Performance Improvement through Path-Based Partitioning in Hardware/Software Co-Design

Elham Azari[1] and Hakduran Koc[2†]
Elham.Azari@asu.edu KocHakduran@uhcl.edu
Arizona State University, Tempe, AZ, USA , University of Houston-Clear Lake, Houston, TX, USA,

Summary
In co-design of an embedded system, hardware/software partitioning has always been a crucial step. Efficient partitioning improves the overall performance of a system significantly. As allocating tasks to either hardware or software components has its own advantages and disadvantages, it typically becomes necessary to tradeoff among the main design metrics such as performance and area. This paper proposes a new approach in partitioning the tasks in a given Control Data Flow Graph (CDFG) to enhance the performance while meeting the area constraint. In order to effectively perform partitioning phase of the co-design, the combination of two main paths are considered: hot path and critical path. These two paths dominate the total execution time of a system. The target co-design architecture consists of two CPUs and two ASICs with different execution time for each task. This paper partitions the hot path and the critical path, and tries to assign as many tasks as possible to the ASICs by giving higher priority to the tasks in the hot paths which directly have significant effect on the critical path. Consequently, the total execution time of a given application is reduced. This, in turn, improves the overall performance without degrading other implementation metrics such as power and reliability.
The experimental results collected in this research indicate that the proposed path-based partitioning method on the co-design architecture improves the performance significantly.

Key words:
Co-design architecture, hardware/software partitioning, CDFG, hot path, critical path.

1. Introduction

Hardware/software partitioning is a crucial step in co-design which is considered as part of the modeling in the main co-design steps. Partitioning allocates each task (often referred to as basic blocks) in a program to either hardware or software components. Partitioning is an NP problem, therefore, efficient and more flexible heuristic algorithms are used to overcome this problem. Although these heuristic algorithms give approximate optimal results, they reduce the partitioning time significantly. The purpose of this research is to explore and present a partitioning method on a given CDFG to assign the basic blocks to either hardware or software units. The target co-design architecture is considered as two ASICs and two CPUs as hardware and software units, respectively. This method improves the performance while the given area constraint is met. Our hypothesis is that, instead of targeting all the basic blocks in a given CDFG, path-based partitioning which targets the important paths can enhance the performance significantly. To implement this hypothesis, in this work, the combination of hot path(s) and critical path are considered as the main target. These two paths have significant effect on the performance of a given application since they dominate the total execution time of an application. The hot path(s) is part of a control path that includes the tasks executed more frequently in a given Control Flow Graph (CFG) such as loops. On the other hand, the critical path is the longest path that determines the shortest time possible to execute the tasks in an embedded application. Since the tasks in a hot path are executed many times at runtime, it has an enormous effect on partitioning the whole application as well as the critical path.
To identify hot paths in a given program, there are advanced profiling techniques-edge weight profiling [1] [2] and Ball-Larus path profiling [3] to name a few. In this work, Ball-Larus path profiling is used which uses the execution frequency assigned to each edge and also saves how often each path in a CFG is executed.
This paper is organized as follows. Section 2 discusses the related work on general and path-based partitioning algorithms and approaches. Section 3 describes the main concepts of our method. Section 4 presents the target architecture used in this work. Section 5 describes the operation of the proposed algorithm and illustrates its operation by an example. Section 6 provides the experimental results collected by comparing our approach with two other methods and at the end, section 7 concludes our work.

2. Related Work

There are many notable research efforts on heuristic and non-heuristic hardware/software partitioning algorithms design in the literatures such as greedy based algorithms, simulated annealing algorithm [4], dynamic programming algorithm [5], genetic algorithm [6] [7], Tabu search [8] [9] [10], Ant Colony Search algorithm [11] [12], Particle

Manuscript received May 5, 2020
Manuscript revised May 20, 2020

Swarm Optimization [13] [14], as well as their improved approaches. Our method uses the hot paths in a graph as one of the important paths in effecting the execution time of a system. There are several path profiling techniques which identify the important paths in a graph. For instance, R. Cohn and G. Lowney [15] use the Hot-Cold Optimization (HCO) technique to partition each program into frequently executed (hot) and infrequently executed (cold) parts. D. Ung and C. Cifuentes [16] propose an algorithm for finding hot paths using edge weight profiles. Yasue et al. [17] present an efficient online path profiling technique, called structural path profiling (SPP) suitable for Just-In-Time compilers. Vaswani et al. [18] introduce a hardware path profiling scheme which is able to detect several types of paths including the profile of hot paths.

Our method is a path-based partitioning method which combines two major paths, namely, critical paths and hot paths. There are some algorithms presented in which they pay attention to critical path for partitioning the tasks. H. Wang and H. Zhang [19] introduce a guiding function in order to improve the efficiency of the greedy algorithm in solving the partitioning problem. It considers the tasks on the critical path as one of the priorities to maximize the performance. G. Khan and M. Jin [20] present a new graph model suitable for partitioning heterogeneous systems. It targets the nodes on the critical path and tries to assign them to the hardware components in order to improve the performance and minimize the hardware area. In hardware/software partitioning part, Lo et al. [21] introduce a solution to calculate a total system execution time which covers hardware and software execution times and the data transfer time between nodes on the critical path. On the other side, some papers present their work by giving the priority to the tasks on hot paths. Jiang et al. [22] propose an efficient heuristic algorithm for hardware/software partitioning on the selected nodes in a hot path. It considers all types of the communications between neighboring blocks using 0-1 knapsack problem [23] with the main objective of minimizing the execution time with a given area constraint. W. Jigang and S. Thambipallai [24] introduce a branch and bound algorithm to partition the hot path selected by path profiling techniques. The proposed method is applied on a Data Flow Graph (DFG) divided in to basic blocks.

