GO Game Inspired Algorithm for Hardware Software Partitioning in Multiprocessor Embedded Systems

Adil Iguider[1], Kaouthar Bousselam[1], Oussama Elissati[1], Mouhcine Chami[1], Abdeslam En-Nouaary[1]

[1] Institut National des Postes et Telecomunications, Lab. STRS,Rabat, Morocco

Correspondence: Adil Iguider, Institut National des Postes et Telecomunications, Lab. STRS,Rabat, Morocco.

Received: October 16, 2019 Accepted: November 4, 2019 Online Published: November 22, 2019

doi:10.5539/cis.v12n4p111 URL:https://doi.org/10.5539/cis.v12n4p111

Abstract

The codesign is a robust methodology, used in modern embedded systems with the objective of achieving the functional specifications and meeting the non-functional requirements. The most interesting step in the codesing is the process of Hardware/Software Partitioning. The aim is to decide which functionalities of the system should be implemented in hardware (HW) or in software (SW). In this article, a new heuristic algorithm is proposed to simultaneously optimize the hardware area (cost) and the execution time (performance) of a multiprocessor system. The proposed algorithm is inspired from game theory and especially from the GO game. The system is modeled using the DAG graph (Data Acyclic Graph), and two players (HW player and SW player) play in turn and choose a block (functionality) from the graph (system). The HW player has the goal of optimizing the global HW area while the SW player has the objective of minimizing the global execution time. After the game termination, and based on the 0-1 Knapsack algorithm, a step of refinement is used to meet the constraint on the total hardware area or on the overall execution time if a constraint is pre-defined. Experimental results show that the proposed algorithm gives better solutions compared to the Simulated Annealing algorithm and the Genetic Algorithm.

Keywords: Multi-Processors Embedded Systems, HW/SW Partitioning, GO Game, MiniMax Algorithm, Bellman-Ford Algorithm, Heuristic Algorithms

1. Introduction

The increasing complexity in the design of modern embedded systems necessitates the adoption of new methodologies. The objective is to allow concurrent development of hardware (HW) and software (SW), and to improve the interaction between the hardware and the software in order to meet the performance requirements and the functional specifications. The Co-design (compound design) is one of the best methodologies used to achieve those objectives. The Co-design is divided into four steps, Co-specification, Co-synthesis, Co-simulation and Co-verification. The system architecture is defined in the Co-synthesis step. The most important process in this step is the Hardware Software Partitioning (HSP) process.

The HSP aims to decide for each functionality of the system, whether it is more advantageous to be implemented in HW or in SW. A number of non-functional factors are involved in the HSP process, the most influencing factors are the performance (execution time) and the hardware area (cost). Several works were proposed to find the best possible balance between these two factors. The first studies specifically used exact algorithms such as the Branch and Bound method (Jigang and Thambipillai, 2004a), the Integer Linear Programming (ILP) algorithm (Niemann and Marwedel, 1996) and Dynamic Programming (Knudsen and Madsen, 1996). These algorithms give the exact solution and they are very useful for systems with a small number of blocks; however when the number of the blocks increases, they tend to be very slow. To overcome this issue, the recent studies focus on heuristic algorithms. Heuristic algorithms give an approximation to the exact solution in a short time, the most classical algorithms are the Genetic Algorithm (Feng et al., 2014), Simulated Annealing algorithm (Banerjee and Dutt, 2004b), Tabu Search algorithm (Lin et al., 2014), Hill Climbing Algorithm (Sim et al., 2008) and Greedy Algorithm (Bhuvaneswari and Jagadeeswari, 2015). The majority of previous works studied the optimization of one metric (cost or performance) while respecting a given constraint on the other metric.

In this article we propose a novel heuristic algorithm, the objective is to simultaneously optimize the cost and the performance of the system. The system is considered to be multiple processors, which guarantees a parallel execution and leads to a higher performance. Directed Acyclic Graph (DAG) is used to represent the system, and the execution time of the system is considered to be equal to the execution time of the critical path (CP) in the graph. The proposed algorithm is in a form of a game inspired from the GO game. Two players (HW player and SW player) are playing in turn and choosing a node (block) from the graph. The game terminates when all the blocks of the system are attributed to either to the HW partition or to the SW partition. The HW player has the objective of minimizing the global cost and the SW

player has the objective of minimizing the overall performance of the system. To choose the best move (block), each of the two players uses the MimiMax algorithm. At the end of the game, if a constraint is pre-defined either on the cost or on the performance, and additional optimization is used for meeting the constraint; this optimization is based on the 0-1 Knapsack algorithm, the objective is to choose the best blocks to move from one partition (HW or SW) to the other partition.

The rest of this article is organized as follows. Section II gives the related work. The problematic is presented in Section III. The proposed approach is described in Section IV. The results of experiments are shown in Section V. Finally, conclusions are summarized in Section VI.

2. Related Works

In recent decades, The Hardware Software Partitioning (HSP) problem has taken big interest in the design of embedded systems. Among the most important objectives is to achieve the best tradeoff between the system performance (execution time) and the cost (hardware area). Several studies have been made to solve the HSP problem while considering these two metrics, and there is principally two families of the proposed algorithms, the family of exact algorithms and the family of heuristic algorithms.

Exact algorithms include mainly Branch and Bound (*B&B*), Integer Linear Programming (*ILP*) and Dynamic Programming (*DP*). Branch and Bound is based on binary tree, the algorithms aims to find the optimal path from the top to the bottom of the tree, the optimal path has the minimum cost under some given constraints, examples of using *B&B* in the HSP problem are given in (Jigang and Thambipillai, 2004a; Strachacki, 2008; Jigang and Thambipillai, 2004b). Integer Linear Programming is composed of a linear function (objective function) with a set of variables and a set of linear inequalities, (Niemann and Marwedel, 1996, 1997) present examples of using the *ILP* to solve the HSP problem. In the dynamic programming method, a large problem is broken down into smaller problems, and by solving the smaller problems, the solution to the initial problem is established, example of using *DP* in HSP problem are described in (Knudsen and Madsen, 1996; Wu and Srikanthan, 2006). Exact algorithms are very efficient for small systems, but as the HSP is an NP-hard problem (Arató et al., 2005), they become difficult to use and time consuming for large scale HSP problems. Thus, heuristic algorithms are generally used to explore the search space in order to find close to optimal solutions in a reasonable amount of computation time.

