A Survey on various method used in Hardware and Software Partitioning


School of Computing, SASTRA Deemed to be university, India

ABSTRACT

Efficient designing of any complex system necessitates decomposition of the same into a set of smaller subsystems. Subsequently, each subsystem can be designed independently and simultaneously to speed up the design process. The process of decomposition is called partitioning. Reconfigurable system with processor and FPGA are the promising innovation for the embedded system applications. The system demonstrates the low power consumption and high performance. The hardware and software (HW/SW) partition minimize the system energy consumption. This survey provides various algorithms and their merits and flaws which are described in the survey article.

Keywords: Field Programmable Gate Array (FPGA), Hardware and Software partitioning, ASIC, Genetic Algorithm, Heuristic Algorithm, Dynamic Programming Partitioning (DPP)

1. INTRODUCTION

Reconfigurable system comprises of processor and FPGA (Field Programmable Gate array) and it’s the most innovative method in embedded system applications. These kind of system meet low power consumption and performance requirements. An Embedded system is a heterogeneous architecture. It contain software processor and hardware like ASIC and FPGA as a coprocessor[1],[2] [7]. The software processor provides the flexibility and reduces the cost. The hardware coprocessor helps to increase the throughput of the system. The dynamic reconfigurable system provides flexibility and offers unlimited hardware resource over time.

Realization in hardware facilitates better performance and the hardware execution is much faster than the software execution. Hardware realization of large application is limited because of cost and size constraints. To overcome the drawbacks both software and custom hardware is realized together to meet the performance constraint. The design space exploration formed by the software and hardware components is referred as Hardware and Software co design. The co-design of such system can be defined as partitioning, co synthesis, co verification and co
simulation of hardware and software parts. The first step of the method hardware and software partitioning is discussed in this paper.

2. HARDWARE AND SOFTWARE PARTITIONING

2.1 General Partitioning

Partitioning [3] is a fundamental CAD optimization problem and it is used in every level of abstraction in synthesis of the design. Partitioning problem consists of connected modules and groups them to satisfy a set of constraints and design variables. During physical synthesis, partitioning is used first then the floor planning, placement and compaction are performed. Modules are gates connected by the wires and the highly connected gates are partitioned in the same level. This type of partition reduces the interconnect delay caused by the wires between the gates. Higher levels of abstraction modules become larger, they move from standard cells to macro cells (architecture level). Partitioning at higher level impacts the performance and the interconnect delay is more and at the lower level it is very hard to correct the modules. Hardware/Software partitioning is a high level of abstraction.

2.2 Representation of partitioning

A k-way partitioning problem is the basis for any variant of the partitioning problem. Given a set of modules \( M = \{m_1,m_2,…,m_n\} \) and find a set of clusters \( P = \{p_1,p_2,…,p_k\} \) such that \( p_i \subseteq M \) for \( i \rightarrow 1 \) to \( k \), \( \bigcup_{i=1}^k p_i = M \cap p_j = \emptyset \) for \( i \rightarrow 1 \) to \( k \) and \( i \neq j \) (i.e. the clusters are mutually disjoint).

The partitioning solution must satisfy a set of constraints and optimize an objective function. The objective function and constraints are subject to the problem and it optimizes it. When \( k=2 \), the problem is called bi-partitioning. The partitioning must be evaluated using some sort of estimation function. The estimation function \( E(P) = DP (dp_1, dp_2, …, dp_q) \) takes a partition \( P \) and returns a set of design parameters \( DP \). The design parameters are properties of the circuit. It includes the area, power, throughput, latency, etc. The estimation function is used to tell if the constraints are met and also evaluates the optimization function. The estimation function is extremely important, especially at the system level. The ideal estimation function would run the entire synthesis flow. Unfortunately, this is an extremely time consuming task. A large design can take several days to fully synthesize. Obviously, this is unfeasible, and evaluates a large number of partitions. The quality of partitioning directly depends on the precision of the evaluation function.
2.3 Formulation of hardware and software Partitioning

Hardware/software partitioning is the problem of dividing an application’s computations into a part that executes as a sequential instructions on a microprocessor (“the software”) and parts that runs as a parallel circuits on some IC fabric like an ASIC or FPGA (“the hardware”) such as to achieve design goals set for metrics like performance, power, size and cost. The circuit part commonly acts as a coprocessor for the microprocessor. Automated high level synthesis tool is available for hardware/software partitioning.