Zhang et al. [26] propose a new swarm intelligence optimization algorithm after analyzing disadvantages of conventional firework algorithm in order to improve the optimization accuracy and decrease the time consumed. They use a new selection strategy to accelerate the convergence speed of the algorithm. Hassine et al. [27] present another algorithm based on HW/SW partitioning to investigate the best tradeoff between power and latency of a given system considering the dark silicon problem. Their algorithm favorable results compared to well-known simulated annealing and genetic algorithms. Yin et al. [28] propose a partitioning algorithm for resource-constraint embedded security systems. They formulate the partitioning problem as a 0-1 Knapsack problem using a modified simulated annealing approach. Iguider et al. [29] focus on non-functional requirements of modern embedded systems while making sure the design meets the functional specifications considering area and execution time constraints. Their approach is based on minimax algorithm and provides more optimal solutions compared to genetic algorithm.

Azari and Koc [30] presents an approach to partition the tasks in a given Control Data Flow Graph (CDFG) representing an application. They consider the combination of two main paths (hot path and critical path) in the system during the partitioning phase of the co-design as these two paths dominate the total execution time. This paper extends the approach presented in [30]. Hou et al. [31] provide a survey of three important aspects of hardware/software partitioning, namely, partitioning models, partitioning algorithms, and parallel algorithms for hardware/software co-design along with possible research directions. In the existing literatures, most path-based partitioning algorithms use either of the hot path or critical path. Since both of these paths significantly affect the total execution time of a system, we use and combine these paths in our method in order to partition the tasks more efficiently. This in turn provides significant improvement in performance.

3. Details of Our Approach

Our approach is based on the combination of two main paths in a CDFG: hot path and critical path. These two paths and the various combinations between them in a CDFG are discussed in the following sections.

3.1 CDFG

Our proposed approach is presented on a CDFG. CDFG is a directed graph that presents the data dependencies in addition to the control dependencies. In general, directed edges in a CDFG determine the execution order of the nodes. Specifically, the control edges represent the transfer of a value or control from one node to another and the data edges specify the data dependency between two individual nodes. An edge can be conditional, representing a condition while implementing an if/case statements or loop constructs. In CDFGs, there are two types of nodes: data flow nodes (also known as basic blocks) and decision nodes. Data flow nodes are pieces of code with no condition, one entry, and one exit point; and, decision nodes are pieces of code with at least one condition.

Figure 1 illustrates a CDFG that is generated from a given specification in C language. The nodes in rectangular form represent the data flow nodes and the nodes in diamond represent the decision nodes. The reason for using this type of graph in our approach is that we need all the paths traversed in a program along with the data flow nodes. These specifications are found in CDFG. The process of translation starts by first deriving a CDFG from the source code of a given specification.

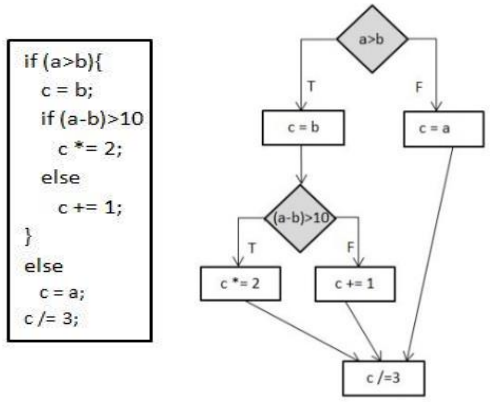
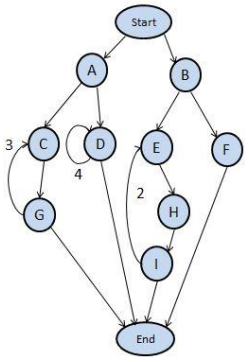


Fig.1. A CDFG generated using the high level code on the left.

3.2 Scheduling CDFG

Scheduling CDFG is different from scheduling DFG that considers only the data dependencies, since all the paths including the loops must be considered. Therefore, for scheduling a CDFG, the first step is to remove all the back edges which indicate the loops. To do so, a loop duplication technique [25] is used in the proposed approach. As the name suggests, it duplicates the tasks in the loops with respect to the order of the tasks and the number of the iterations. The scheduling steps are shown in an example. Therefore, for scheduling a CDFG, the first step is to remove all the back edges which indicate the loops. To do so, a loop duplication technique [25] is used in the proposed approach. As the name suggests, it duplicates the tasks in the loops with respect to the order of the tasks and the number of the iterations. The scheduling steps are shown in an example.