Traditional heuristic methods include hardware-oriented (Gupta et al., 1992; Gupta and De Micheli, 1993) and software-oriented (López-Vallejo and López, 2003) approaches. The hardware-oriented (software-oriented) approach starts with a complete hardware (software) solution, and iteratively moves blocks of the system to the software (hardware) partition while respecting the given constraints. Modern heuristic methods include, inter alia, Genetic Algorithm (*GA*), Simulated Annealing (SA), Greedy Algorithm (*GR*), Hill Climbing Algorithm (*HC*), Tabu Search (TS) and Particle Swarm Optimization (PSO). *GA* algorithm is based on the survival of the fitness principle, the main steps are, the initialization of the first population, the parents' selection, performing the crossover to produce the offspring and the mutation of the offspring, the algorithm iterates from the selection step and evaluate each newly generated population, the algorithm terminates when the best individual that meet the termination condition is found; different approaches based on *GA* algorithm were proposed to solve the HSP problem, some of those approaches are given in (Feng et al., 2014; Zhao et al., 2013; Purnaprajna et al., 2007; Knerr et al., 2007; Li et al., 2014). SA algorithm consists of an analogy between the combinatorial optimization problem and the solid annealing process; the algorithm starts with an initial solution S and a parameter T (temperature), and at each iteration the algorithm generates some neighbors of the current solution, and probabilistically decides between keeping the current solution or replacing it by the best neighbor, and gradually decreases the temperature T; the algorithm iterates until a good enough solution is found for the system. Like the *GA* algorithm, the SA algorithm was used in many studies to deal with the HSP problem, some examples are cited in (Banerjee and Dutt, 2004b,a). *GR* algorithm starts by a candidate set, and iteratively adds the element that gives the best optimization, the algorithm stops when no improvement is obtained, (Bhuvaneswari and Jagadeeswari, 2015; Wu et al., 2010; Lin, 2013) are examples of approaches based on the *GR* algorithm for the HSP problem. *HC* algorithm starts with a sub-optimal solution, and repeatedly improves the solution until some conditions are met; unlike the *GR* algorithm, the *HC* algorithm has the possibility to avoid the local minima; an example of using the *HC* algorithm for the HSP problem is given in (Sim et al., 2008). TS algorithm begin with a starting current solution; then, at each iteration, it creates a candidate list of moves and choose the best admissible candidate, the algorithm iterates until the stopping criterion is satisfied; to avoid the local minima, TS uses a tabu table to discourage the search from coming back to a previously selected candidate; different studies used the TS algorithm to solve the HSP problem, some examples are presented in (Lin et al., 2014; Wu et al., 2013; Hou et al., 2016). PSO is an optimization technique inspired from the natural social behavior and dynamic movements with communications of swarms, PSO uses a number of particles (candidate solutions) that constitute a swarm moving around in the search space looking for the best solution, each particle keeps track of its best solution (pbest)

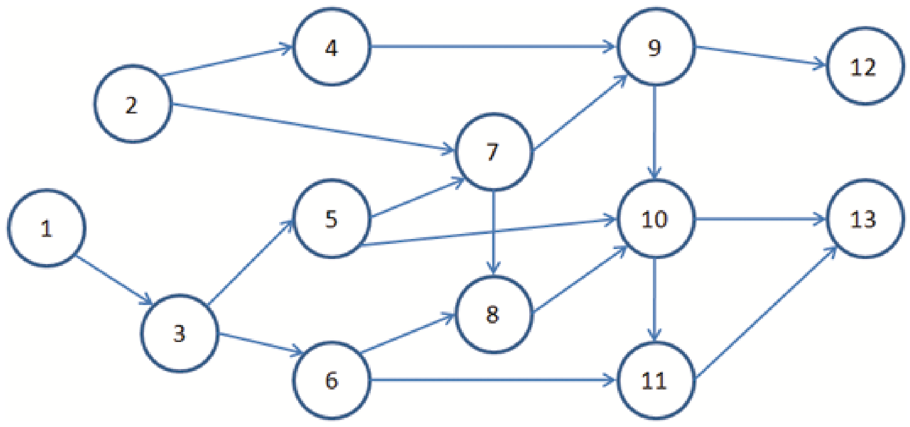

Figure 1. Example of DAG representation

and the best value obtained so far by any particle in its neighborhood (gbest), at each iteration, each particle adjusts its position according to its current position and current velocity and its distance to pbest and to gbest, the algorithm iterates until reaching the terminating condition; (Abdelhalim et al., 2006; Abdelhalim and Habib, 2011; Yan et al., 2018) present examples of using PSO in the HSP problem.

Other heuristic approaches were also proposed; In (Zhang et al., 2009), the proposed algorithms are based on Ant Colony Optimization. In (Zhang et al., 2008), the authors proposed an approach based on Artificial Immune System. In (Iguider et al., 2017), the authors proposed a method based on a shortest path in a Direct Acyclic Graph. another path-based HW/SW partitioning was proposed in (Wu et al., 2010). In (Mann et al., 2007), the possibilities of applying the Kernighan-Lin algorithm to the HSP problem were evaluated. In (Zhang et al., 2018), the authors proposed an algorithm for multi-processing embedded systems, the optimization objective is minimizing the critical path which demonstrates the longest path in a DAG graph, the critical path would determine the time required to execute the tasks on the embedded platform, the hardware area is set as a constraint, the proposed algorithm is based on Shuffled Frog Leaping Algorithm and Greedy Algorithm. In (Ouyang et al., 2016), the proposed algorithm aimed to minimize the overall execution time of the system while meeting the constraint on the hardware area in a heterogeneous multi-processing system, concerning the parallel execution of hardware and software, the finishing time of the system can be defined as the finishing time of the critical path in a tree based graph. Another example of minimizing the execution time of the critical path in a multiple processing embedded system in given in (Zhang et al., 2019), the proposed algorithm is based on Brainstorm Optimization Algorithm.