A partitioning of a system description impacts many aspects of the solution’s quality, that’s not limited to system execution time, total hardware area and cost, power consumption. System characteristics can be estimated at the high level design flow of partitioning. The estimated characteristics of a partitioning are used to evaluate the cost function, which rates the overall quality. The design space explored for the highest quality system partitioning (i.e. the hardware and software partition that minimizes a cost function for an arbitrary set of constraints).

The hardware/software partitioning must be performed early in the design cycle, as the functionality residing on a piece of hardware. The design characteristics like hardware size and performance of the partition must be estimated at the earliest stage. These provide the fast execution time of the partitioning heuristic within reasonable bounds. Therefore, the quality and performance of the partitioning solution depend largely on the function which considers the performance, accuracy and effectiveness of search heuristic that is utilized. These factors make hardware/software partitioning problem very complex and difficult to solve for a general system.

3. VARIOUS METHOD USED IN HARDWARE AND SOFTWARE PARTITIONING

3.1 Genetic Algorithm (GA) based Partitioning
The effectiveness of the GA [4] based approach exploiting the inherent nature of reconfigurable system to obtain optimal or near optimal performance speeds up that is relative to conventional software implementation. The ability to modify the system hardware for numerous application is possible through the use of reconfigurable hardware called FPGA( Run Time Reconfiguration (RTR)). An integrated partition and scheduler are used to efficiently partition hardware and software applications using RTR of reconfigurable resources.
3.1.1 GA problem Representation

The GA is a stochastic optimization algorithm loosely based on concepts of biological evolutionary theory. The application of GA to solve combinatorial problems may either require the development of new encoding/representation schemes and fitness evaluation techniques to reflect the problem.

3.1.2 Partitioning Representation

The problem can be represented as clusters of computations that can provide an optimal solution and attempting to optimize the applications speed up. The string composition is given by, Chromosome cell \( \rightarrow \{ p_i, c_{i_1} c_{i_2} \ldots c_{i_n}\} \)

Chromosome \( \rightarrow [\{ p_{1}, c_{1_1} c_{1_2} \ldots c_{1_n}\}, \{ p_{2}, c_{2_1} c_{2_2} \ldots c_{2_n}\}, \ldots, \{ p_{m}, c_{m_1} c_{m_2} \ldots c_{m_n}\}]. \)

Where \( P_i \) defines the partitioning status of the \( i^{th} \) candidate and \( m \) is the number of candidates that’s eligible to be partitioned to the hardware. The value 1 for \( P_i \) indicates the partition of the \( i^{th} \) candidate to hardware and 0 indicates it is partitioned to the software. The group of bits \( c_{i_1}, c_{i_2} \ldots c_{i_n} \) defines that the \( i^{th} \) candidate has commonality with another candidate, where \( n \) is the number of bits (resolution) in a conventional binary coding scheme that is required to represent the candidate. The candidate number of \( P_i \) is obtained from the decoding of the chromosome. The chromosome length for each solution is defined as \( N= m (n+1) \).

The representations at level two are the chromosome and the phenotype level. The chromosomes explain about the candidates that are partitioned to software and hardware and those candidates that are examined to the common functionality. In table 1, chromosome representing the candidates is represent as binary cells, accumulating the cells to make a binary string. Each cell contains information about a candidate. Every chromosome represents a possible solution to the hardware partition. The phenotype solution details the scheduled sequence of reconfiguration and execution of candidates partitioned to hardware. Phenotype details the behavior (scheduling) of the chromosome and the chromosome details the candidates to be partitioned and their functionality of common association with other candidates.
Table 1: Chromosome binary representation