Figure 2 depicts a CDFG with 9 tasks along with their execution time. The execution time of the start and exit nodes are assumed to be zero. In this graph, there are four separate paths, namely, ACG, AD, BEHI, and BF from the start node to the exit node. For simplicity, in this graph we eliminated the decision nodes. Each node is a data flow node and each edge represents the control dependencies. The back edges in this graph indicate loops with the number of iterations, e.g. in path ACG, nodes C and G will be iterated three times before the path finishes its execution time. In our examples there are no data dependencies between each path, so we schedule each path separately. First, we use the loop duplication technique explained above to remove all the back edges. Figure 3 illustrates the CDFG after the loop duplication, e.g. in the path BEHI, since the number of iterations for the loop EHI is 2, nodes E, H, and I are duplicated once and added right after the first EHI nodes with respect to their order. All the four paths in Figure 3 are converted to ACGCGCG, ADDDD, BEHIEHI, and BF respectively. Now, there are 4 separate paths with no back edges and dependencies, so we can schedule each of them separately. In order to find the latency of the graph, the execution time of all the nodes in each path after duplication will be added together, and the longest execution time among all the paths will be the latency of the graph. In Figure 4, each path is separated and the edges are annotated with the execution time of the nodes. After calculation, we can see that path AD has the total execution time of 18 clock cycles that is higher than the paths ACG, BEHI, and BF with the execution times of 17, 14, and 3 clock cycles, as a result, the longest path is AD and the latency of the graph is 18 clock cycles.



Node	ET (CC)
A	2
B	2
C	3
D	4
E	2
F	1
G	2
H	1
I	3

Fig.2. A CDFG with 9 tasks along with their execution times.

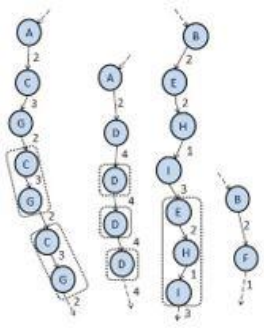


Fig.3. loop duplication on the CDFG

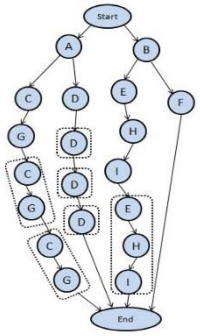


Fig.4. Scheduled CDFG

Path AD has higher execution time in comparison to the paths ACG and BEHI, although it has fewer tasks. This indicates that the latency of a CDFG does not depend on

the number of the nodes in each path; it depends on the execution times of the tasks and the number loop iterations.

3.3 Hot Path

A hot path is part of a control path that includes the tasks (sequence of instructions) executed more frequently in a given CFG, such as body of a loop since it is iterated many times, as shown in Figure 5. Consequently, the other tasks which are not part pf the hot path are considered as cold path. Hot paths dominate the total execution time of an application and are used for quick access to frequently visited tasks. The hot paths have an enormous effect on partitioning of the whole application; therefore, the whole execution time of an application can be improved by optimizing the hot path. As a result, this improves the overall performance of an application significantly. The importance of hot path is that the nodes on this path are highly executed, which has a direct effect on the total execution time. Within a program, there may be numerous hot paths. The program might switch to different hot paths or cycle between them. Therefore, for a specific run-time of a program, each hot path can capture the most frequent execution pattern. If a program's behavior changes, typically, new hot paths will be created. Since most of the programs spend most of their execution time in small portions of the code, identifying the hot path will often result in boosting the execution speed. In addition, collecting hot paths is important for optimizing the target program effectively.

In order to identify a hot path in a program, there are advanced profiling techniques. In the proposed method, Ball-Larus path profiling technique is used which uses the execution frequency assigned to each edge and also save how often each path in a CFG is executed. This profiling technique transforms a given CFG containing loops into an acyclic graph with a limited number of paths.

3.4 Critical Path

Critical path is the longest path that determines the shortest time possible to execute the tasks in an embedded application. In other words, the critical path is a sequence of connected tasks from the start to the exit node that will take the longest to complete. As shown in figure 6, path ABDG has longest execution time in comparison to the path ACEFG, thus, path ABDG is the critical path in this CDFG. The importance of this path is that the nodes on this path delay the execution runtime; as a result, this path dominates the execution latency as well as the hot path. All tasks in a program are assigned an execution time, an estimate of how long they will take to complete and the connections between the tasks (dependencies) are established. To identify the critical path in a CDFG, we need to find the longest execution latency among all the paths. This can be done by scheduling the paths in a CDFG explained in the preceding sections, and adding the execution time of each individual node together. To put it another way, by following a path among all the paths in a CDFG that has the longest duration of connected tasks and adding these durations together, the critical path is identified.

3.5 Different Combination of Hot Path and Critical Path

There are three types of combinations between hot paths and critical path in a CDFG which are as follows: 1) The critical path is the same as the hot path (i.e., those nodes in the critical path are exactly the same nodes which are part of the hot path). For example, there can be a critical path that has one loop with a large number of iterations. 2) The critical path is completely separate from the hot path. In this case, they do not have any. 3) The hot path is part of a critical path. In this case, not only the critical path contains all the nodes in the hot path but it has more nodes.