In this article, the proposed approach has the goal of optimizing the total hardware area and the overall execution time of a multi-processing embedded system, as in (Zhang et al., 2018; Ouyang et al., 2016; Zhang et al., 2019), the optimization of the execution time consists of minimizing the execution time of the critical path in a DAG graph representing the system.

3. Problem Definition

The system (ES) is composed of N block denoted B, such as $B = \{B_1, B_2, \dots B_n\}$. Where each block B_i can be implemented either in HW or in SW. The aim of the HSP is to divide the system into two sets H and S, where H (S) is the set of the blocks to be implemented in HW (SW), such that $H \cap S = \emptyset$ and $H \cup S = B$ while optimizating the defined objective function with the respect of the given constraints.

The system is represented in the form of a DAG as shown in the example in Fig. 1. Each node in the graph represents a block in the system and each vertex represents the communication of a block with its adjacent blocks. Each node can have multiple predecessors and multiple successors.

Each block (node) B_i has five parameters:

- c_{ih}: Hardware area cost of B_i if it is implemented in HW
- c_{is}: Hardware area cost of B_i if it is implemented in SW
- t_{ih}: Execution time of B_i if it is implemented in HW
- t_{is}: Execution time of B_i if it is implemented in SW
- x_i: The block's implementation (in HW $x_i = 1$, in SW $x_i = 0$).

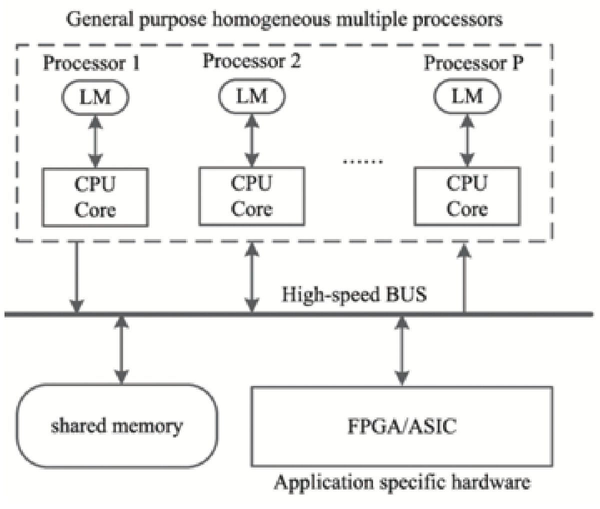

Figure 2. System architecture (Shi et al., 2019)

Each edge (source B_i, destination B_j) in the graph has two parameters:

- c_{ij}: The cost communication
- t_{ij}: The execution time communication

Tools such as Lycos as described in (Madsen et al., 1997) can be used to give an estimation of each of these parameters. It is noted that the cost communication and the execution time communication are considered only if the two attached nodes are implemented differently ($HW - SW$ or $SW - HW$). The values c_{ij} and t_{ij} are equal to zero for any (i, j) for which the edge with the source B_i and the destination B_j doesn't exist in the graph.

Let $X = (x_1, x_2, \dots, x_n)$ the vector solution to the HSP problem. The global hardware cost C as described in 1 is the sum of the cost of all the blocks (HW or SW) plus the communication cost.

$$C = \sum_{i=1}^{n}(x_i \times c_{ih} + (1 - x_i) \times c_{is}) + \sum_{i=1}^{n}\sum_{j=1}^{n}(x_i \times (1 - x_j) \times c_{ij} + (1 - x_i) \times x_j \times c_{ij}) \tag{1}$$

As the system is multi-processors, we consider the target architecture shown in Fig. 2 as described in (Shi et al., 2019).

Providing the parallel execution on multiple processors systems, the total execution time is defined to be the execution time of a critical path cp in the graph. The critical path is the path having the maximum execution time. The execution time of the path cp, is the sum of execution time of all nodes and communication time of all edges in cp. Thus, the total execution time T can be formulated as in 2.

$$T = \sum_{B_i \in cp}(x_i \times t_{ih} + (1 - x_i) \times t_{is}) + \sum_{B_i \in cp}\sum_{B_j \in cp}(x_i \times (1 - x_j) \times t_{ij} + (1 - x_i) \times x_j \times t_{ij}) \tag{2}$$

The HSP problem discussed in this article consists of minimizing C and T as formulated by $P1$ in 3.

$$P1 : \begin{cases} minimize\ C(X), \\ minimize\ T(X), \\ X \in \{0, 1\}^n \end{cases} \tag{3}$$

If a constraint is pre-defined on the hardware cost (C_{max}) or on the execution time (T_{max}), the HSP problem can be formulated respectively by $P2$ in 4 or by $P3$ in 5.

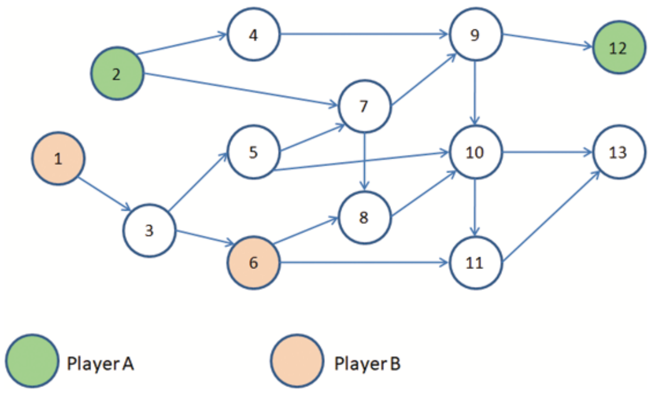

Figure 3. Game Initialization

$$P2: \begin{cases} minimize\ T(X), \\ s.t\ C(X) \leq C_{max}, \\ X \in \{0,1\}^n \end{cases} \tag{4}$$

$$P3: \begin{cases} minimize\ C(X), \\ s.t\ T(X) \leq T_{max}, \\ X \in \{0,1\}^n \end{cases} \tag{5}$$

4. Proposed Approach

The proposed approach for solving the problem $P1$ is in form of a game inspired from the GO game. The DAG graph (Fig. 1) represents the board of the game. The game is consists of two players,, a hardware player A and a software player B. The two players play in turn and choose a node from the graph (board). At the end of the game, the nodes (blocks) belonging to player A will be attributed to the hardware set H and the nodes belonging to player B will be attributed to the software set S. player A has the objective of minimizing the global cost given by 1 while player B has the objective of minimizing the system execution time given by 2.