<table>
<thead>
<tr>
<th>Candidate number</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>Not used</td>
<td>111</td>
</tr>
</tbody>
</table>

3.1.3 Fitness evaluation

Solutions are evaluated by determining the speed up of the hardware – software partition relative to the performance of the application is entirely partitioned to software. The fitness metrics as the inverse of the speedup and the GA converges to a minimum fitness value. The expression for evaluating the speed of hardware and software partitioned is given as,

Fitness =
\[
\frac{(SW_p - \sum_{i=1}^{q-1} SW_{e_i} + T_r + T_m + \sum_{i=1}^{q-1} HW_{e_i})}{SW_p}
\]  

\[\text{----(1)}\]

\(SW_e\) and \(HW_e\) denote the partitioned candidate’s execution time in software and hardware respectively, \(q\) specifies the number of candidates that are partitioned to hardware, where \(q_m\), \(T_r\) and \(T_m\) are the reconfiguration and memory latencies of the candidates in hardware respectively. \(SW_p\) is the application’s execution time when entirely partitioned to software. The fitness measure used in the problem clearly evaluates the desirable speedup from solution (hardware-software partitions) and reflects the effectiveness / quality of solutions.

The approach to solving the hardware-software partitioning is shown in figure 1. When the GA performs a pre-defined number of generations (\(N_{GEN}\)) it exits the partition’s evolution (stopping criteria). Otherwise, it performs selection and its associated genetic operations and
migrations to the current populations to generate new populations of partitioning solutions to an optimum solution (maximum speedup).

The main obstacle in hardware and software partitioning is to increase such performance is the latent reconfiguration that’s overhead inherent to such systems. The GA employs a functional commonality and dynamic reconfiguration weights in the fitness evaluation function with the resource constraint of fixed area available in the hardware. In exploiting the reconfigurable nature of the hardware the GA has to make a tradeoff between the area available and the performance it can obtain.

3.2 Algorithmic aspects for power-efficient hardware/software partitioning

Power efficiency is the one of the major considerations in the hardware/software co design. This paper [5] explains an optimization problem with an objective of minimizing power consumption under the constraints like hardware area A, execution time ε and heuristic algorithm with running time O(n log n). An exact heuristic algorithm is presented, to produce the optimal solution in O(n. A. ε).

3.2.1 Problem model and formulation

HW/SW partitioning problem is modeled based on the system specification like target architecture, specification language and design constraint. Commonly used target architectures are the single CPU Application Specific Integrated Circuit(ASIC) and the specification language is VHDL.

3.2.2 Representation of the Partitioning problem is modeled based on application

The given application corresponds to a sequence of n blocks, denoted as B1, B2,…..Bn that may be moved between hardware and software. Each Bt is followed by Bi+1 for i = 1,2,……n-1. Hardware and software blocks cannot be executed in parallel and the adjacent hardware blocks are assumed to be able to communicate between the read/write variables that they have in common without involving the software side.
Fig 1. HW/SW Partitioning Flow diagram
The computational model for HW/SW partitioning takes place with four blocks. The blocks may be functions or procedures, in which the communication between the blocks can be omitted because it is far less than the calculation time in coarse granularity. HW/SW partitioning modeled is still a NP-complete problem.

\( \alpha_i \) denotes the area penalty of moving Bi to hardware. \( P^s_i \) denotes the power required by B\textsubscript{i} in software implementation. \( P^h_i \) denotes the power required by B\textsubscript{i} in hardware implementation. \( e^s_i \) denotes the execution time of B\textsubscript{i} in software implementation. \( e^h_i \) denotes the execution time of B\textsubscript{i} in hardware implementation.

Fig 2 a) Actual data dependencies between HW/SW block. b) Interpreted Mode
In above figure $p_i$ represents the power saving of moving $B_i$ to hardware. i.e., $p_i = p_{i}^h - p_{i}^s$, $m \leq i \leq 4$. The objective is finding the partitioning, which yields the minimal power consumption while having a total area penalty less than or equal to the available hardware controller area $A$ and having an execution time less than or equal to the given constraint $\varepsilon$. Therefore $A$ is called area constraint and $\varepsilon$ is called time constraint.

