until no block and no implementation way can fit the remaining hardware area.

The proposed heuristic algorithm (HEA) is outlined below.

Input: Source data for the blocks $B_1, B_2, \cdots, B_n$;

Output: The heuristic solution : $solution[1:n]$. /* $solution[i] = j$ indicates block B_i is implemented in j-th way. */

Algorithm HEA

begin

1 $residual\_area := A$;

2 **for** $i := 1$ **to** n **do** /* initializing */

$solution(i) := 0, \quad k_i := 0$;

Calculate $\lambda_i(k_i, j)$ using (4) for $0 < j \leq r_i$;

end of for;

3 **repeat**

3.1 $\lambda_{i'}(k_{i'}, j') := \max\{\lambda_i(k_i, j) \mid 1 \leq i \leq n \text{ and } 0 < j \leq r_i\}$;

3.2 **if** $(a_{i'j'} \leq residual\_area)$ **and** $(\lambda_{i'}(k_{i'}, j') > 0)$

then begin/* implement block $B_{i'}$ in j'-th way, update */

$solution(i') := j'$; /* Update $solution(i')$ */

$k_{i'} := solution(i')$;

Update profit-to-area ratios for block $B_{i'}$ and its neighbor(s);

$residual\_area := residual\_area - a_{i'j'}$;

end;

until (no block fits for the residual hardware area);

4 **Output** $solution[1:n]$;

end.

Steps 1 and 2 run in $O(n \cdot r)$ time. Step 3 also run in $O(n \cdot r)$ to select the maximum profit-to-area ratio and update the communication profits. Therefore, we can conclude that the time complexity of HA

is bounded by $O(n \cdot r)$.

3.2. Tabu Search Algorithm

Tabu search algorithm is a local search algorithm that can be used for effectively solving the NP-hard problems[25]. It utilizes seven primary parameters: stopping rule, tabu list, frequency-base memory, aspiration criteria, tabu conditions, recency-based memory and neighborhoods. We use tabu search algorithm (TSA) to refine the heuristic solution generated by HEA.

The problem $\mathcal{P}$ can be resolved using tabu search as follow. The approximate solution of HEA can be represented by a string of numbers. For example in the following string '145035231', '2' indicates that block B_7 is realized in hardware using the second implementation option, while '0' indicates that the corresponding blocks are realized in software. The customized TSA starts with the initial solution generated by HEA. In each iteration, we generate a fixed number of the neighbors (i.e. neighborhood size) for the current solution. Each neighbor, denoted as X_{neib}, is generated by randomly changing two bits (where each bit is associated with a block). If the generated neighbor fails to meet the area constraint, it is discarded and new attempts are made to generate a new neighbor. This is repeated for a fixed number of times. If all attempts fail, only one block is changed to obtain a feasible solution instead of two.

In each iteration of the TSA, we determine which neighbor is the most suitable based on recency-based memory, frequency-based memory and the neighbor solution's value $E(X_{neib})$. The most suitable neighbor, denoted as X_{local}, is selected to replace the local solution. To exploit recency-based memory, the selected neighbors are labeled as *tabu-active*.

In the entire search process, a neighbor solution X_{neib} may enter the tabu list many times. Assume the latest entrance for X_{neib} is in the iteration $iter\_late(X_{neib})$ and the current search is at the iteration $iter\_curr$. The tabu degree of X_{neib}, denoted as $T\_d(X_{neib})$, is defined as

$$T\_d(X_{neib}) = iter\_late(X_{neib}) + tabu\_tenure - iter\_curr.$$

Tabu degree is updated for each neighbor in each iteration. A positive tabu degree of a neighbor implies that the neighbor is tabu-active. On the other hand, a negative one implies that the neighbor is not tabu-active.

In TSA, the solution value is computed to select a good neighbor solution. Let X_{neib} be a neighbor of the local solution X_{local}. It is evident that the smaller the value of $B(X_{neib})$, the better the quality of the neighbor X_{neib} will be. Thus, X_i is better than X_j if $B(X_i) < B(X_j)$. At each iteration, the move

obtained the best solution in the neighborhood is selected, even if this results in a worse solution. A fixed number of iterations is chosen as the stopping rule of the tabu search. The following is the formal description of the tabu search approach.

Input: X_{heur} – The heuristic solution generated by the algorithm HEA;

Output: X_{best} – the best-so-far solution found by tabu search;

Algorithm TSA

/* Tabu Search Algorithm for the problem MCHS. M indicates the fixed number of iterations. q indicates the neighborhood size. */

begin

1 $X_{local} := X_{heur}$, and $X_{best} := X_{heur}$;

2 **for** *iter* := 1 **to** M **do**

begin

2.1 Generate q neighbors of X_{local};

2.2 Compute the degrees and $B(X_{neib})$s of the q neighbors;

2.3 X_{min_neib} := the neighbor with the minimal $B(X_{neib})$;

2.4 **if** $B(X_{neib}) < B(X_{best})$ /* aspiration criterion */

then $X_{local} := X_{min\_neib}$, and $X_{best} := X_{min\_neib}$

else begin

if all q neighbors are tabu-active

then X_{local} := the neighbor with the minimal tabu degree

else reward the neighbor(s) that never tabu-active and

X_{local} := the neighbor with the minimal $B(X_{neib})$;

end

2.5 Update frequency-based memory and recency-based memory;

/*put X_{local} into tabu list*/

end;

