Ant Colony Optimization using High Level Synthesis

Muhammad Amirul Hakimi Bin Zaprunnizam
Hochschule Hamm Lippstadt
Lippstadt, Germany
muhammad-amirul-hakimi.bin-zaprunnizam@stud.hshl.de

***Abstract*—Ant Colony Optimization (ACO) has recently gained a lot of traction, and it is usually a highlight of major algorithms. The ACO is a probabilistic method that is used to find optimization routes. It was based on the behaviors that ants engage in when searching for food. Biological ants employ pheromone-based communication as their primary mode of communication. In this paper we will see how the ACO algorithm is a probabilistic approach for addressing computational problems that may be simplified to finding optimal pathways across graphs in computer science and operations research. Multi-agent approaches inspired by the behavior of actual ants are referred to as artificial ants. Combinations of artificial ants and local search algorithms have emerged as the preferred solution for a variety of optimization tasks. ACO is a type of simulated evolutionary algorithm that, according to prior study, provides several advantages.**

I. Introduction

Nowadays, engineers are always having problem doing task in designing an operating system. The problems that may be occur during the process is to meet the goal with several limitations on the design and operation. This problem can always be solved by optimization. To achieve the best optimization of the system, there are some factors that needed to take into account for example set of action or elements. In other words, it is a process to find the best solution for one or more objectives function with multiple constraints. By selecting variables that solve the problem, the aim of optimization could be maximized and minimized [1]. In this paper I will introduce one of example of meta-heuristic algorithm which called Ant Colony Optimization that can help in reaching optimization.

The Ant Colony Optimization algorithm was proposed by an Italian scientist names Marco Dorigo as part of his PhD in 1992. In fact, the first ant inspired algorithm was called "Ant System". These days we use improved version of this algorithm which is now called ACO. The main inspiration of the Ant Colony Optimization comes from stigmergy in ant colony. Stigmergy refers to the interaction and coordination of organism in nature by modifying the environment. In stigmergy the trace of an action done by an organism simulates subsequent actions by the same other organism. This idea can prove in nature behavioral for example when a termite passes a ball of mud to another termite to fix the hole without planning and having a direct communication. These also can be seen between humans.

In general, this paper will include the model of Ant Colony Optimization algorithm, High Level Synthesis (HLS) and problem solving using ACO algorithm.

II. Model

In this section, I will show the model of Ant Colony Optimization. This section will help to visualize the model of the algorithm.

There are several iterations in the ACO algorithm. In each cycle, a group of ants uses heuristic knowledge and prior ant populations' experiences to develop comprehensive solutions. The pheromone trail, which is deposited on the constituent parts of a solution, is used to symbolize these gathered experiences. Depending on the problem to be solved, the pheromone can be placed on the components and/or connections in a solution. The following is a description of the pheromone update rule technique. In the ACO algorithm, an ant is a basic computational agent. It builds a solution to the problem one step at a time.

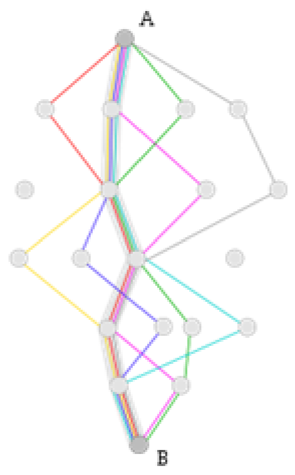

Fig. 1. Map for Ant Colony Optimization

- Circles indicates ants
- Lines indicates paths have been taken by ants to find food

In Figure 1 above shows how ACO works, a technique to calculate optimize path between point A nest and point B in space. In general point A is the nest and point B is the food. These are the steps how the model produced.

1) The first ant going to find food from point A to point B and comes back with leaving pheromone on the trail.
2) Second ant sense that pheromone knowing that it is the shortest path to find that food before the chemical evaporate. When the density of pheromone decreases it means that it is the longer path to find the food.
3) Most likely the next ants follow the line that has most intense pheromone sense.