4.1 Game Strategy

At the beginning of the game, the board is empty. The game is composed of three steps; in the first step the board is initialized; then in the second step, the two players play iteratively in turn according to the rules of the game; the third step represents the termination of the game.

4.1.1 Initialization

The initialization of the board consists of affecting two nodes to player A and two nodes to player B. The nodes attributed to player A are the nodes with the less possible value of HW cost, and the nodes attributed to player B are the nodes with the less possible value of SW execution time; the nodes attributed to A and the the nodes attributed to B must be as far as possible from each other. Fig. 3 gives an example of the graph (board) initialization.

4.1.2 Rules of the Game

The rules of the game are inspired from the GO game; the difference is that in this game, each player plays with the objective of minimizing its score; The score of player A is calculated using the equation 1 and the score of player B is calculated using the equation 2. The rules of the game are defined as follows:

- Rule 1: The list of possible moves for each player, is the list of empty nodes (not attributed) which are adjacent (predecessors and successors) to the nodes belonging to that player. For example, in Fig. 3, the list of possible moves for player A is $\{4, 7, 9\}$, and the list of possible moves for player B is $\{3, 8, 11\}$.
- Rule 2: A player can choose and play one move from the list of its possible moves; once a move is played, the node will be attributed to that player.
- Rule 3: As the objective of each player is to minimize its score, the rule called 'play and conquer' in the Go game is modified (inversed) in this game. When a player plays a move, if the corresponding node is adjacent to the

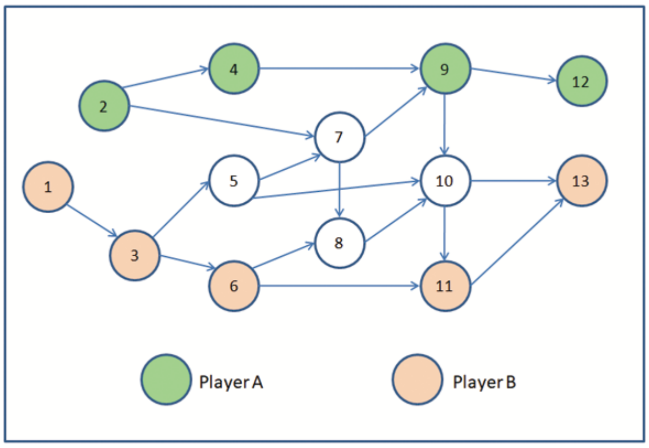

Figure 4. Example of play and conquer move: Before move

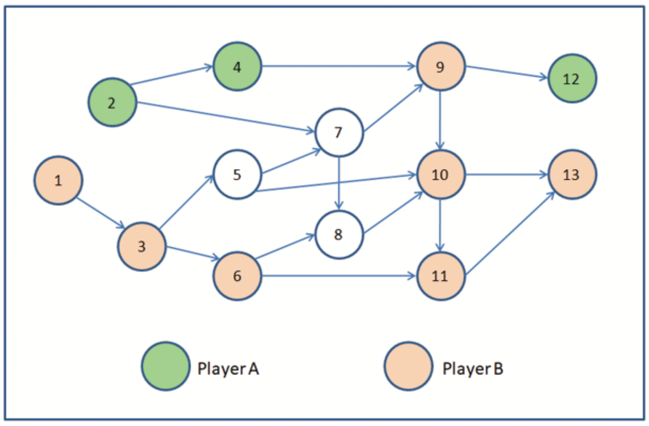

Figure 5. Example of play and conquer move: After move

opponent's nodes, the played node will be attributed to the opponent, and each node which is adjacent to the played node and belonging to player will be converted to the opponent. The example in Fig. 4 and Fig. 5 gives a more detailed explanation of this rule. Before playing (figure *a*), the list of possible moves for player *A* is { 7, 10}, the node 10 is adjacent to the nodes of player *B*, the list of the nodes belonging to *A* and adjacent to the node 10 is {9}, consequently, by applying this rule, if player *A* plays the move 10, the node 10 and the node 9 will belong to player *B* as shown in figure *b*.

4.1.3 Termination

The game terminates when each node in the graph is attributed to either to player *A* (HW) or to player *B* (SW). By the end of the game, the two partitions H and S are constructed.

4.2 Execution Time Computation

The execution time of the system is the execution time of the critical path in the graph. Two fictional nodes (E and S) are added to the graph (board) to represent respectively a unique entry point of the graph and a unique exit point of the graph. The node E is attached to the entry nodes of the system and the node S is attached to the exit nodes of the system. The constructed graph is shown in the example of Fig. 6.

The nodes of this graph don't have a value. Each edge in the graph (source B_i, destination B_j) has one value (weight), this value represents the execution time of the source block plus the execution time communication with the destination block, the value of each edge is calculated following the combinaison given in table 1.

