J Supercomput (2013) 66:118–134
DOI 10.1007/s11227-013-0888-9

Efficient heuristic and tabu search for hardware/software partitioning

Jigang Wu · Pu Wang · Siew-Kei Lam · Thambipillai Srikanthan

Published online: 7 February 2013

Abstract Hardware/software (HW/SW) partitioning is a crucial step in HW/SW codesign that determines which components of the system are implemented on hardware and which ones on software. It has been proved that the HW/SW partitioning problem is NP-hard. In this paper, we present two approaches for HW/SW partitioning that aims to minimize the hardware cost while taking into account software and communication constraints. The first is a heuristic approach that treats the HW/SW partitioning problem as an extended 0–1 knapsack problem. In the second approach, tabu search is used to further improve the solution obtained from the proposed heuristic algorithm. Experimental results show that the proposed algorithms outperform a recently reported work by up to 28 %.

Keywords Algorithm · Heuristic · Tabu search · Hardware/software partitioning

1 Introduction

Hardware/software (HW/SW) codesign techniques are often employed to realize modern embedded systems that typically consist of hardware and software components. HW/SW partitioning is a crucial step in codesign that has been shown to have a

J. Wu (✉) · P. Wang
School of Computer Science and Software Engineering, Tianjin Polytechnic University, 300387 Tianjin, China
e-mail: asjgwu@gmail.com

J. Wu
e-mail: asjgwu@ntu.edu.sg

S.-K. Lam · T. Srikanthan
Centre for High Performance Embedded Systems, Nanyang Technological University, 639798 Singapore, Republic of Singapore

T. Srikanthan
e-mail: astsrikan@ntu.edu.sg

dominant effect on overall system performance [1]. The application to be partitioned is generally given in the form of a task graph $G = (V, E)$, and the vertex set V indicates the tasks that have to be mapped to either hardware or software components. The edge set E represents communication between the components. HW/SW partitioning aims to partition the vertex set in the task graph into two disjoint subsets to satisfy certain constraints while addressing some salient factors such as communication cost or inherent overhead introduced by the management of hardware resources [2].

Exact algorithms such as dynamic programming [3, 4], integer linear programming [5, 6] and branch-and-bound [7] are generally utilized for HW/SW partitioning when the problem size is small. As most formulations of the partitioning problem are NP-hard, exact algorithms quickly become infeasible when the problem size increases. Thus, approximate methods often provide a more natural and feasible approach for HW/SW partitioning with large problem sizes.

Traditional heuristics used in approximate algorithms include hardware-oriented and software-oriented approaches [8]. The hardware-oriented approach begins with a complete hardware solution and iteratively moves parts of the application to software while fulfilling the performance constraints [9, 10]. The software-oriented approach starts with a software program and iteratively moves application segments to hardware in order to improve speed while satisfying time constraint [11–13]. General-purpose heuristics for HW/SW partitioning include genetic algorithms (GA), simulated annealing (SA), tabu search (TS), and particle swarm optimization (PSO). In [14], three heuristic search algorithms, GA, SA, and TS, are compared. The three algorithms run on functional blocks for designs represented as directed acyclic graphs, with the objective of minimizing processing time under various hardware area constraints. In [15], the computing model of the embedded system is extended so that the resource contentions are taken into account, and then a GA-based algorithm is proposed on the extended model. In [16], a computing model is presented for path-based HW/SW partitioning in which communication penalties between system components is considered, followed by an efficient tabu search algorithm to refine the approximate solutions. In addition, PSO technique is introduced in [17], and some other heuristic algorithms are introduced in [18, 19] for HW/SW partitioning, targeting reconfigurable embedded system [20].

It is worthwhile to point that HW/SW partitioning may have multiple objectives such as minimizing power, execution time, hardware cost, communication overhead, etc. As these objectives are often mutually dependent, a typical multiconstrained optimization problem for HW/SW partitioning examines the design trade-offs of a given application, and provides a reasonable partitioning solution that meets user requirements. HW/SW partitioning has been categorized into two types, and it has been proved that, one of them is of NP-hard [21]. The latest discussion on this NP-hard version is in our previous work [22], where we have shown that the HW/SW partitioning problem can be reduced to a variation of knapsack problem.

In this paper, we initially propose a heuristic approach for HW/SW partitioning based on the partitioning objectives, assumptions, and the system model utilized in [21, 22]. The proposed heuristic strategy aims to produce an infeasible but approximate solution initially, without considering communication cost, and then adjusting the approximate solution to be feasible by taking into account the communication

cost. We then employ a customized tabu search approach to further refine the heuristic solution. Extensive experimental results demonstrate the superiority of the proposed algorithms in this paper over the latest approach in [22] in terms of solution quality.

This paper is organized as follows. Section 2 provides the formal description of task graph for hardware/software partitioning, as well as some notations used throughout the paper. In Sect. 3, we describe the details of the proposed heuristic algorithm. In Sect. 4, we present the tabu search algorithm, which is used to refine our heuristic algorithm. Section 5 shows the experimental results and compare the proposed approaches with the existing method. Finally, Sect. 6 concludes the paper.

2 Problem definition and previous algorithms

The definition of the partitioning problem discussed in this paper is based on the following notations. Given an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$, and E indicates the set of edges. $s(v_i)$ (or simply s_i) and $h(v_i)$ (or h_i) denote the software and hardware cost of node v_i, respectively, while $c(i, j)$ denotes the communication cost between v_i and v_j if they are in different contexts. The partition function π induce a new graph $G_\pi = G_\pi(V, E_P)$, where $V = (V_h, V_s)$ and an edge $(V_h, V_s) \in E_P$ exists if there are two adjacent vertices $u, v \in V$ such that $u \in V_h$ and $v \in V_s$. The set E_P corresponds to the set of cutting edges of G induced by the partition.

In this paper, H_P, S_P, and C_P represent the hardware cost, software cost, and communication cost, respectively, under a given partition P. The partitioning problem is modeled as follows [22].

Problem $\mathcal{P}$ Given a graph G with the cost functions s, h, and c, and $R \geq 0$, find a HW/SW partition P with $S_P + C_P \leq R$ that minimizes H_P.