The decision of choosing which paths or lines that are closest is depending on how dense the pheromone is. A permutation of π supplied objects can be used to explain any problem. The amount of pheromone released by ants in each round is calculated using a $n \times n$ pheromone matrix $[\tau_{ij}]$. Any ACO algorithm begins by initializing the pheromone matrix, assigning an initial value to each pheromone entry ($\tau_{init} > 0$). At each cycle, ants produces solutions $\pi_0 ... \pi_{m-1}$. Ants make several local judgments in order to construct solutions for succeeding item selections. The probability distribution over the choices of unchosen items in a certain selection set S is used to make decisions at random. S is determined in the following way [3].

$$\forall_j \in S : \rho_{ij} = \frac{\tau_i^{\alpha}{}_j \eta_i^{\beta}{}_j}{{}_{z \in S} \tau_i^{\alpha}{}_z \eta_i^{\beta}{}_z}$$

The relative weight of pheromone values and heuristic values is determined by the parameters α and β. The ACO algorithm performs several iterations until a given stopping condition is satisfied, such as a predetermined maximum number of iterations, a specific degree of solution quality, or the best solution has not changed over a specified number of iterations.

The ACO begins by creating a set of random solutions comprised of decision variables chosen from a collection of discrete values. All the solutions' fitness values are compared. The choice space is then assigned pheromone concentrations proportional to the fitness values of the solutions. The pheromone concentration indicates attractiveness. The portions of the decision space that produce better solutions have a higher pheromone concentration. The pheromone of a particular value of a decision variable is equal to the sum of all pheromones left by all ants with that value. In the next algorithmic stage, new ants (solutions) are created using knowledge gathered from previous ants. New solutions are created at random using a stochastic function that assigns a probability to each decision variable's allowed values based on its pheromone. Values with a higher pheromone content are more likely to be chosen. If the termination requirements are not met, pheromone concentrations are added to the decision space to produce new solutions after evaluating the fitness values of newly generated solutions. Otherwise, the algorithm will terminate. The ACO flowchart is shown in Figure 2 [2].

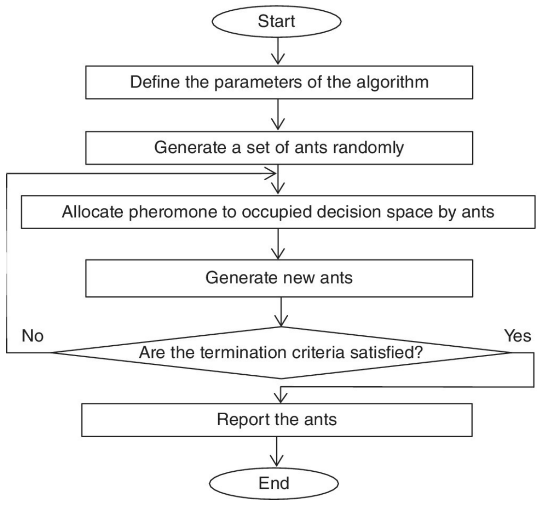
Start
Define the parameters of the algorithm
Generate a set of ants randomly
Allocate pheromone to occupied decision space by ants
Generate new ants
No
Yes
Are the termination criteria satisfied?
Report the ants
End

Fig. 2. Ant Colony Optimization Flowchart [2]

To end this section, reader should be able to understand generally how Ant Colony Optimization algorithm works. The following section will be discussed about High Level Synthesis and relatively solving problem using ACO.

III. High Level Synthesis

In this section, we will exposed to High Level Synthesis. The purpose of HLS is to help hardware designers construct and verify hardware more effectively by giving them more control over design architectural optimization and by allowing the designer to define the design at a higher level of abstraction while the tool implements the RTL.

Specification languages and design processes have evolved similarly in the hardware realm. Simulation tools have been widely used thanks to hardware description languages (HDLs) as Verilog (1986) and VHDL (1987). These HDLs have also been used as inputs to logic synthesis tools, allowing their synthesizable subsets to be defined. An HLS tool performs the following task based on the high-level description of an application, an RTL component library, and certain design restrictions [3].