The execution time of the critical path is the longest path in this graph from the entry point E to the exit point S. Finding the longest path in a graph is hard to solve using dynamic programming, because it lacks the optimal substructure property (an optimal solution that can be constructed from optimal solutions of sub-problems). Fortunately, a positive edge weight connected DAG does have the optimal substructure property. Therefore, to calculate the longest path, we use the Bellman-

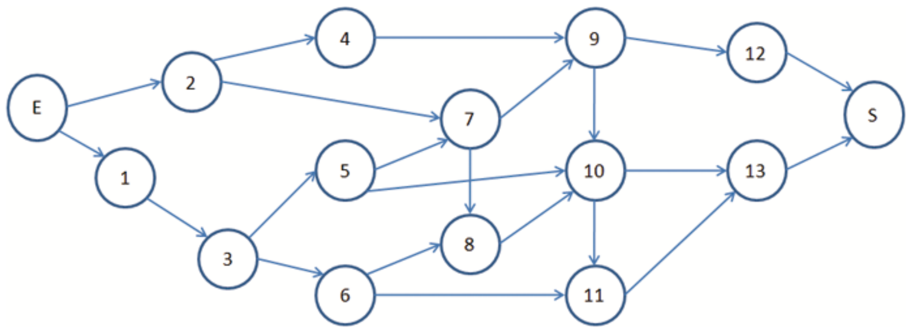

Figure 6. Graph with Entry and Exit nodes

Table 1. Edge's value calculation

B_i	B_j	**value**
E	any type(HW, SW, not attributed)	0
any type	S	0
HW	HW or not attributed	t_{ih}
HW	SW	$t_{ih} + t_{ij}$
SW	SW or not attributed	t_{is}
SW	HW	$t_{is} + t_{ij}$
not attributed	any type	0

Ford algorithm for finding the shortest path by reversing signs on weights in the graph.

4.3 MiniMax Algorithm

To choose the best possible move, each of the two players (A and B) uses the MiniMax algorithm. The MiniMax algorithm is explained in (Elnaggar et al., 2014) and in (Kang et al., 2019). The Minimax algorithm is applied in two player deterministic games. The two players are named Max and Min and each player takes turns and has perfect information of the possible moves of the adversary. From a given position in the game, the objective is to find the best move with the highest score for the Max player, The principle of the MiniMax algorithm is to build a search tree (game tree) with a given depth, the search tree represents all the possible evolutions of the game from the current position (the root) until the given depth. The score of each player is calculated using a utility function (evaluation function) at each state in the game. The same evaluation function is used for the two players for their scores. In the search tree, the nodes at the bottom level, are evaluated using the utility function, and then, the upper nodes are filled until the root node. At each level which is associated to the Max player, the moves that maximize its score are selected, similarly, at each level which is associated to the Min player, the moves that maximize the score of Min player are selected, in the same time, these moves minimize the score of the Max player. Fig. 7 gives an example of the search tree for the MiniMax algorithm.

Player A has the objective of minimizing the global hardware area of the system; when using the MiniMax algorithm, player A is considered to be the Max player that tries to maximize its score, the evaluation function is defined to be equal to the inverse of the hardware cost given in 1; player A (Max) plays virtually against a player Min that tries to maximize its own score and consequently minimize the score of A. Similarly, player B has the objective of minimizing the overall execution time of the system; when using the MiniMax algorithm, player B is considered to be the Max player that tries to maximize its score, the evaluation function is defined to be equal to the inverse of the execution time given in 2; player B (Max) plays virtually against a player Min that tries to maximize its own score and consequently minimize the score of B.

4.4 Refinement

At the end of the game, the two partitions H and S are constructed; consequently H and S represent a solution to the problem $P1$. This solution has an execution time T and a cost C. The step of refinement is used for the problem $P2$ or the problem $P3$, where a constraint is pre-defined either on the total cost of the hardware area or on the total execution time of the system. We consider the problem $P2$ where the hardware cost must respect a constraint C_{max}.

The refinement is based on the 0-1 Knapsack algorithm (KP). After the construction of H and S, the objective of KP is to select some blocks from the set H and move them to the set S in order to meet the constraint C_{max}.

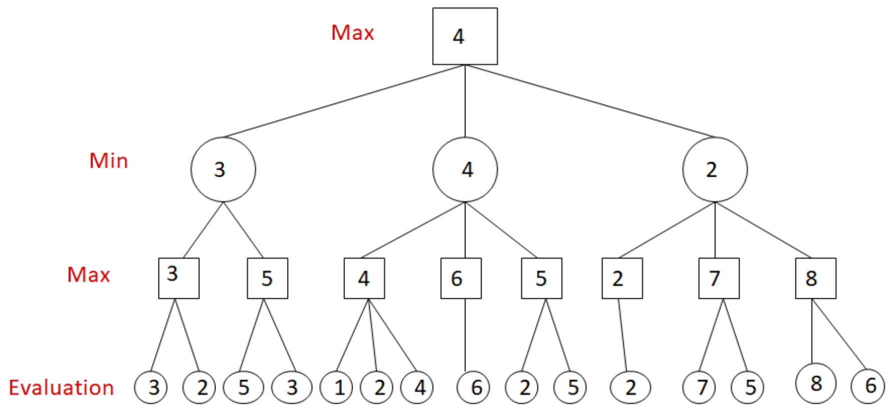

Figure 7. Search tree example

The principle of the 0-1 Knapsack problem is as follows, given a set A of m items, each item a_i has two parameters, a weight w_i and a value v_i, the Knapsack has a capacity W_{max} (maximum weight). The objectif is to solve the problem described in 6.

$$\begin{cases} maximize\ \sum_{i=1}^{m} v_i \\ s.t\ \sum_{i=1}^{m} w_i \leq W_{max} \end{cases} \tag{6}$$

A parameter r_i is calculated for each item i, $r_i = v_i/w_i$ represents the value-to-weight ratio for each item. The items are then sorted in the decreasing order according to r_i: $r_1 \geq r_2 \cdots \geq r_n$. The initial capacity is $C_{kp} = W_{max}$, at each iteration of the algorithm, the first item k with the weight $w_k \leq C_{kp}$ is selected and added to the Knapsack set S_{kp}, and then the capacity is updated: $C_{kp} = C_{kp} - w_k$, the algorithm stops when no item fits in the capacity C_{kp}.

If $C > C_{max}$, the weight and the value of each block of the hardware set H are calculated. For a block $B_i \in H$, we calculate the new hardware cost C_{new} and new execution time T_{new} as if B_i were implemented in SW. The weight and the value of B_i are as follows:

- $w_i = T_{new} - T$
- $v_i = C - C_{new}$

The blocks in the hardware set H are then sorted in the decreasing order according to the parameter $r_i = v_i/w_i$, the blocks are then moved to the software set S one by one until the meeting of the constraint C_{max}.