In the n-dimensional space $\{0, 1\}^n$, let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Here, $\mathbf{x}$ denotes a solution of the problem $\mathcal{P}$, i.e., a partition for the graph G with n nodes. $x_i = 1$ ($x_i = 0$) indicates that the node v_i is partitioned to software (hardware), $1 \leq i \leq n$. $C(\mathbf{x})$ indicates the communication cost of the solution $\mathbf{x}$. As a result, the problem $\mathcal{P}$ can be formulated as the following minimization problem for given R. For more details, see [22].

$$\mathcal{P} \begin{cases} \text{minimize} & \displaystyle\sum_{i=1}^{n} h_i(1 - x_i), \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} s_i x_i + C(\mathbf{x}) \leq R, \\ & x_i \in \{0, 1\},\ i = 1, 2, \ldots, n. \end{cases}$$

The problem $\mathcal{P}$ can be easily converted to the problem $\mathcal{Q}$ as follows:

$$\mathcal{Q}\begin{cases}\text{maximize} & \displaystyle\sum_{i=1}^{n} h_i x_i, \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} s_i x_i + C(\mathbf{x}) \le R, \\ & x_i \in \{0,1\},\ i = 1,2,\ldots,n.\end{cases}$$

We now review knapsack problem that is closely related to the partitioning problem $\mathcal{Q}$. Given a knapsack capacity K and a set of items $S = \{1, 2, \ldots, n\}$, where each item has a weight w_i and a benefit b_i, the knapsack problem aims to find a subset $S' \subset S$, that maximizes the total profit $\sum_{i \in S'} b_i$ under the constraint that $\sum_{i \in S'} w_i \le K$, i.e., all the items fit in a knapsack of capacity K. Mathematically, it can be described as follows:

$$\mathcal{K}\begin{cases}\text{maximize} & \displaystyle\sum_{i=1}^{n} b_i x_i, \\ \text{subject to} & \displaystyle\sum_{i=1}^{n} w_i x_i \le K, \\ & x_i \in \{0,1\},\ i = 1,2,\ldots,n,\end{cases}$$

where x_i is a binary variable which is assigned a value of 1 if item i is included in the knapsack and 0 otherwise. This 0/1 property makes the knapsack problem NP-hard.

In [22], problem $\mathcal{Q}$ was regarded as an extended 0–1 knapsack problem and the partitioning problem was transformed into a one-dimensional (1D) search problem. Three algorithms were proposed for solving the partitioning problem by searching the 1D search space. The approximate optimal solution for the partitioning problem was selected from the feasible solutions of the corresponding knapsack algorithms. The time complexity of algorithms in [22] is $O(n \log n + d \cdot (n + m))$ for graphs with n nodes and m edges, where d is the number of the fragments of the searched solution spaces. For more details of these algorithms, see [22].

3 Proposed heuristic algorithm

Let h_i, s_i and R of problem $\mathcal{Q}$ correspond to b_i, w_i, and K of the problem $\mathcal{K}$, respectively. Problem $\mathcal{Q}$ can now be regarded as an extension of the problem $\mathcal{K}$ with an additional communication cost in the constraint. Hence, algorithms for solving the knapsack problem can be applied to solve problem $\mathcal{Q}$. In this section, a heuristic algorithm based on 0–1 knapsack has been proposed to solve the HW/SW partitioning problem.

A simple but effective algorithm for 0–1 knapsack problem is presented in [23]. It first sorts the items according to their profit-to-weight ratios as follows:

$$\frac{b_1}{w_1} \ge \frac{b_2}{w_2} \ge \cdots \ge \frac{b_n}{w_n}.$$

The algorithm then packs items into the knapsack until the knapsack can no longer accommodate any more item.

Let $\mathbf{x} = (0, 0, \ldots, 0)$ be the initial solution, which corresponds to all components being assigned to hardware. Assigning the component to software is equivalent to packing it into the knapsack. As mentioned in Sect. 2, h_i and s_i indicate the hardware cost and software cost of the component i, respectively. Thus, h_i and s_i can be regarded as the profit and the weight of item i respectively, in the corresponding knapsack problem. In other words, the components to be assigned to software is regarded as the items to be packed into the knapsack.

In the partitioned graph $G_\pi(V, E_P)$, where $V = (V_h, V_s)$, assume the node v_i is in V_h. Then the communication cost corresponding to v_i is $\sum_{k \in V_s} c(i, k)$. If v_i is moved to V_s, the corresponding communication cost becomes to $\sum_{j \in V_h} c(i, j)$, resulting in the cost change c_i calculated by

$$c_i = \sum_{j \in V_h} c(i, j) - \sum_{k \in V_s} c(i, k).$$

It is clear that the problem $\mathcal{Q}$ is reduced to a standard knapsack problem if the communication cost is not considered (i.e., $C(\mathbf{x}) = 0$). This motivates us to solve the problem $\mathcal{Q}$ using the heuristic solution of the corresponding knapsack problem. Our main idea is generating an infeasible but approximate solution for $\mathcal{Q}$ initially, without the consideration for communication cost, and then adjusting the approximate solution to be feasible by considering the impact of the communication cost.

It is noteworthy that the proposed heuristic strategy in this paper is notably different from that presented in [22], where the communication cost is evaluated to be a constant to form a standard knapsack problem in each iteration. The approximate solution of the problem $\mathcal{Q}$ is selected from the heuristic solutions of a sequence of knapsack problems.

Now we outline the heuristic approach as follows:

(1) Initially, we sort the components according to $\frac{h_1}{s_1} \geq \frac{h_2}{s_2} \geq \cdots \geq \frac{h_n}{s_n}$. Then we assign the component i to software, i.e., pack the item i into the knapsack with capacity R, for $i = 1, 2, \ldots$, till $\sum_i s_i \geq R$.
(2) As the current solution is infeasible, we now select the component with the minimum value of $\frac{h_i}{s_i + c_i}$ and then assign it to hardware. In other words, we move the item out of the knapsack. After that, we update the system cost, i.e., hardware cost, software cost, and communication cost. This work repeats till the solution is refined to be feasible according to the limit of knapsack capacity R.
(3) After that, we further adjust the current solution by selecting the component with the maximum value of $\frac{h_i}{s_i + c_i}$ and assigning it to software.