1) Compiles the requirements.
2) Assigns hardware resources (functional units, processors, components for storage, buses, etc.).
3) Schedules activities according to clock cycles.
4) Associates activities with functional units.
5) Associates variables with data structures.
6) Ties bus transfers together.
7) Builds the RTL architecture.

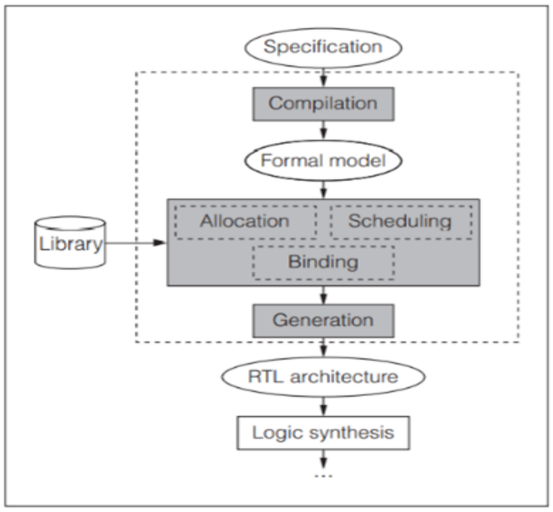

Fig. 3. High Level Synthesis design steps [3]

Sequential synthesis is based on the FSM model, whereas logic synthesis is based on the formalism of Boolean algebra. We enhance the FSM model by adding variable assignments for high-level synthesis. A collection of states, a set of transitions between states, and a set of actions connected with these states or transitions make up the FSM model. The behavioral description is compiled into an internal representation via the high-level synthesis mechanism. This model is used in all synthesis activities. In Figure 3 shows there are 3 main elements in HLS which is allocation, scheduling, and binding.

- **Allocation** - The kind and amount of hardware resources (for example, functional units, storage, or connection components) required to meet the design requirements are defined by allocation. Some components may be introduced while scheduling and binding jobs, depending on the HLS tool. Connectivity components, such as buses or point-to-point connections between components, might be introduced before or after the binding and scheduling activities, for example. The RTL component library is used to pick the components. For each action in the specification model, it's critical to choose at least one component. Other synthesis activities will need to leverage the library's metrics and component attributes (such as area, delay, and power).

- **Scheduling** - All the operations in the specification model must be planned in cycles. To put it another way, for each operation like $a = b\ op\ c$, variables b and c must be read from their sources (either storage components or functional-unit components) and brought to the input of a functional unit capable of executing operation op, and the result a must be brought to its destinations (storage or functional units). The operation can be scheduled in a single clock cycle or over numerous clock cycles, depending on the functional component to which it is mapped. It's possible to link operations (the output of an operation directly feeds an input of another operation).

- **Binding** - A storage unit must be assigned to each variable that carries values across cycles. Multiple variables with non overlapping or mutually exclusive lives can also be bound to the same storage units. Every operation in the specification model must be assigned to one of the functional units that may perform it. If numerous units have this capability, the binding algorithm must choose the best one. Connectivity binding is also required for storage and functional unit binding, as each transfer from component to component must be bound to a connection unit such as a bus or multiplexer.

As I mentioned as paragraph above, the ACO algorithm can solve any problem requires to find shortest path with minimum time. Here in this paper, I will explain how ACO works in High Level Synthesis by using FPGA Routing with minimum CPU time. The use of Boolean-based routing to solve routing problems in FPGA layout is a relatively new concept. Any satisfactory assignment to the variables of the routing Boolean function indicates a legal routing solution in the Boolean based routing issue, which is satisfiable if the layout is routable else routing option is not considered. Recent developments in SAT solving algorithms and efficient implementation approaches have significantly increased capacity of solving routing task of FPGA's and improved efficiency.