### 3.3 Heuristic algorithm

HW/SW partitioning problem is presented as an extension of 0-1 knapsack problem. Given a knapsack capacity $C$ and the set of items $S = \{1,2,...,n\}$, where each item has a weight $w_i$ and a benefit $b_i$. The problem is to find a subset $S' \subset S$, that maximizes the total profit $\sum_{i \in S'} b_i$ under the constraint that $\sum_{i \in S'} w_i \leq C$, i.e., all the items fit in a knapsack of carrying capacity $C$. This problem is called the knapsack problem(KP). The 0-1knapsack is a special case of the general knapsack problem that’s defined above, where each item can either be selected or not selected fractionally.

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{n} p_i x_i, \\
\text{subject to} & \quad \sum_{i=1}^{n} a_i x_i \leq A \\
\sum_{i=1}^{n} e_i x_i & \geq \varepsilon', \text{ and } x_i \in \{0,1\} \quad \text{---(2)}
\end{align*}
\]

The 0-1 KP problem can be solved in a pseudo-polynomial time. The accurate solution can be obtained within 1. It is a simple but very efficient heuristic algorithm for filling the knapsack according to their profit to weight ratio.

\[
\frac{b_1}{w_1} \geq \frac{b_2}{w_2} \geq \cdots \geq \frac{b_n}{w_n} \quad \text{---(3)}
\]

Then in each step the item with the largest profit to weight ratio is packed into the knapsack if the item fits in the unused capacity of the knapsack, until the capacity is used upon no item fits for the residual capacity of the knapsack.
3.3.1 Algorithm HEU

**Input:**
- Hardware area penalty of block Bi, \(1 \leq i \leq n\);
- Power saving of block Bi in hardware, \(1 \leq i \leq n\);
- Time saving of block Bi in hardware, \(1 \leq i \leq n\);
- Available hardware area, \(A\);
- Limits of execution time, \(\varepsilon\).

**Output:** The approximate solution \((x_1, x_2, \ldots, x_n)\).

1. For \(i := 1\) to \(n\) do:
   - \(q_i := \) power rank of Bi;

2. For \(i := 1\) to \(n\) do:
   - \(t_i := \) time rank of Bi;

3. For \(\alpha := 0\) step 0.1 to 1 do:
   - For \(i := 1\) to \(n\) do:
     - \(r_i := \alpha . q_i + (1 - \alpha) . t_i\);
   - For \(i := 1\) to \(n\) do:
     - Move the block with the smallest \(r_i\) to hardware;
   - Until A is used up or no block fits for the residual area;
   - Update the current solution \((x_1, x_2, \ldots, x_n)\) according to its power saving.

3.4 Exact algorithm

It is also known as Dynamic Programming Partitioning (DPP) method and it is a stage-wise search method suitable for optimization problems whose solutions may be viewed as the result of a sequence of decisions. It relies on a principle of optimality. Dynamic programming takes advantage of the duplication and arranges to solve each sub-problem only once.

\(B\left(k, a, e\right)\) denotes the best power saving of blocks from B1, B2, ..., Bk to hardware size area a within the execution time e. The best power saving \(B\left(k, a, e\right)\) equals \(B\left(k-1, a, e - e_k\right)\) or the maximum between \(B\left(k-1, a, e - e_k\right)\) and \(B\left(k-1, a - e_k, e - e_k\right) + p_k\) because the block Bk is assigned either to software or to hardware, respectively. \(B\left(k, a, e\right) = B\left(k-1, a, e - e_k\right)\) corresponds to the case that hardware area is not enough for Bk i.e., \(a_k > a\). Initially, \(B\left(k, a, e\right)\) is set to \(-\infty\) for all k and a if e < 0.
3.4.1 Formulation of the DPP

\[
B(k, a, e) = \begin{cases} 
-\infty & \text{for } e < 0; \\
0 & \text{for } k = 0, a = 0 \text{ or } e = 0 \\
B(k - 1, a, e - e_k^S) & \text{if } a_k > a; \\
\max \left\{ B(k - 1, a - a_k, e - e_k^H + p_k^H) \right\} & \text{else};
\end{cases}
\]