In step (1), the communication cost is not considered, and an infeasible solution is obtained. In step (2), we evaluate the communication change c_i for moving component i to hardware, $s_i + c_i$ represents the capacity which would be released if the component i is removed from the knapsack, and h_i is the decreased amount of total profit. Likewise, in step (3), we also evaluate c_i for moving component i to software, $s_i + c_i$ represents the capacity which would be consumed if component i is moved

into knapsack, and h_i is the increased amount of total profit. It is noteworthy that an item to be removed from the knapsack should ideally minimize the loss of profit and maximize the released capacity. On the other hand, an item to be moved into knapsack should ideally maximize the benefit and consume minimal knapsack capacity. The following pseudocode shows the formal description of the HEUR (Algorithm 1).

Analysis We provide analysis for HEUR on a graph $G = (V, E)$ with n nodes and m edges. The task number is consistent with the node number in this graph.

Theorem 1 *The time complexity of* HEUR *is* $O(n \log n + k \cdot (n + m))$, *where* $n = |V|$, $m = |E|$, k *is the number of the node movements between hardware and software contexts in partitioning.*

Proof The time complexity of HEUR is dominated by two loops: lines 13–20 and lines 22–31. Lines 5–10 can be implemented in $O(n)$ time [23] because a sorting process in line 3 is initially employed, which runs in $O(n \log n)$ time [24]. Lines 14–16 recalculate the communication change value (denoted as c_i in HEUR) if a node has been removed from knapsack. It will require $O(m)$ time because only the edges associated with the node which has been removed from knapsack needs to be reconsidered. Let m_i be the number of the edges associated with the node v_i. From $\sum_{i=1}^{n} m_i = 2m$, we conclude that $O(m)$ time is sufficient for this step. Line 17 searches for the minimum value in an unsorted sequence, and this step is bounded by $O(n)$. Updating $S(\mathbf{x})$, $H(\mathbf{x})$, $C(\mathbf{x})$ (see Sect. 2) in line 18 takes $O(1)$ time. Let k_1 be the number of the iterations of the loop from line 13 to line 19. Thus, the loop in lines 13–19 runs in $O(k_1 \cdot (n + m))$. Similarly, the loop in lines 22–31 runs in $O(k_2 \cdot (n + m))$, where k_2 is the number of the iterations of the loop. Let k be $\max\{k_1, k_2\}$. We conclude that the time complexity of HEUR is $O(n \log n + k \cdot (n + m))$. □

4 Proposed tabu search approach

Tabu search is a well-known metaheuristic that has been successfully used for solving difficult combinatorial optimization problems, whose applications range from graph theory and matroid setting to general pure and mixed integer programming problems [25]. Tabu search starts from an initial solution and iteratively moves to a new solution that is selected in a certain *neighborhood* of the current solution. At each iteration, the *move* yielding the best solution in the neighborhood is selected, even if this results in a worse solution [26]. Since tabu search relies on the principle that intelligent search is based on learning, it employs a flexible memory that keeps track of the search history. In order to avoid being trapped in cyclic search and to enable searching beyond local optimum, tabu search introduces the notion of *tabu list* to forbid recently visited solutions to be generated [27].

In this section, we describe the proposed tabu search strategy for refining the solution obtained from our heuristic algorithm.

Solution representation At any iteration t of our tabu search, a partitioning solution is denoted as $\mathbf{x}^t = (x_1^t, x_2^t, \ldots, x_n^t)$, where $x_i = 1$ ($x_i = 0$) indicates the ith component is assigned to software (hardware).

Algorithm 1 HEUR

Input:

Communication graph G and the constraint R.

Output:

A partitioning solution $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and the total hardware cost.

begin
$\mathbf{x} := (0, 0, \ldots, 0)$; *solution_value* $:= 0$ /* initializing */
Sort all tasks $\{v_i\}_{i \le n}$ according to $\frac{h_1}{s_1} \ge \frac{h_2}{s_2} \ge \cdots \ge \frac{h_n}{s_n}$;
$i := 1$; *rec* $:= R$; /* *rec* means *residual_capacity* */
repeat /* the communication costs are not considered */
  if $s_i \le rec$ **then** /* task i fits in the unused capacity */
    Pack task i into knapsack, and $x_i := 1$, $rec := rec - s_i$;
    $i := i + 1$;
  end if
until $(rec \le 0)$ **or** $(i > n)$ /* a greedy solution and the hardware cost is obtained */
if $(S(\mathbf{x}) + C(\mathbf{x}) \le R)$ **then** *solution_value* $:= H(\mathbf{x})$;
else *solution_value* $:= 0$;
  repeat
    for all i such that $x_i = 1$ **do** /* task i is in knapsack */;
      Evaluate communication change c_i and $\frac{h_i}{s_i + c_i}$ for moving task i out of the knapsack;
    end for
    Move task k with the minimum value of $\frac{h_k}{s_k + c_k}$ out of the knapsack and set x_k to 0;
    update $S(\mathbf{x}), C(\mathbf{x}), H(\mathbf{x})$;
  until $(S(\mathbf{x}) + C(\mathbf{x}) \le R)$
end if
$rec' := R - (S(\mathbf{x}) + C(\mathbf{x}))$;
repeat
  for all i such that $x_i = 0$ **do** /* task i is not in knapsack */
    Evaluate communication change c_i and $\frac{h_i}{s_i + c_i}$ for moving task i into the knapsack;
  end for
  if $s_k + c_k \le rec'$ **then**
    Move task k with the maximum value of $\frac{h_k}{s_k + c_k}$ into the knapsack and set x_k to 1;
    $rec' = rec' - (s_k + c_k)$;
  end if
  update $S(\mathbf{x}), C(\mathbf{x}), H(\mathbf{x})$;
until $(rec' \le 0)$ **or** (no more task to fit for the remaining capacity)
solution_value $:= H(\mathbf{x})$;
end

