Knapsack Model and Algorithm for HW/SW Partitioning Problem

Abhijit Ray, Wu Jigang, and Srikanthan Thambipillai

Centre for High Performance Embedded Systems,
School of Computer Engineering, Nanyang Technological University,
Singapore,639798, Republic of Singapore
{PA8760452, asjgwu, astsrikan}@ntu.edu.sg

Abstract. Hardware software partitioning is one of the most significant problem in embedded system design. The size of the total solution space for this problem is typically quite large. This problem has been investigated extensively. This paper is the first work to model the problem into a knapsack problem. We present a way to split the problem into standard 0-1 knapsack problems, so that most of the classical approaches for 0-1 knapsack problems can be directly applied. We use tight lower bound and tight upper bound on each of these knapsack problems to eliminate sub-problems, which are guaranteed not to give optimal results.

1 Introduction

Hardware software partitioning (HSP) problem is the problem of deciding for each subsystem, whether the required functionality is to be implemented in hardware or software to get the desired performance, while maintaining least cost. At the same time, hardware area minimization and latency constraints present contradictory objectives to be achieved through hardware-software partitioning.

Most of the existing approaches to HSP are based on either hardware oriented partitioning or software oriented partitioning. A *software-oriented* approach means that initially the whole application is allotted to software and during partitioning system parts are moved to hardware until constraints are met. In a *hardware-oriented* approach on the other hand, the whole application is implemented in hardware and during partitioning the parts are moved to software until constraints are violated. A software oriented approach has been proposed by Ernst *et al* [3], Vahid *et al* [4]. Hardware oriented approach has been proposed in Gupta *et al* [5], Niemann *et al* [6]. In [9], the authors proposed a flexible granularity approach for hardware software partitioning. Karam *et al* [7], proposes partitioning schemes for transformative applications i.e., multimedia and digital signal processing applications. The authors try to optimize the number of pipeline stages and memory required for pipelining. The partitioning is done in an iterative manner. Rakhmatov *et al* [8] modeled the hardware software partitioning as a unconstrained bipartitioning problem.

In this paper, we model the partitioning problem into some standard knapsack problems, which can be solved independently to arrive at the solution. Also for every subproblem we calculate the lower bound and its upper bound, this helps in rejecting subproblems, which are not expected to give optimal results. The advantage of our work is that it provides the optimal solution. Moreover, many problems are rejected based on their lower bound and upper bound, and this reduces the number of subproblems that need to be solved; hence the algorithm is quite fast.

2 Model of the Physical Problem

We consider a basic case which can be later extended. In our case the application can be broken down into parts such that each of them can be run simultaneously or in other words the parts do not have any sort of data dependency between them. So we have a set of items $S = \{p_1, p_2, \ldots, p_n\}$ to be partitioned into hardware and software. Let h_i and s_i be the time required for the part p_i to be run in hardware and software respectively. Also let a_i be the area required for hardware implementation of part p_i. And let A be the total area available for hardware implementation. Our goal is to allot each part into hardware and software so that the combined running time of the whole application is minimized in such a way that the area constraint is satisfied. Let us denote the solution of the problems as a vector $X = [x_1, x_2, \ldots, x_n]$ such that $x_i \in \{0, 1\}$, where $x_i = 0\,(1)$ implies that the part p_i is implemented in software (hardware). Since the hardware and software can be run in parallel, the total running time of the application is given by

$$T(X) = max\{H(X), S(X)\} \tag{1}$$

where $H(X)$ is the total running time of the parts running in hardware and $S(X)$ is the total running time of the parts in software. Since all the parts that are implemented in hardware can be run in parallel to each other and all the software parts has to be run in serial, we have

$$H(X) = \max_{1\le i\le n}\{x_i \cdot h_i\} \quad \textit{and} \quad S(X) = \sum_{i=1}^{n}(1 - x_i)\cdot s_i.$$

Hence, the problem discussed in this paper can be modeled into

$$\mathcal{P} \begin{cases} \text{minimize } T(X) \\ \text{subject to } \sum_{i=1}^{n} x_i \cdot a_i \le A \end{cases} \tag{2}$$

3 Problem Splitting and Algorithm

Given a knapsack capacity C and set of items $S = \{1, \ldots, n\}$, where each item has a weight w_i and a benefit b_i. The problem is to find a subset $S' \subset S$, that maximizes the total profit $\sum_{i \in S'} b_i$ under the constraint that $\sum_{i \in S'} w_i \leq C$. This is the knapsack problem. Mathematically, it can be described as follows

$$\text{0-1 KP} \begin{cases} \text{maximize} \sum\limits_{i=1}^{n} p_i \cdot x_i \\ \text{subject to} \sum\limits_{i=1}^{n} w_i \cdot x_i \leq C, \\ \qquad\qquad x_i \in \{0,1\}, i = 1, \dots n \end{cases} \tag{3}$$

where x_i is a binary variable equalling 1 if item i should be included in the knapsack and 0 otherwise. It is well known that this problem is NP-complete [2, 1, 10].

