Microprocessors and Microsystems 39 (2015) 259–270

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Hardware software partitioning of control data flow graph on system on programmable chip

Mehdi Jemai *, Bouraoui Ouni

Laboratory Electronic and Microelectronic, The National Engineering School of Monastir, University of Monastir, Monastir 5000, Tunisia

ARTICLE INFO

Article history:
Available online 11 May 2015

Keywords:
Hardware–software partitioning
SOPC
Control data flow graph
Co-design

ABSTRACT

A System On Programmable Chip (SOPC) is a circuit that integrates all components of an electronic system into a single chip. It may consist of memories, one or more microprocessors, interface devices, configurable logic blocks and other necessary components to achieve an intended function. In this paper, we propose a new hardware–software partitioning algorithm of control data flow graph for SOPC. The main aim of our algorithm is to find a best compromise between hardware and software implementation of operations in order to satisfy design constraints in terms of latency and hardware resources of the target application. Our algorithm has been evaluated on real hardware device. In fact, experimentations have been done using a real FPGA Virtex-5. Results have shown that our algorithm provides a better performing system with the lowest possible cost compared to existing approaches.

1. Introduction

Modern FPGA has become much more sophisticated than before. It can hold central processing unit (CPU) and several RAMs and DSPs at the same time. In fact, new FPGA device (such as Xilinx Virtex-family) includes two subsets of resources: software resources and hardware one. Software resources may be one or more hard processors or DSP (example IBM PowerPC of Xilinx, AVER of Atmel). The hardware resources may be transceivers, analog-blocks, multiply–accumulate modules (MACs), RAMs blocks and Configurable Logic Blocks (CLBs). Therefore, current FPGA which is a programmable System On Chip (SOC); is called System On Programmable Chip (SOPC). Nowadays, several designers prefer to use SOPC to implement their applications. These applications should meet design constraints like performance, flexibility, and time to market. Often, co-design efficiency has been related to hardware–software partitioning task. Hardware–software partitioning is a system-level partitioning problem. It aims to assign operations of the application to the hardware part or to the software part of the SOPC in order to obtain a faster treatment with lowest cost. In this paper, we aim to solve the following problem:

Given a control data flow graph and SOPC circuit; find a possible hardware–software partitioning of a graph on the SOPC in order to get a better compromise between hardware resources used to implement the target graph and its whole latency.

Our algorithm is based on a function called *generating of partition object*. At each algorithm iteration, this function returns a sub-graph called partition object of the original graph. Next, we compute the hardware–software latency and the hardware area cost of each generated partition object. This procedure will be repeated until the hardware–software latency of one among the generated partition objects closes to its hardware area cost. In other words, the mentioned function generates many partition objects (e.g.: 1% hardware 99% software, 25% hardware 75% software, etc.). For each partition object, the latency value will be calculated and a curve will be drawn using these different values. Then, the area cost will be computed as well and a second curve will be produced. When these two curves will be superimposed, a unique intersection point will be obtained. That point refers to the best partition object that should be used.

2. Related works

In the literature, many designers have proposed hardware–software partitioning algorithm for (SOC) System On Chip [1,2]. In previous works, hardware–software partitioning was carried out manually [3]. However, in reality hardware–software partitioning is more complicated and many requirements on: cost, hardware effort, power dissipation and timing performance have to be taken into consideration. So, efforts have been increased in order to automate the hardware–software partitioning task. For that purpose

* Corresponding author.
E-mail addresses: jmehdie@gmail.com (M. Jemai), ouni_bouraoui@yahoo.fr (B. Ouni).

http://dx.doi.org/10.1016/j.micpro.2015.04.006
0141-9331/

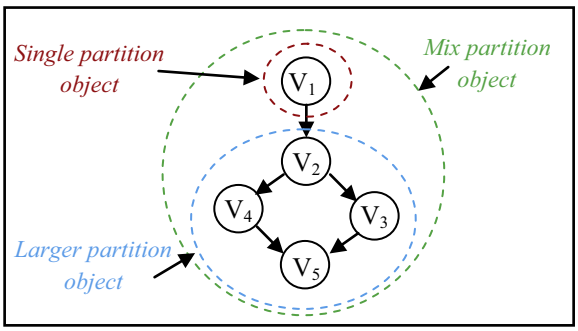

Fig. 1. The three kind of partitions object.

many optimization methods have been used to come up with new algorithms such as exact algorithms that are based on: integer linear programming [4], dynamic programming [5] and branch-and bound [6]. However, exact algorithms are very slow and can be applied only for small size graphs. Hence, to overcome its drawbacks, researchers have turned to more flexible and efficient heuristic algorithms. Traditional heuristic algorithms are software-oriented and hardware-oriented. The software-oriented approach means that the initial implementation of the whole application is supposed to be a software solution. Next, during the partitioning, the operations of the application are migrated to hardware until constraints are met [7]. Moreover, other hardware–software partitioning simulated annealing [8–11],

```
1: Begin
2: For all  v_i ∈ V
3:   Compute ST ( v_i)
4:   Compute J ( v_i)
5: End for
-: //– Generating of single partitions object //
6: k= 1; k ∈ ℕ
7: For all  v_i ∈ V :
8:  If ∀_{v_i∈V} (ST ( v_i) ≥ 1 & (J ( v_i) = 0)) then
9:    P_k <= { v_i }   //  P_k is k^th  single partition object //
10:   P_OG<=P_k;   // P_OG the set of partitions object //
11:   k= k+1;
12:  End if;
13: End for
14: m=k ; m ∈ ℕ
-: //– Generating of large partitions object //
15: For all  v_i ∈ V :
16:  If ∀_{v_i , v_j ∈V} ((J ( v_i) = 1) then
17:   While  (J ( v_j)  ≠ 1)) loop
18:    P_m <= { v_i , ... , v_j} ; // P_m is m^th  large partition object //
19:   End loop
20:   P_OG <=P_m; // P_OG the set of partitions object //
21:   m= m+1;
22:  End if;
23: End for
24: z=m; z ∈ ℕ
-: //– Generating of mix partitions object //
25: While (P_i ≠{V}) loop
26:  For all partition object P_i ∈ P_OG
27:   If ∀_{v_i ∈V} (J (pred (P_i) = 0) then //  pred (P_i) the predecessor of the source node of the i^th large partition //
28:    P_z<= { pred(P_i) ; P_i}: // P_z is z^th  mix partition object //
29:    P_OG <=P_z  // P_OG the set of partitions object //
30:    z= z+1;
31:   else
32:    P_z<= {P_i-1 , P_i } ;
33:    P_OG <=P_z // P_OG the set of partitions object //
34:    z= z+1
35:   End if;
36:   If ∀_{v_i ∈V} , (J (succ (P_i)=0) then // succ(P_i) the successor of the sink node of the i^th large partition
37:    P_z<= { P_i ; succ(P_i) }  ;
38:    P_OG <=P_z  // P_OG the set of partitions object //
39:    z= z+1;
40:   else
41:    P_z<= {P_i; P_i+1};
42:    P_OG <=P_z // P_OG the set of partitions object //
43:    z= z+1;
44:   end if ;
45:   P_z<= {P_z-2; P_i; P_z-1\P_i};
46:   P_OG <=P_z  // P_OG the set of partitions object //
47:   z= z+1;
48:  End for;
49: End loop;
50: End begin
```

Fig. 2. Pseudo-code of generating of partitions object function.