Input:
- \(A_i\) – hardware area penalty of block \(B_i\), \(1 \leq i \leq n\);
- \(P_i\) – Power saving of block \(B_i\) in hardware, \(1 \leq i \leq n\);
- \(\varepsilon_i^S\) – execution time of block \(B_i\) in software, \(1\) in;
- \(\varepsilon_i^H\) – execution time of block \(B_i\) in hardware, \(1\leq i \leq n\);
- \((1, 2, \ldots, A)\) – list of trial hardware areas;
- \((1, 2, \ldots, \varepsilon)\) – list of trial execution times.

Output: The optimal solution stored in \(\text{partition}_\text{list}[1:n]\).

3.4.2 Algorithm DPP

\begin{align*}
1 \text{ Set } B(k, a, e) \text{ to } -\infty \text{ for } e < 0; /\ast \text{initializing } B(k, a, e) /\ast \\
\text{Set } B(k, a, e) \text{ to } 0 \text{ for } k = 0, a = 0 \text{ or } e = 0; \\
2 \text{ for } k := 1 \text{ to } n \text{ do } /\ast \text{ computing } B(k, a, e) \text{ and making trace}\ast/ \\
\quad \text{for } a := 1 \text{ to } A \text{ do} \\
\quad \quad \text{for } e := 1 \text{ to } \varepsilon \text{ do} \\
\quad \quad \quad \text{if } a_k > a \text{ then } B(k, a, e) := B(k - 1, a, e - e_k^S) \\
\quad \quad \quad \quad \text{trace}(k, a, e) := (\text{sw}, a, e - e_k^S); \\
\quad \quad \quad \text{else if } B(k - 1, a, e - e_k^S) > B(k - 1, a - a_k, e - e_k^H + p_k^H) \\
\quad \quad \quad \quad \text{then } B(k, a, e) := B(k - 1, a - a_k, e - e_k^H). 
\end{align*}
trace(k,a,e) := (sw,a,e - e^k_k) else 
{B(k,a,e) := B(k-1,a,e - e^k_k) + p_k; 
trace (k,a,e) := (hw,a,e - e^k_k)}; 
end of.
/*followed by backtracking along the trace for solution.*/
3 (Answer, Area,Timelimit) := trace(n, A,E);
4 for i:=n downto 1 do 
{Partition_list[i] := Answer;
(Answer,Area,Timelimit) := trace(k-1,Area,Timelimit);}
end

This approach mainly emphasize the power consumption in HW/SW partitioning while several technique mainly focus on minimizing the execution time of the system ignoring the power consumption. Among the two kind of partitioning algorithm, heuristic algorithm HEU is very fast and it’s in the order of O(n log n). It gets nearly optimal solution for small-sized problems and is efficient for the large sized problem rather than the exact algorithm which is based on dynamic programming methodology.

3.5 Efficient heuristic algorithm for partitioning

Efficient heuristic algorithm for HW and SW partitioning is done based on the selected hot path. Hot path means the body of the loop, that consists of executed component with high frequency in a given application [6].

3.5.1 Computing models and formulations

Hot path of given application consists of a sequence of n blocks, denoted as B = {B_1,B_2,....B_n} that may moved between hardware and software. Each B_i is followed by B_{i+1} for I = 12,......n-1. Hardware blocks and software blocks cannot be executed in parallel. The adjacent hardware blocks are assumed to be able to communicate the read/write variable they have in common directly between them without involving the software side. H denotes the Hardware and S denotes the software. The objective of the partitioning for B such that B = H U S and H ∩ S = ∅, these yields the best speedup while having a total area penalty no more than the available hardware area.
3.5.2. Proposed heuristic algorithm

First review the knapsack problem that is related to the partitioning problem $P$. Given a knapsack capacity $C$ and the set of items $S = \{1, 2, \ldots, n\}$ where each item has a weight $w_i$ and a profit $p_i$.