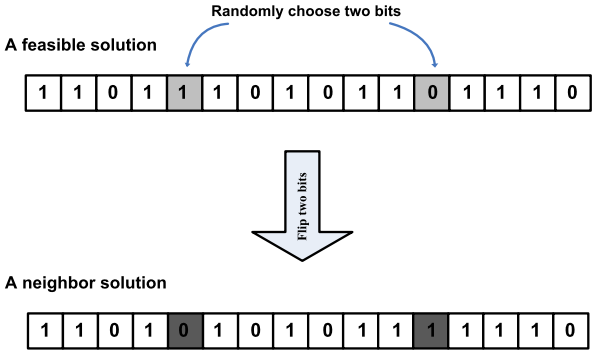


Fig. 1 Generating a neighbor of a feasible solution

Initial solution In this paper, we employ tabu search to refine HEUR proposed in Sect. 3. In particular, the solution of the proposed heuristic algorithm will be used as initial solution for tabu search. During the tabu process, the current solution $\mathbf{x}_{\text{current}}$ becomes the new best solution $\mathbf{x}_{\text{best\_so\_far}}$ only if $\mathbf{x}_{\text{current}}$ is at least as good as $\mathbf{x}_{\text{best\_so\_far}}$ in terms of the optimization objective for the given constraint.

Neighborhood structure Undoubtedly, the neighborhood structure has significant impact on the quality of solution. A different rule could result in a different solution of different quality [28]. In this work, the solution consists of a sequence of 1 s and 0 s. Hence, the neighborhood of a feasible solution set is a feasible solution obtained from flipping two different bits at random as shown in Fig. 1. This step flips the original $x_i = 1$ to $x_i = 0$, and vice versa.

Move and tabu status At the beginning of this process, no move is in tabu search. At any iteration, the algorithm executes the best non-tabu move to a feasible neighbor of the current solution [28]. However, if a tabu move yields a better incumbent, it will also be implemented. This is called *aspiration criterion*. In addition, if all the neighbors are tabu, the oldest one in *tabu list* is implemented. Whenever a move is performed, the reverse move is declared tabu for tt iterations (*tabu_tenure*), where tt is randomly generated in an interval $[\alpha * \sqrt{n}, \beta * \sqrt{n}]$. In the whole search process, a neighbor $\mathbf{x}_{\text{neib}}$ may enter the tabu list many times. Let us assume that the latest entrance for $\mathbf{x}_{\text{neib}}$ is in the iteration $iter\_late(\mathbf{x}_{\text{neib}})$ and the current search is at iteration *iter_curr*. The tabu degree of $\mathbf{x}_{\text{neib}}$, denoted as $Tdegree(\mathbf{x}_{\text{neib}})$, is defined as

$$Tdegree(\mathbf{x}_{\text{neib}}) = iter\_late(\mathbf{x}_{\text{neib}}) + tabu\_tenure - iter\_curr.$$

Tabu degree is updated for each neighbor in each iteration. A nonnegative tabu degree implies that the neighbor is tabu-active, while a negative one implies that the neighbor is not tabu-active. For more details, see the formal description of the algorithm TABU (Algorithm 2).

Selection strategy for candidate solutions Let $\mathbf{x}_{\text{neib}} = (x_1, x_2, \ldots, x_n)$, which is a neighbor of a current solution $\mathbf{x}_{\text{current}}$. We define our objective function *dobj* as

$$dobj(\mathbf{x}_{\text{neib}}) = H(\mathbf{x}_{\text{neib}}) - H(\mathbf{x}_{\text{current}}).$$

Algorithm 2 TABU

Input:

Communication graph G and the constraint R;
The initial partition $\mathbf{x}_{\text{heur}}$ obtained from HEUR.

Output:

a refined partitioning solution $\mathbf{x}_{\text{best\_so\_far}}$.

1: **begin**
2: Initialize *tabu_tenure*, *move_frequence_list*, set the iteration counter $iter = 0$ and begin with *tabu_list* empty.
3: $\mathbf{x}_{\text{current}} := \mathbf{x}_{\text{heur}}$, and $\mathbf{x}_{\text{best\_so\_ far}} := \mathbf{x}_{\text{heur}}$;
4: **repeat**
5: Generate q neighbors of $\mathbf{x}_{\text{current}}$;
6: Update the degrees and *dobj* of the q neighbors;
7: /* if a *neighbor* obtained from a tabu move is better than $\mathbf{x}_{\text{best\_so\_far}}$, its tabu status will be ignored. */
8: **if** all q neighbors are tabu-active **then**
9: $\mathbf{x}_{\text{current}}$:= the neighbor with the minimal tabu degree
10: **else**
11: $\mathbf{x}_{\text{current}}$:= the neighbor with the minimal *dobj*;
12: **end if**
13: **if** $H(\mathbf{x}_{\text{local}}) < H(\mathbf{x}_{\text{best\_so\_far}})$ **then**
14: $\mathbf{x}_{\text{best\_so\_far}} := \mathbf{x}_{\text{current}}$;
15: **end if**
16: **until** Termination criterions are met;
17: **end**

Thus, the smaller the value of $dobj(\mathbf{x}_{\text{neib}})$, the better quality the neighbor $\mathbf{x}_{\text{neib}}$ is. Here, $\text{H}(\mathbf{x}_{\text{neib}})$ and $\text{H}(\mathbf{x}_{\text{best\_so\_far}})$ denote the hardware cost under the partition $\mathbf{x}_{\text{neib}}$ and $\mathbf{x}_{\text{best\_so\_far}}$ respectively. The selection strategy, first conditioned by *tabu status* explained above, employed an additional criterion based on the *move frequency*, which is a long term memory that is used to record the number of times a component c has been moved to hardware or software. This mechanism is used to penalize moves with high frequent counts and favor moves with low frequent counts.

Termination criterion If the iteration counter *iter* has reached the given maximum iteration number M, or *no_improvement* (number of iterations incurred without any improvement in the quality of solution) has reached the given threshhold N, the tabu search would terminate.

In addition, when an infeasible solution outperforms the best-so-far solution, a local search is performed by flipping one bit. This procedure is executed only once to avoid long runtime. This serves as a mechanism to diversify the search and encourage the exploration of new regions in the search space. In algorithm TABU, $\mathbf{x}_{\text{current}}$ indicates the current solution, and $\mathbf{x}_{\text{best\_so\_far}}$ indicates the best-so-far solution. For more details of tabu search, see [25].