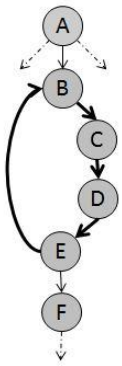


Fig.5. Hot path

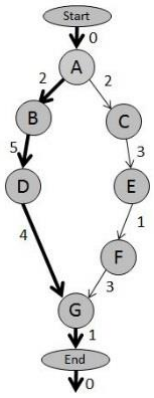


Fig.6. Path ABDG as the critical path

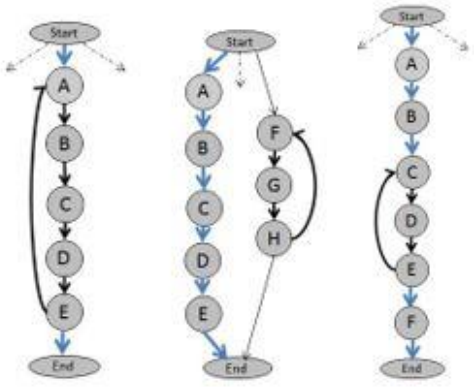


Fig.7. Differenct combinations of the hot path and the critical path

Figure 7 illustrates the three combinations in graph a, b, and c, respectively. For simplicity, each graph is part of a whole graph highlighted with the critical path and the hot path. The first two combinations occur very rarely in real applications. Though our approach is applicable on all the discussed combinations, we use only the third option since it is more common.

4. Target Co-Design Architecture

The architecture used in this paper is illustrated in Figure 8. It is composed of two CPUs as software components and two ASICs as hardware components. All these components have different execution time for each task in the graph. Each CPU communicates with both of the ASICs as shown in figure 8. Besides, all these software and hardware components communicate through a shared bus with a software-managed memory. Each task in the CDFG can have different implementation options on each available component but with different metrics. Given this architecture along with a technology library, a CDFG of an application and an area constraint, our method partitions the graph and assigns the tasks to either one of the CPUs or one of the ASICs. In this work the communication time between the CPUs and the ASICs are considered to be zero.

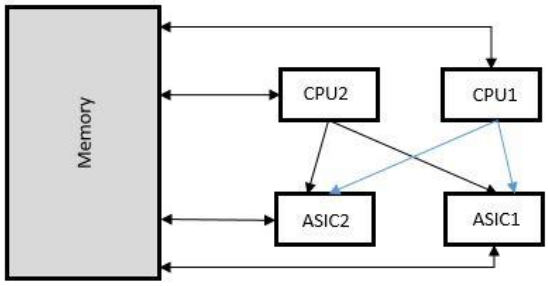


Fig.8. The target co-design architecture composed of two CPUs and two ASICs.

5. The Proposed Algorithm

The goal of the proposed algorithm is to maximize the performance of the given system while meeting the given area constraints for the ASICs. As shown in Algorithm 1, the inputs are a given CDFG, software execution time for the two given CPUs ($CPU1_{ET}$ and $CPU2_{ET}$) and hardware execution time for the two given ASICs ($ASIC1_{ET}$ and $ASIC2_{ET}$) for each node, area of each node on each ASICs ($ASIC1_{area}$ and $ASIC2_{area}$), and the total area available on the chip (A_T) which is the summation of the $ASIC1_{area}$ and $ASIC2_{area}$. The outputs are the partitioned CDFG in which each node is assigned to either hardware or software component, and the execution latency (L) of the partitioned graph. In the beginning of the algorithm, the software sets ($CPU1_{set}$ and $CPU2_{set}$) contain all the nodes in the CDFG, i.e., based on the given execution time of the CPUs for each node, the node executed faster in a CPU is dedicated to that CPU set. On the other hand, both the hardware sets ($ASIC1_{set}$ and $ASIC2_{set}$) are null since all the nodes are assigned to the software components in the beginning. According to the preceding sections, the critical path is found and assigned to Pc. $P_{UpdatedC}$ represents the updated critical path and it is initialized with the value in Pc. The next step is to find all the hot paths in the graph, and initialize L and the area used (AU) with longest execution time and 0, respectively. Line 9 calculates $(CPU_{ET}\text{-}ASIC_{ET})/ASIC_{area}$ for each node, which

Algorithm 1

1. Input: CDFG, $CPU1_{ET}$, $CPU2_{ET}$, $ASIC1_{ET}$, $ASIC2_{ET}$, $ASIC1_{area}$, $ASIC2_{area}$, A_T.
2. Output: Partitioned CDFG, L.
3. $CPU1_{set} = \{n_i\}$, $CPU2_{set} = \{n_j\}$ where Sigma i+j = total number of the nodes in the CDFG
   $ASIC1_{set} = \{\}$; $ASIC2_{set} = \{\}$;
4. P_c = critical path of the CDFG according to the CPU1 and CPU2 sets;
5. $P_{UpdatedC} = P_c$;
6. Find all the hot paths in the CDFG;
7. Initialize L;
8. $A_U = 0$;
9. Calculate the profit ratio for each node*;
10. Set P_{set} for each path;
11. **while** $A_U < A_T$ **do**
12. $P_c = P_{UpdatedC}$;
13. **while** ($P_c = P_{UpdatedC}$) && ((P_{set} in P_c) != {}) **do**
14. Select the first node (n_f) from the P_{set} in P_c with the highest profit ratio;
15. **if** ($A_U + ASIC_{area\ nf}$) < A_T **then***
16. $ASIC_{set} = ASIC_{set} + \{n_f\}$;
17. $CPU_{set} = CPU_{set} - \{n_f\}$;
18. $A_U = (A_U + A_{n_f})$;
19. Update L;
20. $P_{UpdatedC}$ = Updated critical path;
21. **end if**
22. Update P_{set};
23. **end while**
24. **end while**