Let's assume that the items are ordered by their efficiencies in a non-increasing manner, where the efficiency is defined as

$$e_j = \frac{b_j}{w_j} \tag{4}$$

Let

$$\overline{b_j} = \sum_{i=1}^{j} b_i \quad \textit{and} \quad \overline{w_j} = \sum_{i=1}^{j} w_i \quad j = 1, 2, \dots, n \tag{5}$$

The residual capacity r is defined as

$$r = C - \overline{w}_{t-1} \tag{6}$$

By linear relaxation, [2] showed that an upper bound on the total benefit of 0-1 KP is

$$u = \lceil \overline{b}_{t-1} + r \cdot \frac{b_t}{w_t} \rceil \tag{7}$$

and the lower bound is given by,

$$l = \overline{b}_{t-1} \tag{8}$$

In HSP problem, sort all the items $p_1, p_2, \dots, p_n$ in decreasing order of their hardware running time, so that after sorting we have the items ordered as $p'_1, p'_1, \dots, p'_n$ and the following condition is satisfied

$$h'_i \geq h'_j \qquad \text{for all} \quad i \leq j \quad \text{and} \quad 0 \leq i, j \leq n. \tag{9}$$

Let us define $S_T = \sum\limits_{i=1}^{n} s'_i$ and $R_i = S_T - s'_i$. Now we split the problem $\mathcal{P}$ into the following n subproblems $\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n$.

3.1 Subproblem $\mathcal{P}_k$:

Let $k \geq 1$. We fix p'_k to be implemented in hardware i.e., $x_k = 1$, and all the items $1, 2, \ldots, k-1$ are in software, because if any of them, say j is in hardware then any subproblem l such that $l > j$ is a subset of subproblem j. That is we have $x_1 = 0, x_2 = 0, \cdots, x_{k-1} = 0$. The total time is

$$T(X) = max\left\{h'_k, R_k - \sum_{i=k+1}^{n} x_i \cdot s'_i\right\}$$

We have to minimize the total running time $T(X)$, i.e.,

$$\begin{aligned} minimize \quad T(X) &\Longleftrightarrow minimize\left\{R_k - \sum_{i=k+1}^{n} x_i \cdot s'_i\right\} \\ &\Longleftrightarrow maximize\left\{\sum_{i=k+1}^{n} x_i \cdot s'_i\right\} \end{aligned}$$

subject to the constraint $x_i \in \{0, 1\}$ and the area constraint

$$\sum_{i=k+1}^{n} x_i \cdot a_i \leq A - a_k \tag{10}$$

Formally, the subproblem k is described as

$$\mathcal{P}_k \quad \begin{cases} \text{maximize } \sum_{i=k+1}^{n} x_i \cdot s'_i \\ \text{subject to } \sum_{i=k+1}^{n} x_i \cdot a_i \leq A - a_k \end{cases} \tag{11}$$

The bounded interval of subproblem $\mathcal{P}_k$ are

$$L_k = \max\{h'_k, R_k - u_k\},$$
$$U_k = \max\{h'_k, R_k - l_k\},$$

and the optimal solution of $\mathcal{P}_k$ would lie in the range $[L_k, U_k]$ where l_k and u_k are the lower and upper bound of total benefit of $\mathcal{P}_k$, respectively.

If after creating subproblem i, we find that

$$\sum_{j=i+1}^{n} a_j \leq A - a_i \tag{12}$$

This means that after we have fixed items a_i to be implemented in hardware and $a_1, a_2, \ldots, a_{i-1}$ into software, the rest of the items left to be partitioned can be implemented easily in hardware as there are enough hardware space left. Hence, we can stop creating more subproblems as soon as eq[12] is satisfied.