end

3.3. Exact Algorithm

In order to evaluate the performance of the algorithms HEA and TSA, we propose a dynamic programming approach for the problem $\mathcal{P}'$, denoted as DPA in this subsection to calculate the optimal solution of the problem $\mathcal{P}'$. The main idea is as follows. Assuming that the optimal HW/SW partitioning for $B_1, B_2, \cdots, B_{i-1}$ has been computed where the utilized hardware area is less than a. We now consider how to partition the blocks $B_1, B_2, \cdots, B_i$ within the available area a. This is achieved by first arriving at all partitioning possibilities based on representing the current block B_i in software or in hardware. The optimal partitioning results is the best value. If B_i is implemented in software, the optimal partitioning for $B_1, B_2, \cdots, B_i$ for the hardware area a is identical to the optimal partitioning for $B_1, B_2, \cdots, B_{i-1}$. If B_i is moved to hardware in j-th way, the optimal partitioning for $B_1, B_2, \cdots, B_i$ can be found by

examining partitioning for the blocks $B_1, B_2, \cdots, B_{i-1}$ for area $a - a_{ij}$. Now we consider the first i blocks $B_1, B_2, \cdots, B_i$, $i \leq n$. Let $B(i, j, a)$ indicate the objective function value by moving B_i to hardware of area a_{ij} and then moving some or all blocks from $B_1, B_2, \cdots, B_{i-1}$ to area $a - a_{ij}$. $B(i, j, a)$ recursively depends on $B(i-1, t, a - a_{ij})$ for $t = 0, 1, \cdots, r_{i-1}$, because B_{i-1} has $r_{i-1} + 1$ possible assignments, and the hardware space has been partially occupied by block B_i.

Let $c_i(j, t)$ indicate the communication time and power consumption in the objective function between blocks B_i and B_{i+1} when B_i is implemented in hardware of area a_{ij} (in j-th way, $0 \leq j \leq r_i$) and B_{i+1} is implemented in hardware of area $a_{i+1,t}$ (in t-th way, $0 \leq t \leq r_{i+1}$). We define $c_i(j, t)$ for $i = 1, 2, \cdots, n-1$, as follows.

$$c_i(j,t) = \begin{cases} \alpha \cdot t_i^{ss} + (1-\alpha) \cdot p_i^{ss}, \text{ for } j = 0 \ \& \ t = 0; \\ \alpha \cdot t_i^{sh} + (1-\alpha) \cdot p_i^{sh}, \text{ for } j = 0 \ \& \ t > 0; \\ \alpha \cdot t_i^{hs} + (1-\alpha) \cdot p_i^{hs}, \text{ for } j > 0 \ \& \ t = 0; \\ \alpha \cdot t_i^{hh} + (1-\alpha) \cdot p_i^{hh}, \text{ for } j > 0 \ \& \ t > 0; \end{cases}$$

Also, let $B\_op(n, A)$ indicate the optimal objective function value achievable by moving some or all the blocks from $B_1, B_2, \cdots, B_n$ to hardware of area A. The recurrent idea as described above can be outlined as the following DPA:

$$\text{DPA}: \begin{cases} B(1,j,a) = \begin{cases} \infty & \text{for } a_{1j} > a, \\ \alpha \cdot e_{1j} + (1-\alpha) \cdot p_{1j} & \text{otherwise,} \end{cases} \\ \qquad \text{for } 0 \leq a \leq A \text{ and } 0 \leq j \leq r_1; \\ B(i,j,a) = \begin{cases} \infty \quad \text{for } a_{ij} > a, \\ \min\limits_{0 \leq t \leq r_{i-1}} \{B(i-1, t, a - a_{ij} + c_{i-1}(t,j) \\ \quad + \alpha \cdot e_{ij} + (1-\alpha) \cdot p_{ij}\} \quad \text{otherwise,} \end{cases} \\ \qquad \text{for } 2 \leq i \leq n-1, \ 0 \leq j \leq r_i \text{ and } 0 \leq a \leq A; \\ B(n,j,A) = \begin{cases} \infty \quad \text{for } a_{nj} > a, \\ \min\limits_{0 \leq t \leq r_{n-1}} \{B(n-1, t, A - a_{nj}) + c_{n-1}(t,j) \\ \quad + \alpha \cdot e_{nj} + (1-\alpha) \cdot p_{nj}\} \quad \text{otherwise,} \end{cases} \\ \qquad \text{for } 0 \leq j \leq r_n; \\ B\_op(n, A) = \min\limits_{0 \leq j \leq r_n} \{B(n, j, A)\} \end{cases}$$

The pseudo-code below is given for the problem with n blocks and a list of trial area $\langle A_0, A_1, \cdots, A_m \rangle$. A_0 is set to 0, as it corresponds to software implementation. A_m is set to the whole hardware size A.

Step 1 is used for the initialization of the first block, that corresponds to the case of $i = 1$ in formula DPA. Each $B(i, j, a)$ is calculated by the nested for-loops (steps 2 and 3), the optimal objective function value $B\_op(n, A_m)$ is calculated by step 4, in which the calculations for backtracking are undertaken. $trace(i, j, a) = l$ means that the block B_{i-1} is implemented in l-th way. The array $solution[1 : n]$ is used to store the solution of partitioning n blocks within the area A_m.

Let $r = \max_{1 \le i \le n}\{r_i\}$. Given n blocks and the list of trial area $\langle A_0, A_1, \cdots, A_m \rangle$, Step 2 runs in $O(n \cdot r^2 \cdot m)$ time, that dominates the computing time of DPA. By similar analysis, we conclude that the space complexity is bounded by $O(n \cdot r \cdot m)$.