Table 1 Summary of the used benchmarks, cited from [22]

Name	n	m	Size	Description
crc32	25	34	152	32-bit cyclic redundancy check. From the Telecommunications category of MiBench [29]
patricia	21	50	192	Routine to insert values into Patricia tries, which are used to store routing tables. From the Network category of MiBench [29]
dijkstra	26	71	265	Computes shortest paths in a graph. From the Network category of MiBench [29]
clustering	150	333	1299	Image segmentation algorithm in a medical application
rc6	329	448	2002	RC6 cryptographic algorithm
random1	1000	1000	5000	Random graph
random2	1000	2000	8000	Random graph
random3	1000	3000	11000	Random graph
random4	1500	1500	7500	Random graph
random5	1500	3000	12000	Random graph
random6	1500	4500	16500	Random graph
random7	2000	2000	10000	Random graph
random8	2000	4000	16000	Random graph
random9	2000	6000	22000	Random graph

It is worthwhile to point out that the solution refined by `TABU` definitely is better than the initial solution provided by `HEUR`. This is because, the best-so-far solution is updated only when `TABU` finds a better local solution according to the line 13 of `TABU`.

5 Experimental results

In [22], three algorithms, named as `Alg-new1`, `Alg-new2` and `Alg-new3`, are proposed. We pick `Alg-new3` to compare with the algorithms presented in this paper, as it works best among the three algorithms. In this section, we utilize `OLD` to indicate `Alg-new3`. As the proposed methods are based on heuristic rather than exact ones, we have to empirically determine their performance and effectiveness. In order to make a fair comparison, our implementations are based on the source codes provided by the author of [22] and on the same benchmarks used in [22]. The characteristics of the test cases are summarized in Table 1, whereby n and m indicate the

Table 2 Parameter setting for tabu search

Parameters	Description	Values
S	Neighborhood size	2000
α	Tabu tenure factor	0.5
β	Tabu tenure factor	1
M	Maximum iteration number	2000
N	Non-improvement threshhold	200

number of nodes and the number of edges in the communication graph, respectively. Also, parameter setting of tabu search are given in Table 2.

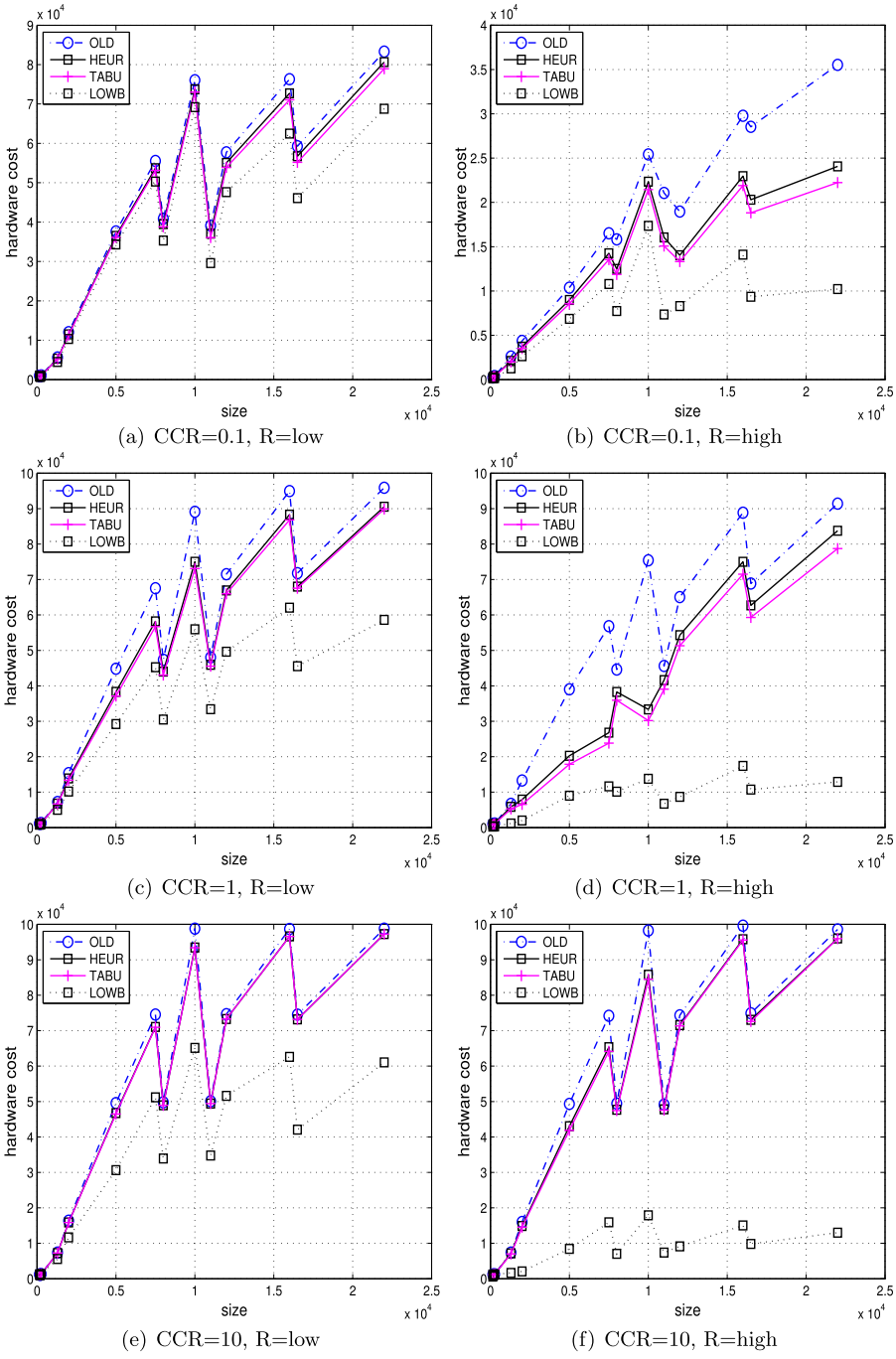
In addition, we have adopted the same experimental setup as in [22], which is summarized as follows:

- The formula $size = 2n + 3m$ is utilized for the size evaluation of the given graph. This is because each node is assigned two values (its hardware and software costs) and each edge is assigned three numbers (the identities of its endpoints and its communication cost).
- Where software costs were not available, they are generated as uniform random numbers from the interval [1, 100]. Where hardware costs were not available, they are generated as random numbers from a normal distribution with expected value $\kappa \cdot s_i$ and a given standard deviation, where s_i is the software cost of the given node. As pointed out in [22], the value of κ only corresponds to the choice of units for software and hardware costs, and thus it has no algorithmic implications.
- The communication costs were generated as uniform random numbers from the interval $[0, 2 \cdot \rho \cdot s_{\max}]$, where $s_{\max}$ is the highest software cost. Thus, communication costs have an expected value of $\rho \cdot s_{\max}$, and ρ is the so-called communication to computation ratio (CCR). ρ was taken as 0.1, 1, and 10, corresponding to computation-intensive case, intermediate (computation-and-communication intensive) case, and communication-intensive case, respectively.
- R was randomly generated as a uniform random number (1) from the interval $[0, \frac{1}{2}\sum s_i]$ (corresponding to the strict real-time constraint), (2) from $[\frac{1}{2}\sum s_i, \sum s_i]$ (corresponding to the loose real-time constraint). The two cases are indicated as $R = \text{low}$ and $R = \text{high}$, respectively.

We tested the proposed algorithms for different values of CCR and constraint R.

Figure 2 shows the quality of the solutions for different constraints and different ratios of communication to computation. In the figure, d is the number of fragments of searched solution space and it is set to d_x [22]. Abscissa represents the size of the problem while the vertical axis indicates the total hardware cost after partitioning. `LOWB` in the figure represents the lower bound on the solution quality, which is calculated using the method proposed in [22]. Since the objective is to minimize hardware costs, smaller values are preferred. Generally, the solutions found by the new algorithms are better than or comparable to the solutions found by `OLD`, especially when the heuristic solution was refined by tabu search.

Table 3 shows the improvements over `OLD` for each case on the used benchmarks. It is evident that the improvements are significant when R is high, i.e., when the



(a) CCR=0.1, R=low (b) CCR=0.1, R=high

(c) CCR=1, R=low (d) CCR=1, R=high

(e) CCR=10, R=low (f) CCR=10, R=high

Fig. 2 Solution quality and lower bound, averaged over 30 instances for different constraints and different ratios of communication to computation, where $d = d_x$

Table 3 Improvements over OLD, averaged over 30 instances

	CCR	R	*imp* (%)
case 1	0.1	low	5.2
case 2	0.1	high	21.7
case 3	1.0	low	9.4
case 4	1.0	high	28.3
case 5	10.0	low	2.3
case 6	10.0	high	6.0

constraint R is relaxed. This is due to the fact that R is analogous to the knapsack capacity. A larger knapsack will provide more opportunity for packing more items into the knapsack, leading to better solutions. To highlight the impact of R in the experimental results, we investigated the distribution of the improvement over the algorithm OLD for a set of instances. Formally, we define *improvement* of algorithm $\mathcal{A}$ over algorithm $\mathcal{B}$ as

$$\left(1 - \frac{hardware\_cost\_of\_\mathcal{A}}{hardware\_cost\_of\_\mathcal{B}}\right) \times 100\ \%.$$

Let *imp* be the improvement of a new algorithm over OLD [22] that is calculated by the formula above. For a given instance, $imp > 0$, $imp = 0$ and $imp < 0$ reflect that the performance of the new algorithm is better than, same as, and worst than the old one, respectively. The *imp* values are collected from -50 % to 50 %, without loss of generality. This corresponds to the distribution interval $[-50, \ldots, -10, -5, 0, 5, 10, \ldots, 50]$ with unit length of 5 in the X-axis shown in Fig. 3. In our statistics,

- if $imp = 0$, we regard the improvement as 0 %.
- If $imp < 0$ and it is in the unit interval $[a, b)$, where $b \leq 0$, we regard the improvement as a%.
- If $imp > 0$ and it is in the unit interval $(a, b]$, where $a \geq 0$, we regard the improvement as b%.

For example, if $-5\ \% \leq imp < 0\ \%$, we view the improvement as -5 %. Our empirical study is based on statistics from 100 random instances. Figure 3 shows that when other parameters remained unchanged, for relatively large R, solutions with high quality can be obtained. For example, in the case of CCR $= 0.1$, $R =$ low, the corresponding improvements are all under 25 %. But in the case of CCR $= 0.1$, $R =$ high, the corresponding distribution interval is $[0, 50]$. It can be observed that the improvement becomes significantly larger when CCR increases to 1 and 10. In summary, the proposed algorithms can produce better solutions with improvement of 28 % on average (see Fig. 2d) and up to 50 % in certain cases, the numerical detail is in Table 4.

As shown in Table 4, tabu search can improve the solution quality in reasonable runtime. In particular, tabu search can achieve high quality results in the order of seconds even for relatively large graphs. Figure 4 shows the refinement of tabu search over our heuristic algorithm. The dotted line represents the case where $R =$ low, and the solid line represents the case where $R =$ high. The lines with the same color have

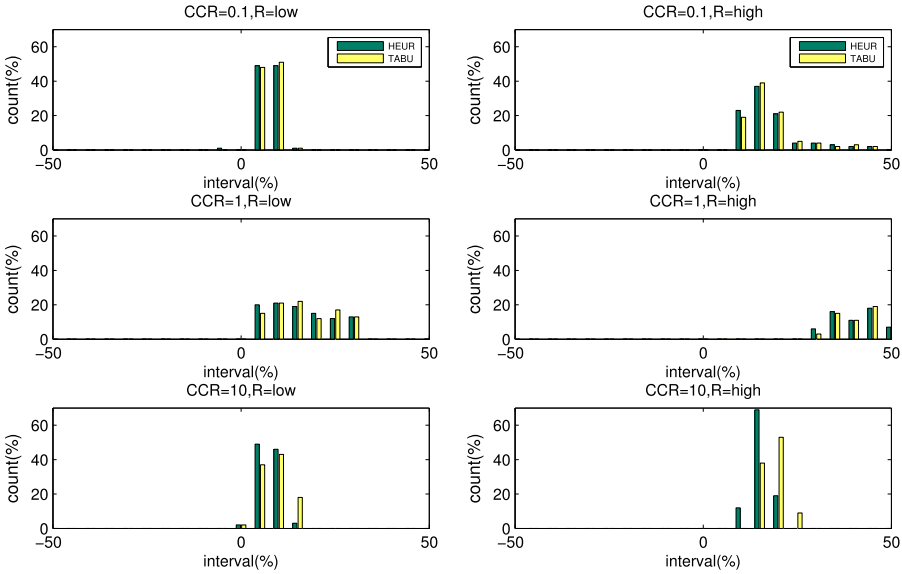


Fig. 3 Distribution of improvements over OLD, 100 random instances on different cases with $size = 5000$ and $d = d_x$