* profit-ratio: each node has 4 profit ratio in this architecture, $(CPU1_{ET}\text{-}ASIC1_{ET})/ASIC1_{area}$, $(CPU1_{ET}\text{-}ASIC2_{ET})/ASIC2_{area}$, $(CPU2_{ET}\text{-}ASIC1_{ET})/ASIC1_{area}$, $(CPU2_{ET}\text{-}ASIC2_{ET})/ASIC2_{area}$.
* ASICset and CPUset in this step are chosen based on the initial partitioning phase.

is called as profit ratio in knapsack problem. This value is one of the priorities given to each node. Line 10 indicates a set for each path excluding the hot paths in the CDFG (P_{set}) that contains all the nodes in the decreasing order according to their profit value but the highest priority is given to the nodes that are in the hot paths. In other words, for each path, first the nodes in the hot paths are set in decreasing order then other nodes are added. If two nodes in a P_{set} have the same priority in terms of profit value and being in a hot path, the one that has less $ASIC_{area}$ has higher priority and is set earlier in P_{set}. In line 11 through 23, after assigning $P_{UpdatedC}$ to Pc, first node from the P_{set} of the critical path is selected and assigned to either of the $ASIC1_{set}$ or $ASIC2_{set}$ (based on the calculated profit ratio), if AU is less than AT. Each time one node is assigned to an $ASIC_{set}$, it is removed from its CPU_{set}. In the last step, the selected node will be removed from all the $P_{set(s)}$ regardless of its assignment to $ASIC_{set}$. If one selected node does not meet the area condition; the other nodes that meet the area condition will be chosen. This iterates as long as the total area used for all the nodes assigned to $ASIC_{sets}$ is less than A_T.

5.1 Illustrated Example

This section describes the operation of the proposed approach in an example. In the CDFG illustrated in figure 3, there are 26 basic blocks which represent the tasks in the program and several edges, which indicates the control dependency between the nodes. There are 4 back edges which represent the loops, and the number shown on each of them denotes the number of iteration for the specified loop. Based on Ball-Larus path profiling technique, there are totally 6 individual paths in this graph, considering the fact that paths can also start and end on loop back edges.

Based on the edge value assignment step in the Ball-Larus path profiling, Table 2 indicates the unique IDs for all the paths in the CDFG given in Figure 3. Table 1 presents random generated values for the inputs of the algorithm. The random values assigned to the execution times of CPUs ($CPU1_{ET}$ and $CPU2_{ET}$) and ASICs ($ASIC1_{ET}$ and $ASIC2_{ET}$) ranges from 10 to 40 and 1 to 5 respectively. The arbitrary values assigned to the ASICs area ($ASIC1_{Area}$ and $ASIC2_{Area}$) ranges from 5 to 35. The first step is to calculate the profit ratio value for each basic block. Let us assume that A_T is 100. According to the given inputs, we find the critical paths, all the hot paths and the execution latency. The followings are the initial results:

$ASIC1_{set}$ = {} and $ASIC2_{set}$ = {}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,R,S,T,V,W,X,Y}
$CPU2_{set}$= {B,C,H,N,P,Q,U,Z}
P_c = Path 3
Hot paths: {Path 10, Path 11, Path 12, Path 13}
L = 349 cycles

As explained in section 3.5, all the paths excluding hot paths, should be set in decreasing order giving the highest priority to the nodes in path 10, path 11, path 12, and path 13. Accordingly, all the P_{sets} are ordered as follows:

P_{set9} (ADJ) = {J,D,A} D,J (higher priority)
P_{set8} (AEKV) = {V,E,K,A}
P_{set7} (AELT) = {E,L,T,A}
P_{set6} (AELUW) = {W,U,E,L,A} W,U (higher priority)
P_{set5} (AEMX) = {M,X,E,A}
P_{set4} (AFN) = {N,F,A} N,F (higher priority)
P_{set1} (BGO) = {G,B,O}
P_{set3} (BHPRYZ) = {Y,R,P,Z,H,B} Y,R,P,Z(high prio)
P_{set2} (BHQ) = {H,B,Q}
P_{set0} (CIS) = {C,I,S}

In the first iteration, node Y is selected from P_{set3} since it is the initial value of P_c. Based on the profit ratio calculation, node Y has higher ratio when it is assigned to ASIC1. Since the area condition is met ($ASIC1_{area}$ =12 is less than A_T =100), node Y will be assigned to $ASIC1_{set}$ and removed from $CPU1_{set}$. Then all the hardware and software sets will be updated as follows:

$ASIC1_{set}$ = {Y} and $ASIC2_{set}$ = {}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,R,S,T,V,W,X}
$CPU2_{set}$= {B,C,H,N,P,Q,U,Z}

The updated critical path will remain the same as before, i.e., P_{set3}. L will decrease to 261 cycles and A_U will be 12. The selected node in addition to the collected results in the next iterations are as follows:

Second iteration: node R
$ASIC1_{set}$ = {Y,R} and $ASIC2_{set}$ = {}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,S,T,V,W,X}
$CPU2_{set}$= {B,C,H,N,P,Q,U,Z}
P_c: Path 3
L = 169 cycles and A_U = 25