Unlike heuristic algorithms, DPA not only depends on the number of the blocks but also on the plot granularity of the given hardware area A. In fact, given n blocks and the list of trial areas $\langle A_1, A_2, \cdots, A_m \rangle$, its complexity is bounded by $O(n \cdot r^2 \cdot m)$ time. But DPA is very limited in dealing with the problem. This is mainly due to the computational and memory requirements up to $O(n \cdot r \cdot m)$ in practice. However, DPA can produce the optimal solution according to the dynamic programming principle of optimality [1]. We utilize it to evaluate the proposed HEA and TSA.

Input: Source data for the blocks $B_1, B_2, \cdots, B_n$;
$A_0, A_1, \cdots, A_m$ – trial hardware area for total area A, where,
$A_0 = 0$ corresponds to software implementation, and $A_m = A$.
Output:The optimal solution stored in $solution[1 : n]$.
Algorithm DPA
begin
1 **for** $j := 0$ **to** r_1 **do** /* initializing for the first block */
  for $a := A_0$ **to** A_m **do**
    if $a_{1,j} \le a$ **then** $B(1, j, a) := \alpha \cdot e_{1j} + (1 - \alpha) \cdot p_{1j}$ **else** $B(1, j, a) := \infty$;
2 **for** $i := 2$ **to** $n - 1$ **do** /* computing for the first $n - 1$ blocks */
  for $j := 0$ **to** r_i **do**
    for $a := A_0$ **to** A_m **do**
    if $a_{ij} \le a$ **then**
    begin
      for $t := 0$ **to** r_{i-1} **do**
        if $a_{ij} + a_{i-1,t} \le a$ **then**
         $temp(t) := B(i - 1, t, a - a_{ij}) + c_{i-1}(t, j) + \alpha \cdot e_{ij} + (1 - \alpha) \cdot p_{ij}$
        else $temp(t) := \infty$;
      end of for;
      Find l such that $temp(l) = \min_{0 \le t \le r_{k-1}} \{temp(t)\}$;
      $B(i, j, a) := temp(l)$ and $trace(i, j, a) := l$;
    end
    else $B(i, j, a) := \infty$ and $trace(i, j, a) := NA$;
3 **for** $j := 0$ **to** r_n **do** /* computing for n-th (last) block */
  if $a_{nj} \le A_m$ **then**
  begin

for $t := 0$ **to** r_{n-1} **do**

if $a_{nj} + a_{n-1,t} \leq A_m$ **then**

$temp(t) := B(n-1, t, A_m - a_{nj}) + c_{n-1}(t, j) + \alpha \cdot e_{nj} + (1-\alpha) \cdot P_{nj}$

else $temp(t) := \infty$;

end of for;

Find l such that $temp(l) = \min_{0 \leq t \leq r_{n-1}} \{temp(t)\}$;

$B(n, j, A_m) := temp(l)$ and $trace(n, j, A_m) := l$;

end

else $B(n, j, A_m) := \infty$;

/*followed by backtracking along the trace for solution.*/

4 Find l such that $B(n, l, A_m) = \min_{0 \leq j \leq r_n} \{B(n, j, A_m)\}$;

$B\_op(n, A_m) := B(n, l, A_m)$, $solution(n) := l$ and $area := A_m$;

for $i := n-1$ **down to** 1 **do**

$solution(i) := trace(i+1, solution(i+1), area)$;

$q := solution(i+1)$ and $area := area - a_{i+1,q}$

end of for;

end

4. Simulation Results

The proposed algorithms HEA, TSA and DPA are simulated in C on a Workstation–Intel Xeon E5-1650, 3.20GHz CPU, 16.00GB of RAM, running Microsoft Windows 7 enterprise operating system. Random instances generated in the similar manner to that used in [6][19][20][22] are utilized in our simulations. In order to make the dimension of time and power consumption comparable in terms of importance, we use the same order of magnitude to generate their value randomly.

- a_{ij} is randomly generated in $[1, 40]$, for $j = 1, 2, \cdots, r_i$. A is set to $\beta \cdot \bar{A}$, where $\bar{A} = \sum_{i=1}^{n} a_{i,r_i}$, for $0 < \beta < 1$. Given integer r, each r_i is randomly generated in $[1, r]$.
- e_i^s, i.e., $e_{i,0}$, is randomly generated in $[1, 100]$ for $i = 1, 2, \cdots, n$.
- e_{ij} is set to $\mu_i \cdot e_{i,j-1}$ for $j = 1, 2, \cdots, r_i$, where μ_i is randomly generated in $(0, 1)$, for $i = 1, 2, \cdots, n$.
- p_i^s, i.e., $p_{i,0}$, is randomly generated in $[30, 100]$ for $i = 1, 2, \cdots, n$.
- p_{ij} is randomly generated in $[1, p_i^s]$ for $j = 1, 2, \cdots, r_i$.

- t_i^{ss}, t_i^{sh}, t_i^{hs} and t_i^{hh} are randomly generated in $[0, t]$ for each i, where t is called *communication time basis*, $0 \leq t \leq 100$.

- p_i^{ss}, p_i^{sh}, p_i^{hs} and p_i^{hh} are randomly generated in $[0, p]$ for each i, where p is called *communication power basis*, $0 \leq p \leq 100$.