If $C < C_{max}$, the weight and the value of each block of the software set S are calculated. The objective is to move some blocks from the set S to the set H in order to get more optimal execution time while respecting the constraint C_{max}. For a block $B_i \in S$, we calculate the new hardware cost C_{new} and new execution time T_{new} as if B_i were implemented in HW. The weight and the value of B_i are as follows:

- $w_i = C_{new} - C$
- $v_i = T - T_{new}$

The blocks in the hardware set S are then sorted in the decreasing order according to the parameter $r_i = v_i/w_i$, the blocks are then moved to the hardware set H using the KP algorithm with the capacity $C_{max} - C$.

5. Experiments

In this part, we consider the problem P_2 which consists of minimizing the system's execution time under a constraint on the hardware cost. We verify the of the proposed approach results with the results of the Genetic Algorithm (GA) and the Simulated Annealing algorithm (SA). The three algorithms were implemented in Java and executed under Windows 10 on Dell (Intel Core (TM) i5-6300; 2.4 GHz; 8 GB of RAM). The parameters of each block B_i (c_{ih}, c_{is}, t_{ih} and t_{is}), the communication cost (c_{ij}) and communication the execution time (t_{ij}) were generated randomly with values between 0 and 10, with the respect of the following conditions:

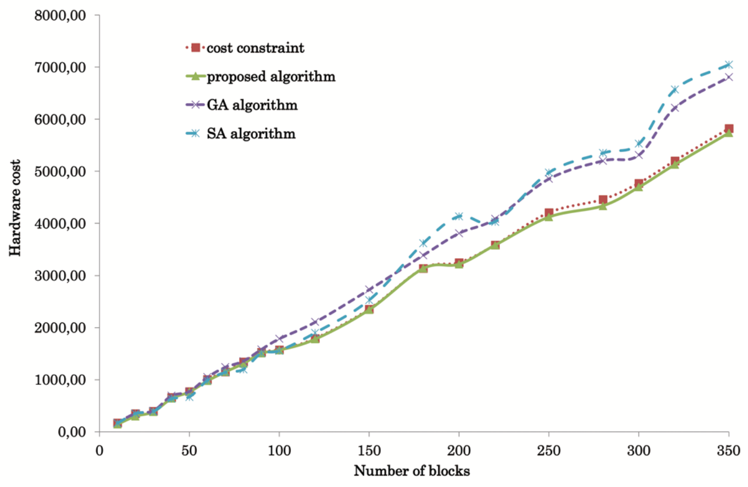

Figure 8. Comparison of the system's hardware cost between the three algorithms and the cost constraint

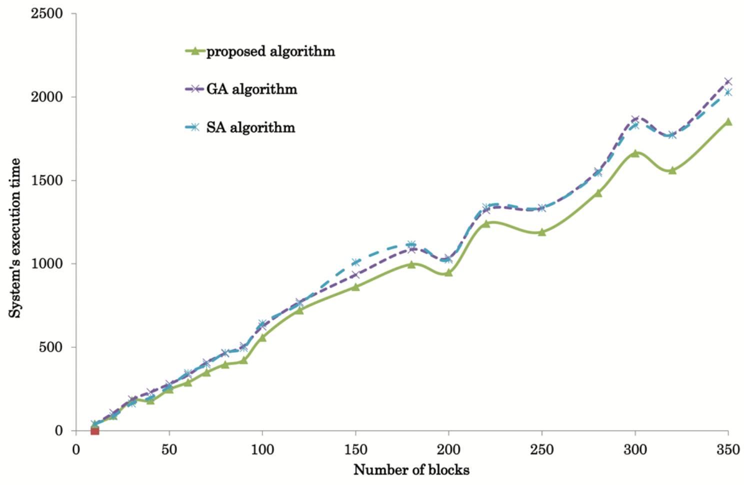

Figure 9. Comparison of the system's execution time between the three algorithms

- $c_{is} \leq c_{ih}$, $c_{ij} \leq c_{ih}$
- $t_{ih} \leq t_{is}$, $t_{ij} \leq t_{is}$

The constraint of the hardware cost C_{max} was also generated randomly between the minimum and the maximum hardware cost of the system. We made a serie of tests while varying the number of the blocks of the system from 5 to 350.

For the GA algorithm, each individual (chromosome) from a population is encoded using binary encoding scheme, if the i^{nd} gene is equal to 1 (0), then its corresponding block B_i is implemented in HW (SW). For the SA algorithm, each solution is also encoded using binary encoding scheme. As the objective is to minimize the execution time under the hardware cost constraint, we defined the fitness function for the GA algorithm and the cost function for the SA algorithm as follows:

$$\begin{cases} F = 1/T \\ F = F \times (C_{max}/C)^5 \;\; if\; C > C_{max} \end{cases} \tag{7}$$

The term ($F = F \times (C_{max}/C)^5$) is used to add penalty on the fitness (cost) function if the hardware cost constraint is violated. Fig. 8 gives the comparison in term of hardware cost between the three algorithms; the proposed algorithm and the SA algorithm give better cost compared to the GA algorithm for systems with small number ob blocks ($N \leq 100$),