Table 4 Improvements over OLD and runtime for CCR = 1 and R = high, averaged over 100 instances

Benchmark	Solution	Runtime (ms)		
Name	*imp* (%)	OLD	HEUR	TABU
crc32	26.95	0	0	748
patricia	12.15	0	0	998
dijkstra	10.54	0	0	1230
clustering	21.35	1	1	4881
rc6	50.69	1	1	7758
random1	54.19	7	11	18726
random2	58.05	8	23	41465
random3	19.46	9	28	48237
random4	59.95	16	30	48945
random5	14.38	17	44	62902
random6	21.09	19	45	95312
random7	19.54	27	49	103950
random8	13.86	27	58	197080
random9	13.87	32	80	260240

the same CCR. It can be observed that the improvement of tabu search over HEUR increases significantly when R is relaxed.

It is worthwhile to point out that the solution quality of tabu search is heavily influenced by parameter setting. In this paper, we derived a good setting by evaluating

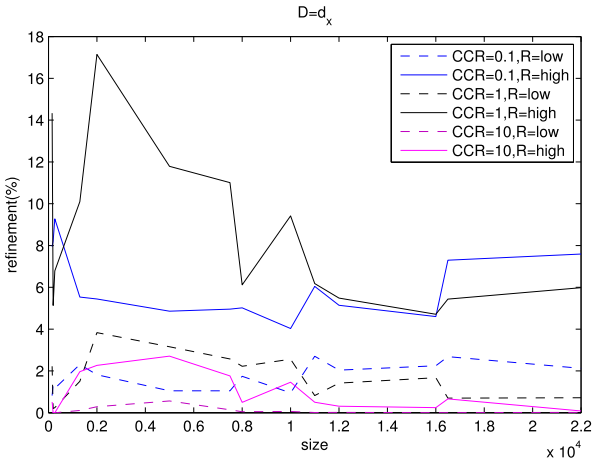
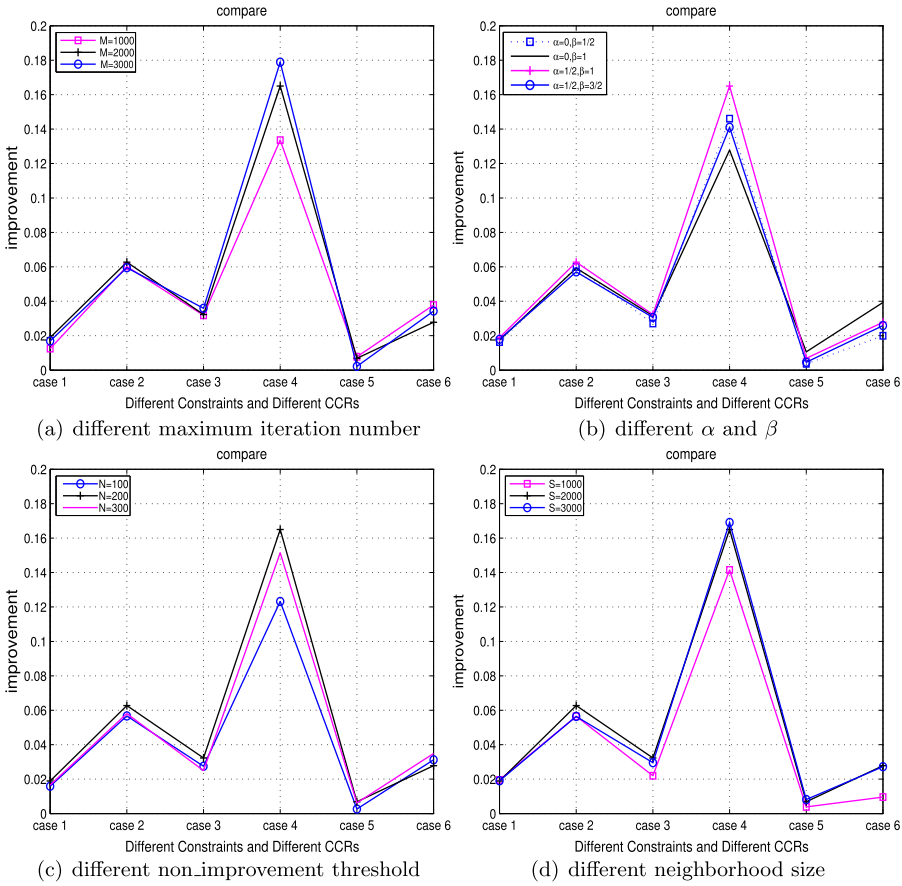


Fig. 4 Refinements of TABU over HEUR for different cases

tabu search in some relatively small graphs in our benchmarks. Figure 5 shows the impact of parameter setting in tabu search on the solution quality. The X-axis shows the different cases of CCR and R as described in Table 3. The Y-axis indicates the results of the refinements by tabu search over HEUR. We can observe that the proposed algorithms work well in case 4, which matches the result as shown in Fig. 2d. Here, Fig. 5a shows that the refinement produces better results for larger M. We pick 2000 as the final value of M, which gives a good trade-off between the solution quality and the runtime. Similarly, we set S to 2000, instead of 3000, according to the refinements shown in Fig. 5d. Figure 5b shows that the refinements lead to better results for the case of $\alpha = 0.5$ and $\beta = 1$. In addition, Fig. 5c shows that the best solution is obtained when $N = 200$. Both of the subfigures demonstrate that our setting for α, β and N, as shown in Table 2, achieves the best results.