Let $\mathbf{x}^*$ be the optimal solution produced by DPA, $B(\mathbf{x}^*)$ be the solution value of $\mathbf{x}^*$. The quality of an approximate solution $\mathbf{x}$ is measured by

$$\varepsilon(\mathbf{x}) = \frac{B(\mathbf{x}) - B(\mathbf{x}^*)}{B(\mathbf{x}^*)} \times 100\%.$$

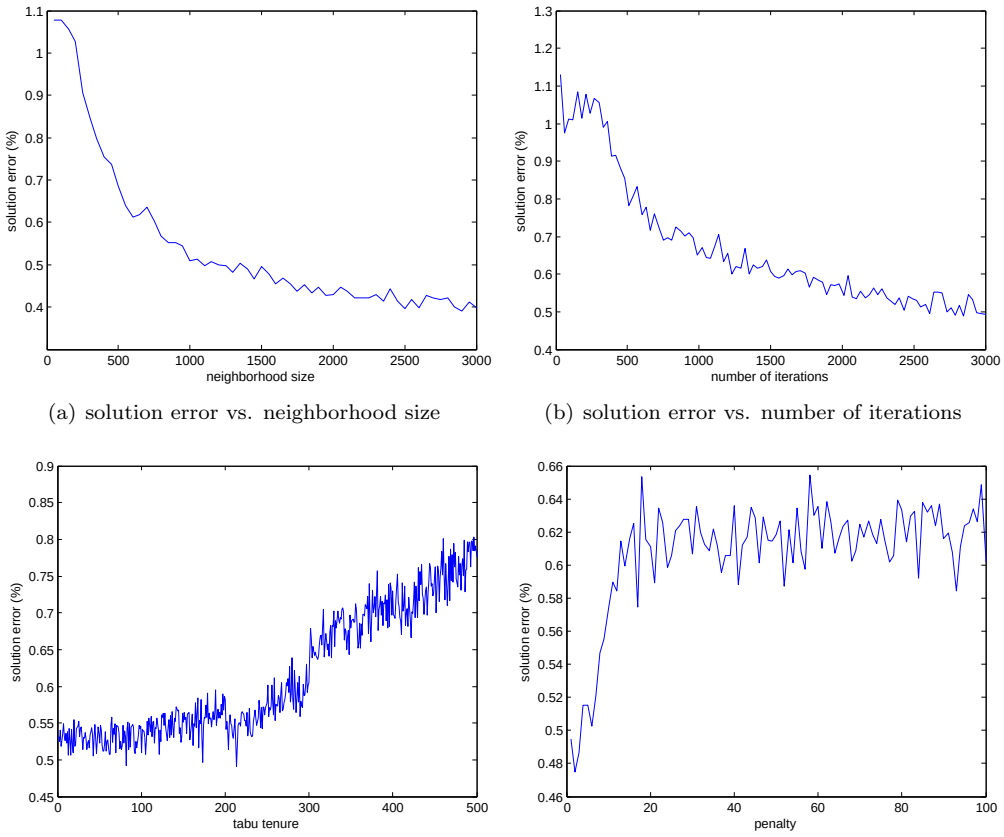
Here, $\varepsilon(\mathbf{x})$ is called solution error. In our simulations, when one parameter (e.g., neighborhood size in figure 3(a)) changes, all other parameters are set to a fixed value.

4.1. Solution by TSA

For a given problem, the solution produced by TSA mainly depends on the utilized 4 parameters, neighborhood size, the number of the iterations, tabu tenure and penalty value [6][20][22]. Without loss of generality, the area constraint is set to $50\% \cdot \bar{A}$, $\alpha = 0.5$, communication time (power) is set to 50, and each block is assumed to have at most 3 hardware implementation ways. Figure 3 shows the relationship between solution error and the 4 parameters of TSA on the random instance with 1000 blocks. The solution error appears smaller and smaller with the increasing neighborhood size, as shown in figure 3(a). This is due to the fact that when more neighbors are considered in the search process, there is a higher probability of getting a better solution. The solution error decreases faster for the neighborhood size in $(0, 500]$. While the solution error continues to decrease after 1000, the effort to refine the current solution becomes increasingly difficult. Hence, we choose 1000 as the neighborhood size of TSA in our simulations. Similar characteristics are exhibited for the number of the iterations as shown in figure 3(b). TSA may refine the HEA solution to a better solution, and the solution error is reduced from 1.13% to 0.5% with increasing number of iterations. After 1000 iterations, the rate of decreasing error is not significant. From figure 3(c) and (d), we can observe that 100 and 2 are relatively good values for the tabu tenure and the penalty respectively as they may lead to the solutions with the minimum error.

4.2. Performance Comparison

The proposed three algorithms are compared based on two metrics, solution value and solution error, with different communication time, communication power, area constraint, number of ways (implemen-



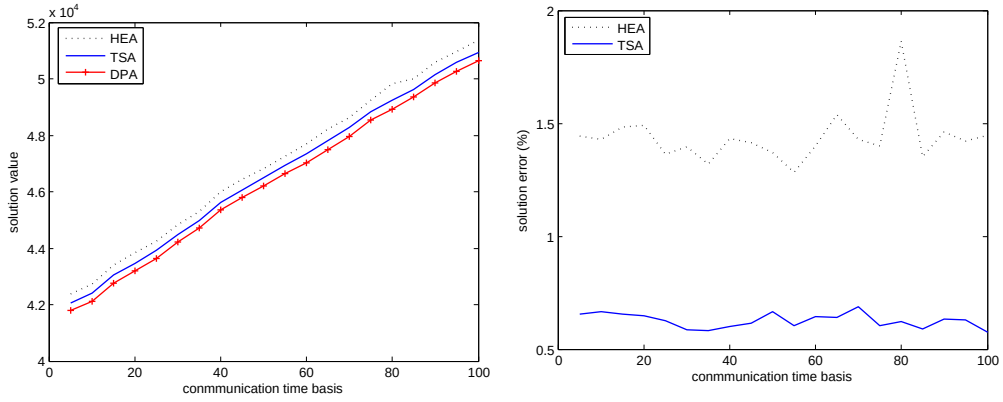
(a) solution error vs. neighborhood size