Input: Source data for the blocked $B_1, B_2, \ldots, B_n$:

- $a_i$: hardware area penalty of block $B_i$, $1 \leq i \leq n$;
- $s_i$: execution time of block $B_i$ in software, $1 \leq i \leq n$;
- $h_i$: execution time of block $B_i$ in hardware, $1 \leq i \leq n$;
- $c_{i,i+1}$: Communication time between blocks $B_i$ and $B_{i+1}$, $1 \leq i \leq n$;

Output: The heuristic solution $(x_1, x_2, \ldots, x_n)$.

Algorithm HEA
begin
1 for $i := 1$ to $n$ do $\alpha_i := \frac{p_i}{a_i}$; /* calculate the profit to area ratio for each block */
2 $H := \emptyset$; $S := \{B_1, B_2, \ldots, B_n\}$; /* $H(S)$ indicates the block set assigned to hardware (software) */
3 $k := 1$; residual area := $A := (x_1, x_2, \ldots, x_n) := (0, 0, \ldots, 0)$
3.1 repeat
3.1.1 $B_r := \text{the block with max } \frac{\sigma_r}{a_r}$; /* select the block with the maximum profit-to-area ratio in the set $s$ */
3.2 if $(\alpha_r \leq \text{residual area})$ and $(\sigma_r > 0)$
then* block $B_r$ fits in the residual area */
begin
$x_r := 1$; /* Assign block $B_r$ to hardware */
$H := H \cup \{B_r\}$, /* update $H$ */
Update $\sigma_r$ (if $r > 1$) and $\sigma_r+1$ (if $r < n$);
residual area := residual area - $a_r$;
end;
3.3 $k := k + 1$;
3.4 $S := S \setminus \{B_r\}$; /* update $s$ */
until (residual area $\leq 0$) or (k $> n$);
4 Output $(x_1, x_2, \ldots, x_n)$;
end.
3.6 Refining heuristic solution via Tabu Search

Tabu Search (TS) is one of the traditional heuristic based algorithms to search for the global optimal solution for NP-hard problems. Generally TS consists of five parameters: Local search procedure, neighborhood structure, tabu conditions, aspiration condition and stopping rule. In the searching process, TS keeps a list, which is called a tabu list of the search moves during each iteration, to restrict the local search procedure in reusing those moves. A recent-based memory stores the recent search areas in order to avoid the cycling search of the local zone. The tabu search of a search move can be released at a certain time according to the recent-based memory size. A frequency based memory is used to store frequency of the searching in each area. An aspiration criteria is utilized so that, if a tabu move generates a better solution than all feasible solutions obtained so far, its tabu status is neglected.

Input: \( X_{\text{heur}} \) – The heuristic solution generated by the algorithm HEA;
Output: \( X_{\text{best-so-far}} \) – the best solution found by Tabu search;

3.6.1 Algorithm TSA
/* Tabu search Algorithm for the problem p. \( M \) indicates the fixed number of iterations, \( q \) indicates the neighborhood size */
Begin
1 \( X_{\text{local}} \rightarrow X_{\text{heur}} \) and \( X_{\text{best-so-far}} \rightarrow X_{\text{heur}} \);
2 for \( i :=1 \) to \( M \) do
   begin
      2.1 Generate \( q \) neighbors of \( X_{\text{local}} \);
      2.2 Update the degrees and \( d_{\text{cost}} \)s of the \( q \) neighbors;
      2.3 if all \( q \) neighbors are tabu-active
         then \( X_{\text{local}} := \) the neighbor with the minimal tabu degree
         else
            \( X_{\text{local}} := \) the neighbor with the minimal \( d_{\text{cost}} \);
      2.4 if \( E(X_{\text{local}}) < E(X_{\text{best-so-far}}) \) then
Heuristic algorithm for path based HW/SW partitioning on an extended computing model in which communication penalties between neighboring components are considered. The proposed heuristic algorithm is able to produce nearly optimal solutions for the case of coarse granularity; it also can generate high quality approximate solutions for the fine granularity. Efficient algorithm is based on tabu search. The tabu search algorithm is able to refine the heuristic solutions to the nearly optimal ones in acceptable runtime, both for the coarse granularity and for the fine granularity.