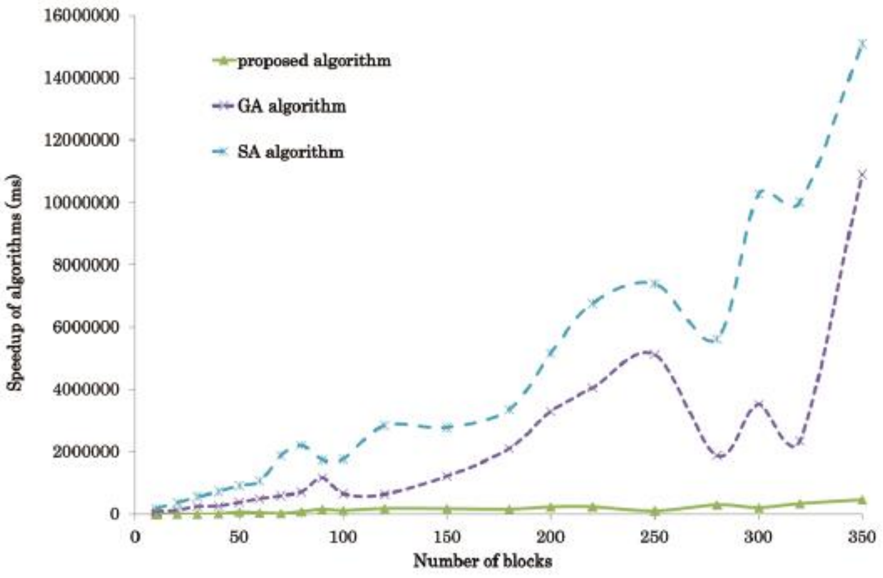

Figure 10. Comparison between the speedup of the three algorithms

when the number of blocks increases, the *S A* algorithm and the *GA* algorithm don't respect the cost constraint while the cost given by the proposed algorithm is always under the cost constraint.

Fig: 9 gives the comparison in term of system's execution time between the three algorithms; the results show that the proposed algorithm leads to more optimal execution time for almost all the tests and particulalry when the number of the blocks is big.

Fig: 10 gives the comparison between the speedup of the three algorithms in milliseconds, the figure show that the proposed approach is very fast compared to *GA* and *S A* algorithms.

6. Conclusions

In this article, we proposed a novel heuristic approach to deal with the Hardware Software Partitioning problem in Multiprocessors embedded systems. The proposed algorithm is based on game theory and especially on the GO game and using the Minimax algorithm. The objective is to simultaneously optimize the two most important parameters of the system, the global hardware cost and the overall execution time. The proposed algorithm was also enhanced to solve the *HS P* problem with the goal of optimizing one parameter under a given constraint on the other parameter. Experiment results showed that the system's performances (execution time and hardware cost) calculated while applying the proposed approach are more optimal compared to the Genetic Algorithm and to the Simulated Annealing algorithm. Experiment results showed also that the proposed algorithm is very fast compared to *GA* and *S A* algorithms. In the work, we will add the alpha-beta pruning to the Minimax algorithm which will augment the speedup of the algorithm, we will also extend the proposed algorithm to deal with an *HS P* problem with the goal of minimizing several parameters of the system.

References

Abdelhalim, M. B., & Habib, S. D. (2011). An integrated high-level hardware/software partitioning methodology. *Design Automation for Embedded Systems*, *15*(1), 19-50. https://doi.org/10.1007/s10617-010-9068-9

Abdelhalim, M., Salama, A., & Habib, S. D. (2006). Hardware software partitioning using particle swarm optimization technique. In *6th International Workshop on System on Chip for Real Time Applications*, pp. 189-194. IEEE. https://doi.org/10.1109/IWSOC.2006.348234

Arat ́o, P., Mann, Z. A., & Orb ́an, A. (2005). Algorithmic aspects of hardware/software partitioning. *ACM Trans. Des. Autom. Electron. Syst.*, *10*(1), 136-156. https://doi.org/10.1145/1044111.1044119

Banerjee, S., & Dutt, N. (2004a). Efficient search space exploration for hw-sw partitioning. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 122-127. ACM. https://doi.org/10.1145/1016720.1016752

Banerjee, S., & Dutt, N. (2004b). Very fast simulated annealing for hw-sw partitioning. *Technical Report, CECS-TR-04-17*.

Bhuvaneswari, M., & Jagadeeswari, M. (2015). Hardware/software partitioning for embedded systems. In *Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embedded Systems*, pp. 21-36. Springer. https://doi.org/10.1007/978-81-322-1958-3_2

Elnaggar, A. A., Gadallah, M., Aziem, M. A., & El-Deeb, H. (2014). A comparative study of game tree searching methods. *International Journal of Advanced Computer Science and Applications*, *5*(5), 68-77. https://doi.org/10.14569/IJACSA.2014.050510

Feng, J., Hu, J., & Qi, C. W. (2014). Hardware/software partitioning algorithm based on genetic algorithm. *Journal of Computers, 9*, 1309-1315. https://doi.org/10.4304/jcp.9.6.1309-1315

Gupta, R. K., & De Micheli, G. (1993). Hardware-software cosynthesis for digital systems. *IEEE Design* & *test of computers*, *10*(3), 29-41. https://doi.org/10.1109/54.232470

Gupta, R. K., Coelho, C. N., & De Micheli, G. (1992). Synthesis and simulation of digital systems containing interacting hardware and software components. In *Proceedings 29th ACM/IEEE Design Automation Conference*, pp. 225-230. IEEE. https://doi.org/10.1109/DAC.1992.227832

Hou, N., He, F., Chen, Y., & Zhou, Y. (2016). An adaptive neighborhood taboo search on gpu for hardware/software codesign. In *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 239-244. https://doi.org/10.1109/CSCWD.2016.7565995

Iguider, A., Chami, M., Elissati, O., & En-Nouaary, A. (2017). Embedded systems hw/sw partitioning based on lagrangian relaxation method. In *Proceedings of the Mediterranean Symposium on Smart City Applications*, pp. 149-160. Springer. https://doi.org/10.1007/978-3-319-74500-8_14

Jigang, W., & Thambipillai, S. (2004a). A branch-and-bound algorithm for hardware/software partitioning. In *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology*, pp. 526-529.

Jigang, W., & Thambipillai, S. (2004b). A branch-and-bound algorithm for hardware/software partitioning. In *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology*, pp. 526-529.