Third iteration: node P
$ASIC1_{set}$ = {Y,R} and $ASIC2_{set}$ = {P}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,S,T,V,W,X}
$CPU2_{set}$= {B,C,H,N,Q,U,Z}
P_c: Path 3
L = 141 cycles and A_U = 44

Fourth iteration: node Z
$ASIC1_{set}$ = {Y,R,Z} and $ASIC2_{set}$ = {P}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,S,T,V,W,X}
$CPU2_{set}$= {B,C,H,N,Q,U}
P_c: Path 6
L = 126 cycles and A_U = 72

Fifth iteration: node W
$ASIC1_{set}$ = {Y,R,Z} and $ASIC2_{set}$ = {P,W}
$CPU1_{set}$ = {A,D,E,F,G,I,J,K,L,M,O,S,T,V,X}
$CPU2_{set}$= {B,C,H,N,Q,U}
P_c: Path 9
L = 104 cycles and A_U = 99

sixth iteration: node J

$ASIC1_{set}$ = {Y,R,Z} and $ASIC2_{set}$ = {P,W,J}

Table 1. Random values given to the inputs of the algorithm along with their calculated profit value.

B.B	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$CPU1_{ET}$	11	35	29	10	12	13	27	35	12	27	23	11	18	19	11	32	35	27	17	13	35	18	25	28	23	24
$CPU2_{ET}$	17	30	17	32	34	13	28	31	28	38	23	28	18	18	32	11	13	29	23	17	21	23	34	35	27	11
$ASIC1_{ET}$	5	1	5	5	4	1	1	2	3	3	5	1	1	4	3	2	1	4	5	3	5	4	4	4	1	1
$ASIC2_{ET}$	2	5	2	5	1	1	2	5	5	4	4	1	1	1	3	4	1	3	5	2	5	4	3	2	1	2
$ASIC1_{Area}$	34	35	27	30	26	25	24	30	9	27	35	24	20	27	30	34	26	13	18	24	24	7	22	18	12	28
$ASIC2_{Area}$	35	23	5	10	11	30	30	18	7	21	23	11	5	21	13	19	21	26	22	18	5	19	6	11	33	30

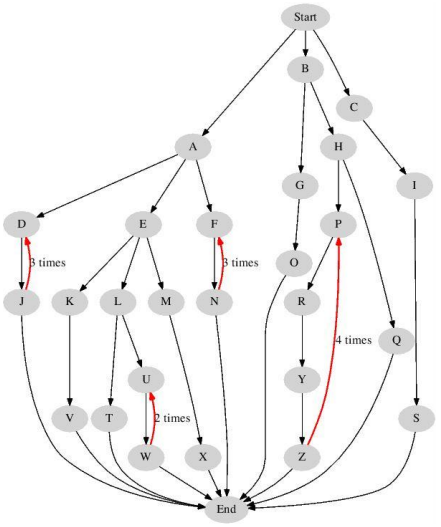


Fig.9. Given CDFG with individual tasks and 4 loops.

$CPU1_{set}$ = {A,D,E,F,G,I,K,L,M,O,S,T,V,X}
$CPU2_{set}$= {B,C,H,N,Q,U}
P_c: Path 4
L = 122 cycles and A_U = 104

The sixth iteration is actually the last iteration based on the given total area. There is only 1 unit of area left and the area of the remaining nodes is more than that value, therefore the algorithm terminates and the final result will be the results in the sixth iteration.

Evidently, as we increase the presumed area, the execution time decreases. If profit ratio is the same for a CPU, then the ASIC that has less ET is chosen. Like task 'I' in this graph.

6. Experimental Evaluation

This section presents the experimental results of the proposed approach. Table3 contains the first set of results. There are 10 randomly generated CDFGs in which the number of nodes varies from 9 to 44. Each CDFG is randomly generated using Task Graphs For Free (TGFF). This tool is designed to generate pseudo-random task-graphs for use in different area of research related to scheduling and binding. TGFF does not generate back edges are randomly added to the generated graph.

Path ID	Path Name
0	CIS
1	BGO
2	BHQ
3	BHPRYZ
4	AFN
7	AELT
8	AEKV
9	ADJ
10	PRYZ
11	FN
12	UW
13	DJ

Table 2. Calculated Path IDs based on Ball-Larus prfiling technique

The number of iteration for each back edge ranges from 2 to 5. For each graph, one random number is generated as an area constraint for the whole hardware components. Our approach is compared with two other approaches. The first one is 0-1 knapsack problem (KP), in which nodes are selected only according to their profit-weight ratio and it does not matter if the nodes are part of the hot path or critical path. Unlike the first approach, the second method is a path-based method called as Critical Path (CP) approach which is similar to our approach without considering the hot paths as a priority. In this approach, the nodes are selected only if they are in the critical path. In the second column of table 3, the execution time of each of the three approaches is calculated and in the third column, the improvement in execution time is calculated by comparing our approach with the other two. The improvement on average for the two comparisons is 45.4% and 15.3%, respectively, which shows the effectiveness of our approach. In graph 1 and 10, our approach does not show any improvement in comparison to the CP approach. In graph 1, those nodes selected in our approach, and the ones selected in CP approach are the same but with different order. The reason is that, the critical path does not change after each iteration, and according to the presumed total area, both approaches repeat until all the nodes in the critical path are assigned to

CDFG	Number of Nodes	Number of Paths	Area Constraint (Gates)	Execution Time (clock cycles)			Improvement (%)	
				KP	CP	Proposed approach	Our approach vs KP	Our approach vs CP

Table 3. The experiment evaluation comparing the proposed approach with two other approach. KP: Knapsack approach; CP: Critical Path.