3.7 Hybrid application of heuristic algorithm based hardware and software partitioning

Hybrid algorithm (Tabu Search(TS) + Simulated Annealing(SA)) is proposed for hardware and software partitioning problem. These algorithm introduces techniques from SA into the process of generating the tabu table in order to accelerate the TS and escape the pitfall of the local search[7].

Tabu Search and Simulated Annealing

Tabu Search(TS) is a heuristic local search and is an alternative to randomized methods and enhance the performance of the local search method. The TS starts with a random initial solution or a specialized initial solution generated by some algorithms. TS produce a randomly mutated solution which replaces the current solution from its neighborhood. The solution mutation in the neighborhood is defined by the function that with each solution associates a set of neighbor solutions. There is an important short term memory called Tabu table. The need of Tabu Table is maintaining a selective history composed of some solution in previous mutations. The architecture of TS consists of forbidden solutions in the Tabu Table from being considered again for further iterations.

Simulated (SA) is a stochastic computational technique evolved from statistical mechanics for figuring out the near global minimum solutions to global optimization problems. The characteristic of SA is similar to thermodynamics, particularly in the manner of metal coding and...
annealing. The sequence of moves (i.e., neighborhood configurations), the SA is assumed to change its arrangement from energy \( E_{\text{old}} \) to \( E_{\text{new}} \) with probability

\[
\text{Prob} = \exp(-\frac{(E_{\text{new}} - E_{\text{old}})}{kt})
\]

Where \( k \) is known as the Boltzmann constant.

\( E_{\text{new}} \) – New configuration energy state.

\( E_{\text{old}} \) – Old configuration energy state.

### 3.7.2 Refining Tabu Table by Simulated Annealing

To get the optimal solution search, some techniques such as the aspiration criterion and Tabu frequency were considered in Tabu Table. Customize the aspiration criterion for TS to the HW/SW partitioning problem. Global optima search is the characteristic of SA and it is reasonable to use SA generating a neighbor list in TS. To escape a TS from local optimal solution it randomly selects a task from the initial solution and change its implementation state (from HW to SW, or from SW to HW) to obtain a new feasible solution.

### 3.7.3 TSSA

*Input:* Initial solution denoted as \( \xi \).

*Output:* Partitioning result denoted as \( A' \)

1. Set medium parameter \( \tau \rightarrow \varepsilon \)
2. Set medium parameter \( \phi \) to record best solution in TSSA
3. While the TSSA does not satisfy the termination condition do
   4. \( A' \leftarrow \text{SA - NeighborList - Generator}(\tau) \)
      Select best solution \( \text{sol}[i], \text{sol}[i] \in A' \)
      Randomly select node \( i \) from solution \( \tau \)
      if \( \text{sol}[i] \) satisfies the limit area of FPGA and the performance of \( \text{sol}[i] \) is better than \( \tau \) then
         \[
         \sigma \leftarrow \text{sol}[i]
         \text{goto:stop}
         \]
      for solution \( \text{sol}[i] \) in \( A' \) do
         for task \( t_i \) in \( \text{sol}[i] \) do
            if \( t_i = \text{HW} \) then
               \[
               \text{frequency}[i] \leftarrow \text{frequency}[i] + 1
               \]
      Performance\( (\tau) \leftarrow \text{performance} + \text{frequency} \)
   Update Neighbor List \( A' \)
for task $t_i$ in sol[$i$] do
  go to: stop
if $\forall sol[i] \in A'$, $\exists sol[i] \in$ Tabulist then
  Select relatively better solution $sol[i], sol[i] \in A$
  $\tau \leftarrow sol[i]$[f]
  goto: stop
stop: if the performance of $\tau$ is better than $\varphi$ then
  $\varphi \leftarrow \tau$
Add $\tau$ in Tabu list
Update Tabu List
Update HW Frequency
return $\varphi$

A Hybrid algorithm combines TS and SA in order to avoid the local search for the optimization problem resulting in better acceleration and high performance. This criterion is able to keep the good balance of the CPU and hardware resources, especially for the case of larger task graphs.