Kang, X., Wang, Y., & Hu, Y. (2019). Research on different heuristics for minimax algorithm insight from connect-4 game. *Journal of Intelligent Learning Systems and Applications, 11*, 15-31. https://doi.org/10.4236/jilsa.2019.112002

Knerr, B., Holzer, M., & Rupp, M. (2007). Novel genome coding of genetic algorithms for the system partitioning problem. In *International Symposium on Industrial Embedded Systems*, pp. 134-141. https://doi.org/10.1109/SIES.2007.4297327

Knudsen, P. V., & Madsen, J. (1996). Pace: a dynamic programming algorithm for hardware/software partitioning. In *Proceedings of the Fourth International Workshop on Hardware/Software Co-Design*, pp. 85-92. https://doi.org/10.1109/HCS.1996.492230

L ́opez-Vallejo, M., & L ́opez, J. C. (2003). On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems (TODAES), 8*(3), 269-297. https://doi.org/10.1145/785411.785412

Li, W., Li, L., Sun, J., Lv, Z., & Guan, F. (2014). Hardware/software partitioning of combination of clustering algorithm and genetic algorithm. *International Journal of Control and Automation, 7*, 347-356. https://doi.org/10.14257/ijca.2014.7.1.31

Lin, G. (2013). An iterative greedy algorithm for hardware/software partitioning. In *2013 Ninth International Conference on Natural Computation (ICNC)*, pp. 777-781. https://doi.org/10.1109/ICNC.2013.6818080

Lin, G., Zhu, W., & Ali, M. M. (2014). A tabu search-based memetic algorithm for hardware/software partitioning. *Mathematical Problems in Engineering, 2014*, 1309-1315. https://doi.org/10.1155/2014/103059

Madsen, J., Grode, J., Knudsen, P. V., Petersen, M. E., & Haxthausen, A. (1997). Lycos: The lyngby co-synthesis system. *Design Automation for Embedded Systems, 2*(2), 195-235. https://doi.org/10.1023/A:1008884219274

Mann, Z., Orb ́an, A., & Farkas, V. (2007). Evaluating the kernighan-lin heuristic for hardware/software partitioning. *International Journal of Applied Mathematics and Computer Science, 17*(2), 249-267. https://doi.org/10.2478/v10006-007-0022-3

Niemann, R., & Marwedel, P. (1996). Hardware/software partitioning using integer programming. In *Proceedings of the European Design and Test Conference*, pp. 473-479. https://doi.org/10.1109/EDTC.1996.494343

Niemann, R., & Marwedel, P. (1997). An algorithm for hardware/software partitioning using mixed integer linear programming. *Design Automation for Embedded Systems, 2*(2), 165-193. https://doi.org/10.1023/A:1008832202436

Ouyang, A., Peng, X., Liu, J., & Sallam, A. (2016). Hardware/software partitioning for heterogenous mpsoc considering communication overhead. *International Journal of Parallel Programming, 45*. https://doi.org/10.1007/s10766-016-0466-x

Purnaprajna, M., Reformat, M., & Pedrycz, W. (2007). Genetic algorithms for hardware–software partitioning and optimal resource allocation. *Journal of Systems Architecture, 53*(7), 339-354. https://doi.org/10.1016/j.sysarc.2006.10.012

Shi, W., Wu, J., Jiang, G., & Lam, S. K. (2019, 02). Multiple-choice hardware/software partitioning for tree task-graph on mpsoc. *The Computer Journal*. https://doi.org/10.1093/comjnl/bxy140

Sim, J. E., Mitra, T., & Wong, W. F. (2008). Defining neighborhood relations for fast spatial-temporal partitioning of applications on reconfigurable architectures. In *Proceedings of International Conference on ICECE Technology*, pp. 121-128. https://doi.org/10.1109/FPT.2008.4762374

Strachacki, M. (2008). Speedup of branch and bound method for hardware/software partitioning. In *1st International Conference on Information Technology*, pp. 1-4. https://doi.org/10.1109/INFTECH.2008.4621608

Wu, J., & Srikanthan, T. (2006). Low-complex dynamic programming algorithm for hardware/software partitioning. *Information processing letters, 98*(2), 41-46. https://doi.org/10.1016/j.ipl.2005.12.008

Wu, J., Srikanthan, T., & Chen, G. (2010). Algorithmic aspects of hardware/software partitioning: 1d search algorithms. *IEEE Transactions on Computers, 59*(4), 532-544. https://doi.org/10.1109/TC.2009.173

Wu, J., Srikanthan, T., & Lei, T. (2010). Efficient heuristic algorithms for path-based hardware/software partitioning. *Mathematical and Computer Modelling, 51*(7-8), 974-984. https://doi.org/10.1016/j.mcm.2009.08.029

Wu, J., Wang, P., Lam, S. K., & Srikanthan, T. (2013). Efficient heuristic and tabu search for hardware/software partitioning. *The Journal of Supercomputing, 66*(1), 118-134. https://doi.org/10.1007/s11227-013-0888-9

Yan, X., He, F., Hou, N., & Ai, H. (2018). An efficient particle swarm optimization for large-scale hardware/software co-design system. *International Journal of Cooperative Information Systems, 27*(01), 1741001. https://doi.org/10.1142/S0218843017410015

Zhang, T., Yang, C., & Zhao, X. (2019, 02). Using improved brainstorm optimization algorithm for hardware/software partitioning. *Applied Sciences, 9*, 866. https://doi.org/10.3390/app9050866

Zhang, T., Zhao, X., & Li, X. (2018). Efficient hardware/software partitioning based on a hybrid algorithm. *IEEE Access, 6*, 60736-60744. https://doi.org/10.1109/ACCESS.2018.2873636

Zhang, Y. D., Wu, L., Wei, G., Wu, H. Q., & Guo, Y. L. (2009). Hardware/software partition using adaptive ant colony algorithm. *Kongzhi yu Juece/Control and Decision, 24*, 1385-1389. https://doi.org/10.3724/SP.J.1187.2009.08032

Zhang, Y., Luo, W., Zhang, Z., Li, B., &Wang, X. (2008). A hardware/software partitioning algorithm based on artificial immune principles. *Applied Soft Computing, 8*(1), 383-391. https://doi.org/10.1016/j.asoc.2007.03.003

Zhao, X., Zhang, H., Jiang, Y., Song, S., Jiao, X., & Gu, M. (2013). An effective heuristic-based approach for partitioning. *Journal of Applied Mathematics, 2013*. https://doi.org/10.1155/2013/138037