A point to be noted is that all subproblems are not of the same size. This is because for subproblem P_i, we fix item i to be implemented in hardware and all the items $1, 2, \ldots, i-1$ are in software. This is because if in subproblem P_i, item k, $\{k < i\}$ is implemented in hardware then it becomes a subset of subproblem P_k. Therefore the size of the problem decreases from subproblem 1 to n.

3.2 Algorithm

The outline of the algorithm for solving the HSP problem is given below:

```
BOUND := 0;
sort all the items to be partitioned in decreasing order of
their hardware running time;
form the subproblems P(i), i=1, 2,...,n;
for(i:=1 ; i <= n ; i++ ){
    calculate the upper bound U(i)
        and the lower bound L(i) for P(i);
    if( L(i) > BOUND){
        BOUND := L(i);
    }
}
while( there are subproblems left to be solved){
    select the subproblem with the highest lower bound;
    if( U(i) < BOUND ){
        reject this subproblem;
    }else {
        solve this subproblem;
        B(i): = benifit of the above solution;
        if ( B(i) > BOUND ){
            BOUND := B(i);
        }
    }
}
```

4 Experimental Works

The proposed algorithm was implemented on a Pentium, 500MHz system, running on Linux. We used random data for partitioning. For solving the individual 0-1 knapsack problems we used the algorithm for 0-1 knapsack problem, given in [1]. The experiment was performed for different problem sizes and area constraints. Table 1 gives a count of the number of subproblems that needed to be solved to arrive at the optimal solution for the whole problem.

5 Conclusion

We have proposed a algorithm for hardware/software partitioning problem. The proposed algorithm models the problem into the Knapsack problem, which is a known NP-complete problem, and then splits the whole problem into some independent sub-problems. Upper and lower bounds of each subproblem is used to reject some sub-problems. As a result, fewer subproblems needs to be solved.

Table 1. Number of subproblems solved for different problem size and area constraints.

size	fraction of area put as constraint								
n	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
30	1	3	1	1	1	1	1	1	1
60	1	1	1	1	1	1	1	1	1
90	1	1	1	1	1	1	1	1	1
300	1	1	1	1	1	1	1	1	1
500	1	1	1	1	1	1	1	1	1
700	1	1	1	1	1	1	1	1	1
1000	1	1	1	1	1	1	1	1	1
2000	1	1	1	1	1	1	1	1	1
3000	1	1	1	1	1	1	1	1	1

Also, since the subproblems can be solved in parallel, this approach can be effectively used in a distributed computing environment. We are currently extending our knapsack model to consider cases when the items to be partitioned are not independent and hence communication between the items to be partitioned becomes an issue.

References

1. D. Pisinger,*A minimal algorithm for the 0-1 knapsack problem*, Operations Research, 1997,Page(s): 758-767.
2. D. Pisinger, *Algorithms for knapsack problems*, Ph.D. Thesis, 1995, Page(s):1-200.
3. R. Ernst, J.Henkel and T. Benner,*Hardware-Software Cosynthesis for Microcontrollers*,IEEE Design and Test of Computers, 1993, Page(s):64-75.
4. F.Vahid, D.D. Gajski and J. Jong, *A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning*, IEEE/ACM Proceedings European Conference on Design Automation(EuroDAC), 1994, Page(s):214-219.
5. R.K. Gupta and G.D. Micheli, *System-level synthesis using reprogrammable components*, Proceedings. [3rd] European Conference on Design Automation, 1992, Page(s):2-7.
6. R. Niemann and P. Marwedel, *Hardware/software partitioning using integer programming*, Proceedings European Design and Test Conference, 1996. ED&TC 96, Page(s):473-479.
7. K.S. Chatha and R. Vemuri, *Hardware-software partitioning and pipelined scheduling of transformative applications*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , Volume: 10 Issue: 3 , Jun 2002, Page(s):193-208.
8. D.N. Rakhmatov and S.B.K. Vrudhula, *Hardware-software bipartitioning for dynamically reconfigurable systems*, Hardware/Software Codesign, 2002. CODES 2002. Proceedings of the Tenth International Symposium on, Page(s):145-150.
9. Jorg Henkel and Rold Ernst, *An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation technique*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , Volume: 9 Issue: 2 , Apr 2001, Page(s):273-289.
10. W. Jigang, L. Yunfei and H. Schroeder, *A minimal reduction approach for the collapsing knapsack problem*,Computing and Informatics, Volume: 20, 2001,Page(s): 359-369.