1	10	3	50	88	52	52	40	0
2	14	4	83	297	121	86	71	28
3	19	6	61	210	133	120	42	10
4	13	5	80	113	87	53	53	39
5	16	6	71	107	65	48	60	26
6	9	4	60	62	62	56	9	9
7	16	5	78	299	123	106	64	13
8	10	4	55	77	64	53	31	17
9	26	10	100	143	118	104	27	11
10	44	20	71	221	101	101	57	0
Improvement on Average (%)							45.4	15.3

hardware components. Therefore, it does not show any improvement. In graph 10, the given values make the approach, considering the fact that the execution time of our approach is still lesser. The reason is that the selected

CDFG	Execution Time (CC) without area increase			Execution Time (CC) with 20% area increase			Execution Time (CC) with 40% area increase		
	KP	CP	HP+CP	KP	CP	HP+CP	KP	CP	HP+CP

Table 4. The second set of experimental evaluation assuming 20% and 40% area increase is allowed.

3	210	133	120	198	133	98	182	124	86
4	113	87	53	87	67	53	87	50	50
5	107	65	48	77	48	46	77	43	43
6	62	62	56	62	62	44	62	44	44
7	299	123	106	278	108	86	278	85	73
8	77	64	53	77	46	46	77	44	44
9	143	118	104	143	104	101	143	90	82
10	221	101	101	221	95	89	221	89	83
Improvement on average (%)				**Our approach vs KP**			**Our approach vs CP**	**Our approach vs KP**	**Our approach vs CP**
				46.3			14.0	48.2	7.1

selected nodes in both approaches to be the same but in different order, though the critical path changes after a few iterations. The second

set of results is shown in Table 4. This table presents the execution time of the generated graphs shown in table1 with 20% and 40% area increase, respectively. In 20% area increase, our approach has 46.3% execution time improvement on average in comparison to KP, and 14.0% in comparison to CP. In 40% area increase, there is 48.2% improvement in comparison to KP and 7.1% in comparison to CP. In graph 8, 20% and 40% area increase make our approach and the CP approach to reduce the execution time with the same amount. The reason is that all the nodes in the hot paths were already assigned to the hardware components before area increase; therefore, the node selected in both the approaches is on the critical path but not the hot path. That is the reason we do not see any improvement in this case.

The 20% area increase in graph 4 shows a big improvement in CP approach in comparison to our node in CP approach is on the hot path that was already chosen by our approach. In graph 1, 40% area increase does not affect the execution time of any of the approaches, since there is not enough unit of area left for assigning the nodes to the hardware components.

As mentioned before, KP approach is not path-based and does not choose the nodes according to their direct effects on the execution time of the whole system. This is the reason in all the cases, there is a big difference in execution time between KP and our approach. Besides, KP assigns the nodes with small area to hardware components first. As we increase the area, the area of the remaining nodes is high, that is the reason KP does not give good results most of the times.

7. Conclusions

In this research, we propose an approach to efficiently partition a given CDFG to hardware and software tasks. The target co-design architecture contains two CPUs as software units and two ASICs as hardware units. The goal of this approach is to improve the performance by efficiently partitioning those nodes that are part of the critical path and the hot path while an area constraint is given. The hot paths in a given CDFG is found using the path profiling techniques, and the critical path is identified by finding the longest paths in terms of the execution time. Since these two paths dominate the total execution time of a system, they have a significant effect on the overall performance of a system without degrading other factors such as power consumption and reliability. The main idea in this research is to give higher priority to the tasks in the critical path, which are part of the hot paths as well. Among those tasks, the highest priority is given to the ones that directly have considerable effect on the critical path, e.g. the nodes which are part of a loop. When these frequently executed tasks are assigned to the ASICs, due to higher speed in comparison to CPUs, the execution time of the whole system decreases significantly, this leads to boosting the performance. We compared our approach with two other existing methods and the collected results show the effectiveness of our approach.

References

[1] D. E. Stevenson and F. R. Knuth, "Optimal measurement points for program frequency counts," BIT Numerical Mathematics, pp. 313-322, 1973.

[2] K. Ebcioglu, R. D. Groves, K. Kim, G. M. Silberman and I. Ziv, "VLIW compilation techniques in a superscalar environment," in Programming language design and implementation, 1994.

[3] T. Ball and J. R. Larus, "Efficient path profiling," in Microarchitecture, Paris, 1996.

[4] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," Science, pp. 671-680, 1983.

[5] J. Madsen, J. Grode and P. V. Knudsen, "Hardware/Software Partitioning Using the Lycos System," Springer US, 1997, pp. 283-305.

[6] V. Srinivasan, S. Radhakrishnan and R. Vemuri, "Hardware software partitioning with integrated hardware design space exploration," in Design, Automation, and Test in Europe, Paris, 1998.

[7] L. Luo, H. He, Q. Dou and W. Xu, "Hardware/ Software Partitioning for Heterogeneous Multicore SoC Using Genetic Algorithm," in Intelligent System Design andEngineering Application (ISDEA), Sanya, Hainan, 2012.