(b) solution error vs. number of iterations

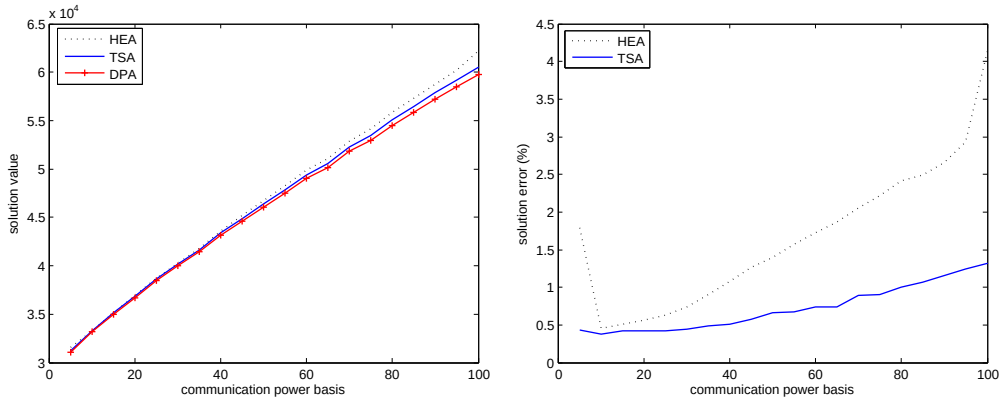
(c) solution error vs. tabu tenure

(d) solution error vs. penalty

Figure 3: TSA solution quality on different parameters, averaged over 20 instances with $n = 1000$, $A = 50\% \cdot \bar{A}$, $\alpha = 0.5$, $t = 50$, $p = 50$ and $r = 3$. The neighborhood size, the number of the iterations, tabu tenure and penalty value are set to 1000, 1000, 100, and 2, respectively.



(a) solution value vs. communication time basis (b) solution error vs. communication time basis



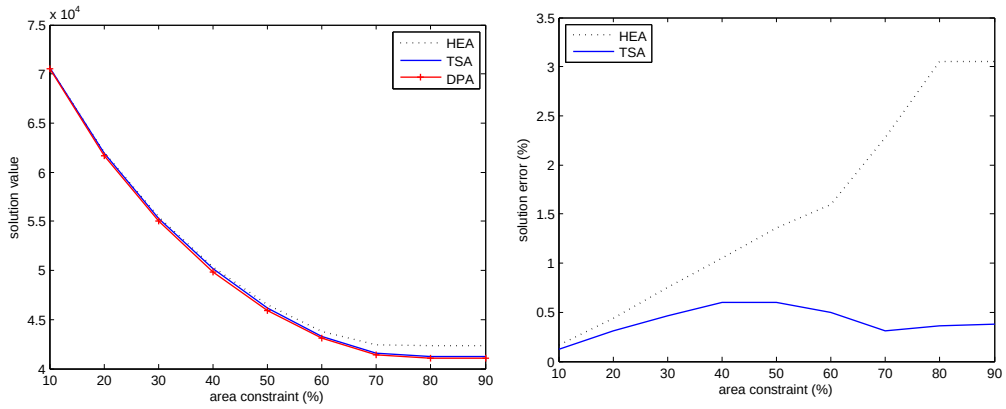
(c) solution value vs. communication power basis (d) solution error vs. communication power basis

Figure 4: Solution vs. communication for algorithms HEA, TSA and DPA, averaged over 20 instances with $n = 1000$, $A = 50\% \cdot \bar{A}$, $\alpha = 0.5$, $t = 50$, $p = 50$ and $r = 3$. The neighborhood size, the number of the iterations, tabu tenure and penalty value are set to 1000, 2000, 100, and 2, respectively.

tation choices). From figure 4(a) and 4(b), we can observe that the solution error is almost stable with varying communication time. The TSA can refine the solution error of HEA to be under 1%. Foe example the solution error of HEA is about 1.4% for communication time basis is 40, and it can be refined to 0.6% by TSA in figure 4(b).

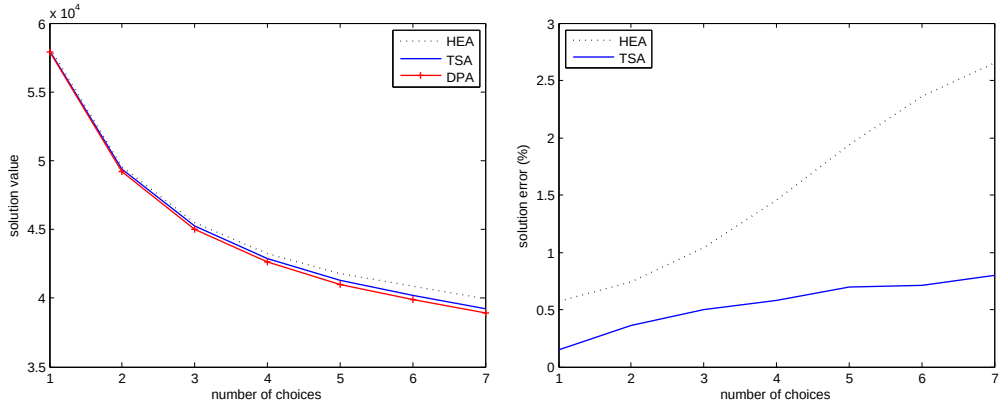
Different scenario are exhibited when the communication power varies. The solution error of HEA initially drops below 10 then rises. This is due to the combined influence of communication time and communication power. However, the solution of HEA can be significantly refined by TSA, resulting in better solutions with the error of no more than 1.5% as shown in figure 4(d).