3.8 Energy-aware run time task partitioning

This paper provides a series of algorithms for reducing energy consumption under timing and area constraints for providing an efficient hardware and software partitioning. The list of algorithm are Energy-Aware Task Management, Contention aware Task Placement, subtask repartition, online Area-lock allocator [8].

3.8.2 Algorithm: Energy-aware Task Management (EA-TM)

1 While the system is running do
2   if a new task $\tau_i$ has arrived then
3     for each subtask in $\tau_i$ do
4       $E_f^{SW}_{i,j} = C^{SW}_{i,j} \times P_{i,j}^{SW}$
5       $E_f^{HW}_{i,j} = \frac{TP_{R_{j,p}}^{PR}_{i,j} + C^{HW}_{i,j} \times P_{i,j}^{HW}}{D_i}$
6     if $E_f^{SW}_{i,j} < E_f^{HW}_{i,j}$ then
7        $T_{yp_{i,j}} = SW$
else

    $Type_{i,j} = HW$

if $Type_{i,j-1} = Type_{i,j}$ then

    $\tau_{i,j}$ assign to the $c_{\infty}$

else

    $x=x+1$

    $\tau_{i,j}$ assign to the $c_{\infty}$

Set the type of $c_{\infty}$ as the $Type_{i,j}$

while contention-aware Task placement() is Failed do

    for each subtask in $T_i$ do

        if $Type_{i,j-1} = Type_{i,j}$ then

            $\tau_{i,j}$ assign to the $c_{\infty}$

        else

            $x=x+1$

            $\tau_{i,j}$ assign to the $c_{\infty}$

Set the type of $c_{\infty}$ as the $Type_{i,j}$

Invoke Schedulability Test;

Assign local deadline of each subtask by ALAP

Insert all software subtasks into the processor queue

Insert all hardware subtasks into the ICAP queue

Set all subtasks without predecessor as ready

if there is any subtask in the processor queue then

    Execute the ready subtask with the shortest deadline in the processor queue

if there is any subtask in the ICAP queue then

    choose the ready one with the shortest deadline in the queue
Online Area – lock Allocator();

if a subtask $\tau_{i,j}$ is finished then

if there is an immediate succeeding subtask $\tau_{i,k}$ of subtask $\tau_{i,j}$ then

if subtasks $\tau_{i,j}$ and $\tau_{i,k}$ are different types or they are in different rows then

Insert communication $a_{j,k}^i$ into the communication queue

else

if all predecessors of subtasks $\tau_{i,k}$ are ready then

Set the subtask $\tau_{i,k}$ as ready

end

The set of tasks $T = \{\tau_1, \tau_2, ... , \tau_m\}$ and each task $\tau_i$ and has a deadline $D_i$, $C_{i,j}^{SW}$ is software execution time when the subtask $\tau_{i,j}$ is executed on the processor, $C_{i,j}^{HW}$ is the hardware execution time when the subtask is executed on the FPGA, $TPR_{i,j}$ is the required reconfiguration time when the subtask $\tau_{i,j}$ is executed on a FPGA. $P_{i,j}^{SW}$, $P_{i,j}^{HW}$ and $P_{i,j}^{PR}$ are the execution power on the processor and FPGA, reconfiguration power respectively.[9][10]

To fully utilize the partially reconfigurable system, the partition task is based on energy utilization. The capabilities of EA-TM algorithms with varying chip sizes and applications are evaluated.

4. Conclusion

The survey provides some list of algorithm for hardware and software partitioning with the analysis of complexity, time consumption, and power and area requirement. It helps to explore the efficiency of the system with minimum power and area consumption. The algorithms are addresses here Genetic and Heuristics approaches. Heuristic approaches are trail and error method for finding the optimization towards the research work. Meta heuristics is another big area for computational intelligence that gives extend solution to the NP hard problems.
References