6 Conclusions

We have presented a simple but very efficient heuristic algorithm to solve the HW/SW partitioning problem, based on an extended 0–1 knapsack problem. The proposed algorithm is capable of generating good approximate solutions in less than 80 ms even for the largest problem set considered in this paper. We have also customized a tabu search approach to refine the solution of the proposed heuristic algorithm. Experimental results show that both the heuristic approach and the tabu search can provide better solutions than a recently reported method in most of the cases considered (and comparable in the remaining ones). In particular, the improvement of the proposed algorithms over the existing method increases notably for large CCR. This demonstrates that the proposed algorithms can effectively reduce the hardware cost in applications that have large communication overhead. Furthermore, the proposed approaches become more attractive when the constraints are relaxed. The contributions in this paper are based on the same computing model used in [21, 22], whereby the domination relationship between the components is not considered. Our future work



(a) different maximum iteration number (b) different α and β

(c) different non_improvement threshold (d) different neighborhood size

Fig. 5 Refinement over HEUR, averaged over 30 instances with $n = 329$ and $m = 448$

will extend the computing model to take into account the domination relationship and develop the corresponding algorithms for HW/SW partitioning.

Acknowledgements This work was supported by the National Natural Science Foundation of China under Grant No. 61173032.

References

1. Staunstrup J, Wolf WH (1997) Hardware/software co-design: principles and practice. Springer, Berlin
2. Wu J, Srikanthan T, Jiao T (2008) Efficient heuristics for functional partitioning and scheduling in hardware/software co-design. Des Autom Embed Syst 12(4):345–375
3. Onils M, Jantsch A, Hemani A, Tenhunen H (1995) Interactive hardware-software partitioning and memory allocation based on data transfer profiling. In: International conference on recent advances in mechatronics, Istanbul, Turkey, pp 447–452
4. Wu J, Srikanthan T (2006) Low-complex dynamic programming algorithm for hardware/software partitioning. Inf Process Lett 98(2):41–46

5. Niemann R, Marwedel P (1997) An algorithm for hardware/software partitioning using mixed integer linear programming. Des Autom Embed Syst 2:165–193. Special issue
6. Weinhardt M (1995) Integer programming for partitioning in software oriented codesign. In: FPL'95: proceedings of the 5th international workshop on field-programmable logic and applications. Springer, London, pp 227–234
7. Chatha KS, Vemuri R (2002) Hardware–software partitioning and pipelined scheduling of transformative applications. IEEE Trans Very Large Scale Integr (VLSI) Syst 10(3):193–208
8. Ernst R, Henkel J, Benner T (1993) Hardware–software cosynthesis for microcontrollers. IEEE Des Test Comput 10(4):64–75
9. Wu J, Srikanthan T, Yan C (2008) Algorithmic aspects for power-efficient hardware/software partitioning. Math Comput Simul 79(4):1204–1215
10. Niemann R, Marwedel P (1996) Hardware/software partitioning using integer programming. In: Proceedings of the European design and test conference (ED and TC). Paris, France. IEEE Comput Soc, Los Alamitos, pp 473–480
11. Vahid F, Gajski DD (1995) Clustering for improved system-level functional partitioning. In: ISSS'95: proceedings of the 8th international symposium on system synthesis. ACM, New York, pp 28–35
12. Vahid F, Gajski DD, Gong J (1994) A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. In: EURO-DAC'94: proceedings of the conference on European design automation. IEEE Comput Soc, Los Alamitos, pp 214–219
13. Quan G, Hu X, Greenwood GW (1995) Preference-driven hierarchical hardware/software partitioning. In: ICCD'99: proceedings of the 1999 IEEE international conference on computer design. IEEE Comput Soc, Washington, pp 652–657
14. Wiangtong T, Cheung PYK, Luk W (2002) Comparing three heuristic search methods for functional partitioning in hardware-software codesign. Des Autom Embed Syst 6(4):425–449
15. Shuang D, Shan D, Shi Z, Liucun Z (2010) GA-based Algorithm for Hardware/Software Partitioning with Resource Contentions. In: IEEE International Conference on Advanced Computer Control, vol 1, pp 68–72
16. Wu J, Srikanthan T, Ting L (2010) Efficient heuristic algorithms for path-based hardware/software partitioning. Math Comput Model 51(7–8):974–984
17. Abdelhalim MB, Habib SE-D (2011) An integrated high-level hardware/software partitioning methodology. Des Autom Embed Syst 15(1):19–50
18. Mu J, Roman L (2009) Autonomous hardware/software partitioning and voltage/frequency scaling for low-power embedded systems. ACM Trans Des Autom Electron Syst 15(1):2. doi:10.1145/1640457.1640459
19. Yuan M, Gu Z, He X (2010) Hardware/software partitioning and pipelined scheduling on runtime reconfigurable FPGAs. ACM Trans Des Autom Electron Syst 15(2):1–41
20. Cui L (2012) A novel approach to hardware/software partitioning for reconfigurable embedded systems. J Comput 7(10):2518–2525
21. Arato P, Mann ZA, Orban A (2005) Algorithmic aspects of hardware/software partitioning. ACM Trans Des Autom Electron Syst 10(1):136–156
22. Wu J, Srikanthan T, Chen G (2010) Algorithmic aspects of hardware/software partitioning: 1D search algorithms. IEEE Trans Comput 59(4):532–544
23. Martello S, Toth P (1990) Knapsack problems: algorithms and computer implementations. Wiley, New York
24. Knuth DE (1998) The art of computer programming, 2nd edn. Sorting and searching, vol 3. Addison-Wesley, Reading
25. Glover F, Laguna M (1997) Tabu search. Kluwer Academic, Boston, pp 10–150
26. Gendreau M, Iori M, Laporte G (2008) A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. Networks 51(1):4–18. Special issue
27. Benlica U, Hao J (2011) An effective multilevel tabu search approach for balanced graph partitioning. Comput Oper Res 38(7):1066–1075
28. Fan W, Machemehl R (2008) A Tabu Search based heuristic method for the transit route network design problem. In: Computer-aided Systems in Public Transport, vol 600, pp 387–408. Part 4
29. Guthaus M, Ringenberg J, Ernst D et al (2001) Mibench: a free, commercially representative embedded benchmark suite. In: 4th IEEE international workshop on workload characteristics, pp 3–14