From figure 5(a) and (b), it can be observed that HEA is able to produce nearly optimal solutions when the area is tightly constrained ($A < 20\% \cdot \bar{A}$). Each solution value gradually becomes better with increasing hardware area. This is because more hardware area generally supports more blocks in hardware



(a) solution value vs. area constraint

(b) solution error vs. area constraint



(c) solution value vs. number of choices

(d) solution error vs. number of choices

Figure 5: Solution vs. area and number of choices for algorithms HEA, TSA and DPA, averaged over 20 instances with $n = 1000$, $A = 50\% \cdot \bar{A}$, $\alpha = 0.5$, $t = 50$, $p = 50$ and $r = 3$. The neighborhood size, the number of the iterations, tabu tenure and penalty value are set to 1000, 2000, 100, and 2, respectively.

implementation, resulting in smaller solution values (execution time and power consumption). TSA can still maintain the solution error to be under 1.0%.

As shown in figure 5(c), larger number of implementation choices also contribute to reduction in the solution value. For example, the solution value of each algorithm is more than 5.7×10^4 for the case where only one choice (implementation option) is allowed. But it reduces to about 4.0×10^4 for the case where 6 choices are allowed. With the increasing number of the implementation choices, the solution error of the two algorithms become larger. However the solution error for HEA is less than 2.8% and the solution error of TSA is under 1.0%, as shown in figure 5(d).

Table 1: The solution quality ε (%) and the algorithm runtime $\mathcal{T}$ (ms) on different area constraint, averaged over 20 random instances, $r = 3$, $t = 50$ and $p = 50$.

	$A = 20\% \cdot \bar{A}$						$A = 50\% \cdot \bar{A}$						$A = 80\% \cdot \bar{A}$					
	DPA		HEA		TSA		DPA		HEA		TSA		DPA		HEA		TSA	
n	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$
100	5973.7	16.4	0.38	0.0	0.10	1585.8	4477.6	43.7	1.13	0.0	0.18	837.0	3948.4	61.6	2.58	0.0	0.07	578.8
150	9038.5	36.7	0.58	0.0	0.12	1915.0	6758.2	87.9	1.10	0.0	0.25	1088.2	5873.4	142.7	2.69	0.0	0.08	679.5
200	11921.9	64.8	0.32	0.0	0.11	2250.3	9022.2	162.3	1.07	0.0	0.26	1286.2	7927.1	255.1	2.51	0.0	0.10	813.8
250	15007.0	99.0	0.43	0.0	0.14	2441.4	11304.6	248.1	1.06	0.0	0.33	1341.0	9784.1	401.8	2.67	0.0	0.09	911.1
300	17982.1	144.0	0.36	0.1	0.16	2878.5	13555.8	362.7	1.11	0.0	0.31	1619.9	11885.9	568.6	2.50	0.0	0.10	1030.4
500	30155.8	399.4	0.35	0.8	0.17	4211.3	22574.0	998.6	1.10	2.3	0.46	2219.0	19715.6	1598.2	2.63	0.8	0.12	1406.5
600	35954.8	581.8	0.36	1.5	0.19	4531.8	26995.8	1440.7	1.13	2.4	0.47	2637.9	23803.3	2318.9	2.69	1.6	0.13	1633.3
700	41866.0	781.4	0.35	1.6	0.21	5349.3	31408.1	1967.3	1.08	3.1	0.46	2975.9	27685.8	3146.5	2.60	2.3	0.13	1842.4
800	47992.3	1026.0	0.36	1.8	0.21	5741.7	36031.1	2569.5	1.08	3.9	0.47	3270.0	31700.1	4102.2	2.54	2.4	0.12	2009.3
900	54069.6	1293.9	0.34	3.1	0.22	6389.3	40692.3	3252.7	1.03	4.7	0.50	3622.4	35518.6	5211.9	2.74	3.2	0.15	2227.0
1000	60112.1	1609.2	0.35	3.9	0.26	6993.4	45124.8	4042.1	1.16	7.1	0.56	3996.0	39489.3	6482.8	2.72	6.7	0.17	2455.7
1500	89910.2	3766.1	0.35	8.0	0.27	9958.9	67696.5	9335.2	1.12	7.8	0.61	5658.1	59255.3	14985.5	2.64	14.2	0.24	3428.9
2000	120162.5	3443.7	0.33	9.3	0.28	12881.1	-	-	-	17.8	-	7246.9	-	-	-	21.3	-	4400.8

4.3. Solution Quality and Runtime

Table 1 and table 2 show the quality of the approximate solutions and the runtime of the proposed algorithms on the problems with different area constraints and α. The smaller the value of ε, the higher the quality of the approximate solution. As shown in Table 1 and Table 2, the solution quality of HEA is relatively stable for varying problem size (number of the blocks). In contrast to HEA, the problem size slightly impacts the solution quality of TSA.

TSA can generate better solutions than HEA, but it takes longer runtime than HEA. For the case where $n = 1000$ and $A = 50\% \cdot \bar{A}$ in table 1, HEA produces a solution with error 1.16 in 7.1ms, while TSA generates better solution with error 0.56 but in 3996ms. This is the trade-off between the solution accuracy and computational effort.

DPA can generate an exact solution, but it is limited by the computer memory constraints. This is reflected as ' - ' in table 1 in cases where the number of blocks or the available hardware area is large. It is noteworthy that in these cases, HEA can continue to produce good approximate solutions.

5. Conclusions

In modern embedded systems each task has various implementation options both in software and in hardware, where different implementation is of different speed, power and cost. This leads to an explosion in the design space. In order to overcome the challenges in embedded systems design, this paper has introduced a bi-objective multiple-choice HW/SW partitioning problem, which is characterized of mul-

Table 2: The solution quality ε (%) and the algorithm runtime $\mathcal{T}$ (ms) on different α, averaged over 20 random instances, $r = 3$, $t = 50$ and $p = 50$.