[8] J. Wu, P. Wang, S. Lam and T. Srikanthan, "Efficient heuristic and tabu search for hardwar/software partitioning," The Journal of Supercomputing, pp. 118-134, 2013.

[9] U. Hao and J. Benlica, "an effective multilevel tabu search approach for balanced graph partitioning," Computers and Operations Research, pp. 1066-1075, 2011.

[10] J. Wu, T. Srikanthan and T. Lei, "Efficient heuristic algorithms for path-based hardware/software partitioning," Mathematical and Computer Modelling, pp. 974-984, 2010.

[11] Y. Zhang, L. Wu, G. Wei, H. Wu and Y. Guo, "Hardware/software partition using adaptive ant colony algorithm," in Control and Decision, Nanjing, 2009.

[12] T. He and Y. Guo, "Power Consumption Optimization and Delay Based on Ant Colony Algorithm in Networks-on-Chip," Engineering Review, pp. 219-225, 2013.

[13] J. Wu, T. Srikanthan and G. Chen, "Algorithmic Aspects of Hardware/Software Partitioning: 1D Search Algorithms," IEEE Transactions on Computers, pp. 532-544, 2009.

[14] S. Li, C. Wong, C. Yu and C. Hsu, "Hardware/Software co-design for particle swarm optimization algorithm," in Systems Man and Cybernetics (SMC), Istanbul, 2010.

[15] R. Lowney and P. G. Cohn, "Hot cold optimization of large Windows/NT applications," in Microarchitecture, Paris, 1996.

[16] D. Cifuentes and C. Ung, "Optimising hot paths in a dynamic binary translator," in Binary Translation, 2000.

[17] T. Yasue, T. Suganuma, H. Komatsu and T. Nakatani, "An efficient online path profiling framework for Java just-in-time compilers," in Parallel Architectures and Compilation Techniques, 2003.

[18] K. Vaswani, Thazhuthaveetil, J. Matthew and Y. N. Srikant, "A programmable hardware path profiler," in Code Generation and Optimization, 2005.

[19] H. Zhang and W. H., "Improved HW/SW partitioning algorithm on efficient use of hardware resource," in Computer and Automation Engineering (ICCAE), Singapore, 2010.

[20] G. N. Jin and K. M., "A new graph structure for hardware-software partitioning of heterogeneous systems," in Electrical and Computer Engineering, 2004.

[21] C. Lo, J. Luo and M. Shieh, "Hardware/Software Codesign of Resource Constrained Real-Time Systems," in Information Assurance and Security, 2009.

[22] W. Jigang, T. Polytech, T. Lei and T. Srikanthan, "Efficient Approximate Algorithm for Hardware/Software Partitioning," in Computer and Information Science, Shanghai, 2009.

[23] A. Ray, W. Jigang and S. Thambipillai, "Knapsack Model and Algorithm for HW/SW Partitioning Problem," in Computational Science - ICCS 2004, Springer Berlin Heidelberg, 2004, pp. 200-205.

[24] W. Thambipillai and S. Jigang, "A branch-and-bound algorithm for hardware/software partitioning," in Signal Processing and Information Technology, 2004.

[25] B. Kaminska, "Scheduling of a control and data flow graph," in Circuits and Systems, 1993.

[26] T. Zhang, Q. Yue, X. Zhao, and G. Liu, “An improved firework algorithm for hardware/software partitioning,” in Applied Intelligence. 49, 950–962, 2019.

[27] S. Hassine, M. Jemai, and B. Ouni, Bouraoui, “Power and Execution Time Optimization through Hardware Software Partitioning Algorithm for Core Based Embedded System,” in Journal of Optimization, 2017.

[28] S. Yin, C. Xu, Y. Qin, “A HW/SW partitioning algorithm

for embedded security systems," in Journal of Computational Information Systems. 11. 237-246, 2015.
[29] A. Iguider, K. Bousselam, O. El Issati, M. Chami, and A. En-Nouaary, "Embedded Systems Hardware Software Partitioning Approach Based on Game Theory", Edition 3, Innovations in Smart Cities Applications, Springer, 2020.
[30] E. Azari and H. Koc, "Improving performance through path-based hardware/software partitioning," 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC), Sierre, 2015, pp. 54-59.
[31] N. Hou, X. Yan, and F. He, "A survey on partitioning models, solution algorithms and algorithm parallelization for hardware/software co-design," In Design Automation for Embedded Systems. 23(3), 2019.

Elham Azari received her B.S. degree in Computer Engineering at Ferdowsi University of Mashhad, Iran in 2011; and her M.S. degree from Computer Engineering at University of Houston-Clear Lake in 2015. She is currently a Ph.D. candidate at Arizona State University. Her research interests include digital design, embedded systems and deep learning.

Hakduran Koc received his B.S. degree in Electronics Engineering from Ankara University in 1997. After working in the industry for two years, he joined Syracuse University, NY where he received his M.S. and Ph.D. degrees in Computer Engineering in 2001 and 2008, respectively. During his graduate study, he was at the Pennsylvania State University as visiting scholar. He is currently chair and an associate professor of Computer Engineering at University of Houston-Clear Lake. His research is in the areas of digital design, embedded systems, and computer architecture. He is