	$\alpha = 0.2$						$\alpha = 0.5$						$\alpha = 0.8$					
	DPA		HEA		TSA		DPA		HEA		TSA		DPA		HEA		TSA	
n	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$	$B(\mathbf{x}^*)$	$\mathcal{T}$	ε	$\mathcal{T}$	ε	$\mathcal{T}$
100	4625.1	45.3	1.38	0.0	0.30	765.0	4477.6	43.7	1.13	0.0	0.18	837.0	3892.6	36.8	1.35	0.0	0.18	916.4
150	6904.0	88.1	1.55	0.0	0.34	1003.9	6758.2	87.9	1.10	0.0	0.25	1088.2	5957.4	90.6	1.21	0.0	0.25	1136.5
200	9124.2	159.9	1.45	0.0	0.32	1103.7	9022.2	162.3	1.07	0.0	0.26	1286.2	7960.7	160.0	1.10	0.0	0.39	1314.4
250	11467.2	248.2	1.40	0.0	0.36	1278.5	11304.6	248.1	1.06	0.0	0.33	1341.0	9983.5	252.9	1.50	0.0	0.46	1592.8
300	13792.9	359.5	1.53	0.0	0.39	1510.8	13555.8	362.7	1.11	0.0	0.31	1619.9	11980.4	357.2	1.26	0.0	0.47	1767.5
500	22994.4	1000.8	1.41	0.8	0.46	2055.3	22574.0	998.6	1.10	2.3	0.46	2219.0	19926.5	1003.0	1.15	1.6	0.47	2627.1
600	27578.6	1446.1	1.40	2.4	0.58	2412.5	26995.8	1440.7	1.13	2.4	0.47	2637.9	23982.7	1440.7	1.21	2.3	0.56	2949.2
700	32254.0	1956.3	1.39	3.1	0.56	2708.9	31408.1	1967.3	1.08	3.1	0.46	2975.9	28010.5	1951.6	1.27	2.4	0.65	3342.4
800	36754.2	2578.0	1.38	3.9	0.60	3000.6	36031.1	2569.5	1.08	3.9	0.47	3270.0	32070.7	2551.5	1.18	3.8	0.59	3758.2
900	41454.3	3252.6	1.28	4.0	0.54	3351.0	40692.3	3252.7	1.03	4.7	0.50	3622.4	35906.9	3234.7	1.16	3.9	0.68	4095.0
1000	46107.4	4025.4	1.28	5.5	0.59	3673.0	45124.8	4042.1	1.16	7.1	0.56	3996.0	40110.0	4001.4	1.13	4.7	0.67	4396.0
1500	69061.2	9424.7	1.39	11.0	0.70	5162.0	67696.5	9335.2	1.12	7.8	0.61	5658.1	59836.1	9378.9	1.17	10.3	0.75	6340.0
2000	-	-	-	19.7	-	7029.0	-	-	-	17.8	-	7246.9	-	-	-	18.8	-	8129.3

tiple conflicting design goals for the complex applications. We optimize the solution of the partitioning problem with the following two objectives: minimizing execution time and power consumption, while meeting the area constraint. We have extended the computing model of the problem so that the power consumption of each component and the communication between neighboring components are considered. The new computing model has been formulated as a non-linear minimization optimizing problem. We have proposed three algorithms utilizing the weighted sum scalarization method, 1) an efficient heuristic algorithm based on the bi-objective knapsack problem to rapidly generate an approximate solution, 2) a tabu search algorithm that refines the solution of the heuristic method, and 3) a dynamic programming algorithm to calculate the exact solution. The proposed heuristic algorithm together with tabu search, can be applied to solve large problems, while the dynamic programming algorithm can provide exact solutions for relatively small problems. It has been shown that our heuristic algorithm is able to rapidly generate approximate solutions that are very close to the exact ones. In addition, the proposed tabu search algorithm can further refine the heuristic method to generate the solutions with errors that are less than 1.0% for all cases considered in this paper.

Acknowledgments

This work was supported by the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20131201110002, and the Key Laboratory of Computer Architecture Opening Topic Fund Subsidization under Grant No. CARCH201303.

References

[1] S. Edwards, L. Lavagno, E. Lee, A. Sangiovanni-Vincentelli, Design of embedded systems: Formal models validation, and synthesis, In:Proceedings of the IEEE 85 (3) (1997) 366–390. doi:10.1109/5.558710.

[2] R. Gupta, G. D. Micheli, Hardware-software cosynthesis for digital systems, IEEE Design and Test of Computers 10(3) (1993) 29–41. doi:10.1109/54.232470.

[3] J. Wu, T. Srikanthan, C. Yan, Algorithmic aspects for power-efficient hardware/software partitioning, Mathematics and Computers in Simulation 79(4) (2008) 1204–1215. doi:10.1016/j.matcom.2007.09.003.

[4] R. Ernst, J. Henkel, T. Benner, Hardware-software co-synthesis for micro-controllers, IEEE Design and Test of Computer 10(4) (1993) 64–75. doi:10.1109/54.245964.

[5] F. Vahid, D. Gajski, Clustering for improved system-level functional partitioning, In: Proceedings of the 8th International Symposium on System Synthesis (1995) 28–33doi:10.1109/ISSS.1995.520609.

[6] P. Arato, Z. Mann, A. Orban, Algorithmic aspects of hardware/software partitioning, ACM Transactions on Design Automation of Electronic Systems 10(1) (2005) 136–156. doi:10.1145/1044111.1044119.

[7] S. Banerjee, N. Dutt, Efficient search space exploration for hw-sw partitioning, In:Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2004) 122–127doi:10.1109/CODESS.2004.240863.

[8] J. Wu, T. Srikanthan, Low-complex dynamic programming algorithm for hardware/software partitioning, Information Processing Letters 98 (2006) 41–46. doi:10.1016/j.ipl.2005.12.008.

[9] J. Madsen, J. Grode, P. Knudsen, M. Petersen, A. Haxthausen, Lycos: The lyngby co-synthesis system, Information Processing Letters 2 (1997) 195–235. doi:10.1023/A:1008884219274.

[10] P. Arato, S. Juhasz, Z. Mann, A. Orban, D. Papp, Hardware-software partitioning in embedded system design, In:Proceedings of the IEEE International Symposium on Intelligent Signal Processing (2003) 197–2025doi:10.1109/ISP.2003.1275838.

[11] R. Niemann, P. Marwedel, An algorithm for hardware/software partitioning using mixed integer linear programming, Design Automation for Embedded Systems, special Issue: Partitioning Methods for Embedded Systems 2(2) (1997) 165–193. doi:10.1023/A:1008832202436.

[12] G. Quan, X. Hu, G. Greenwood, Preference-driven hierarchical hardware/software partitioning, In:Proceedings of IEEE International Conference on Computer Design (1999) 652–657doi:10.1109/ICCD.1999.808611.

[13] S. Tahaee, A. Jahangir, H. Habibi-Masouleh, Improving the performance of heuristic searches with judicious initial point selection, In:Proceedings of the 5th IEEE International Symposium on Embedded Computing (2008) 14–19doi:10.1109/SEC.2008.65.

[14] M. Abdelhalim, S. Habib, An integrated high-level hardware/software partitioning methodology, Design Automation for Embedded Systems 15(1) (2011) 19–50. doi:10.1007/s10617-010-9068-9.

[15] M. Yuan, Z. Gu, X. He, X. Liu, L. Jiang, Hardware/software partitioning and pipelined scheduling on runtime reconfigurable fpgas, ACM Transactions on Design Automation of Electronic Systems 15(2) (2010) 1–41. doi:10.1145/1698759.1698763.

[16] J. Mu, L. Roman, Autonomous hardware/software partitioning and voltage/frequency scaling for low-power embedded systems, ACM Transactions on Design Automation of Electronic Systems 15(1) (2009) 2. doi:10.1145/1640457.1640459.

[17] T. Wiangtong, P. Cheung, W. Luk, Comparing three heuristic search methods for functional partitioning in hardwaresoftware codesign, Design Automation for Embedded Systems 6(4) (2002) 425–449. doi:10.1023/A:1016567828852.

[18] M. Lopez-Vallejo, J. Lopez, On the hardware-software partitioning problem: System modeling and partitioning techniques, ACM Transactions on Design Automation of Electronic Systems 8(3) (2003) 269–297. doi:10.1145/785411.785412.

[19] J. Wu, P. Wang, S. Lam, T. Srikanthan, Efficient heuristic and tabu search for hardware/software partitioning, Journal of Supercomputing 66(1) (2013) 118–134. doi:10.1007/s11227-013-0888-9.

[20] J. Wu, T. Srikanthan, G. Chen, Algorithmic aspects of hardware/software partitioning: One-dimensional search algorithms, IEEE Transactions on Computers 59(4) (2010) 532–544. doi:10.1109/TC.2009.173.

[21] S. Tahaee, A. Jahangir, A polynomial algorithm for partitioning problems, ACM Transactions on Embedded Computing Systems 9(4) (2010) 34:134:38. doi:10.1145/1721695.1721700.

[22] J. Wu, Q. Sun, T. Srikanthan, Algorithm aspects for multi-choice hardware/software partition, Computers and operations research 39 (2012) 3281–3292. doi:10.1016/j.cor.2012.04.013.

[23] Y. L. Aung, S. K. Lam, T. Srikanthan, Addressing productivity challenges in domain-specific reconfigurable platforms: A case study on extended kalman filter-based motor control, Journal of Low Power Electronics 10 (3) (2014) 455–466. doi:10.1166/jolpe.2014.1342.

[24] H. kellerer, U. Pferschy, D. Pisinger, Knapsack problems, Springer-Verlag Press.

[25] F. Glover, M. Laguna, Tabu search, Kluwer Academic 2 (1997) 10–150.

Wenjun Shi is a PhD student in Tianjin Polytechnic University. Her main research interests include hardware/software co-design and high performance architecture.

Jigang Wu received his B.S. degree from Lanzhou University and Ph.D. degree from the University of Science and Technology of China. He was with Nanyang Technological University, Singapore, from 2000 to 2009. He was with Tianjin Polytechnic University, China, in 2009. He joined Guangdong university of technology, China, from 2015. His research interests include high performance architecture, hardware/software co-design.

Siew-Kei Lam received his BASc, MEng and PhD from Nanyang Technological University (NTU), Singapore. He is currently an Assistant Professor in School of Computer Engineering, NTU and his research investigates methods for realizing custom computing solutions in embedded systems. He has published over 75international refereed journals and conferences in design methodologies forheterogeneous systems, embedded vision, and computer arithmetic.

Thambipillai Srikanthan joined Nanyang Technological University, Singapore in 1991. At present, he holds chair professor and joint appointment as Director of Centre for High Performance Embedded Systems (CHiPES). He founded CHiPES and elevated it to a university level research Centre. His research interests include design methodologies for complex embedded systems, architectural translations of compute intensive algorithms.