A. *FPGA Layout*

In this section, I will show one of the problems that arise in FPGA. Before we can handle the issue, we need to understand the architecture of FPGA. This experiment was held using Xilinx 4000 [4]. The designs are categorized into three parts which are

- CLB - customizable logic blocks
- CB - connection block
- SB - switching block

In the conventional island of FPGA design, shown in Figure 4, CLB performs combinational and sequential logic operation of a circuit and labeled as L. In the C and S blocks, programmable switches comprise the routing resources. CLB pins are connected to channels via programmable switches in C blocks. Signals can pass through or through S blocks that are surrounded by C blocks. The routing capability of an FPGA design is defined by three parameters W, Fc, and Fs. The channel width W determines the number of lines in a vertical or horizontal channel. The number of channels that each logic pin can link to is given as Fc in the adaptability of the C blocks. The number of extra paths that each wire segment enters a S block is given by Fs, which represents the adaptability of the S block. $Fs = 3$ because each wire section entering this S block can link to one track on each of the three other sides. $Fc = 2$ because each logic pin can be linked to any two tracks in the C block

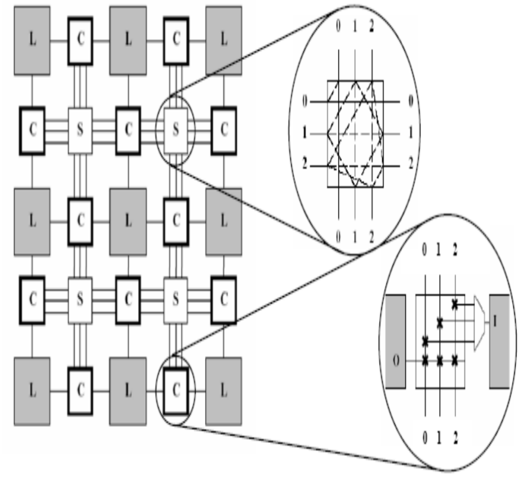

Fig. 4. Island Style FPGA Model

The problem that arises is the signal that going to transmit from L block will pass through all unnecessary S and C blocks. This will cause instance delay when delivering the data. By using Boolean SAT, we can convert a routing task into an atomic Boolean function that is satisfiable (has an assignment of input variables such that the resulting function evaluates to the constant "1") if and only if the design is routable. Any valid assignment to the Boolean function's binary variables yields a legitimate routing solution. The formula that we using to find legitimate routing is [4]

$$R(X) = E(X) * L(X)$$

Where $R(X)$ is Boolean routability function, $L(X)$ is liveness constraint function and $E(X)$ is exclusivity constraint function. Meanwhile, X is an appropriate Boolean vector of binary variables that encode the track number for each two-pin connection. $L(X)$ ensures that for each two-pin connection, at least one global route alternative shall be picked as the final lawful routing solution while $E(X)$ electrically distinct nets with overlapping vertical or horizontal spans in the same channel that are always assigned to different tracks [4].

B. Solve Using ACO Algorithm

We are going to use directed graph as represent of the problem, $G = (V, E)$, with two vertices, $I, j \varepsilon V$, that correspond to the two input options that the input may adopt, 0 or 1. After we create this graph, we literally can solve FPGA routing issue using ACO. The circuits' major inputs are then arranged, and vertices are linked to vertices representing nearby pins. With the starting weights set to 1, each edge $e \varepsilon E$ is assigned a weight that indicates the quantity of pheromone that an ant may release on the routing path. Ants use a probabilistic strategy to explore the graph, with each calculation resulting in a potential routing vector. The stages for our ACO framework are shown in the Figure 5 below.

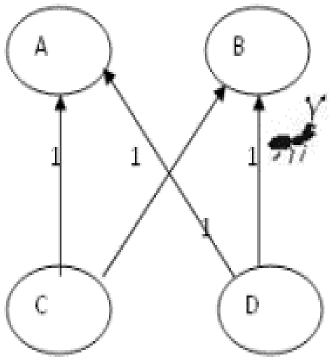

Fig. 5. Graph showing initial weights assigns to the routing paths [4]

1) Create all of the SAT questions and set $iterCount = 2M$ (where M is the number of nets in an FPGA circuit). An $M \times M$ pheromone matrix encodes the pheromone information.
2) Repeat until $iterCount$ does not equal zero.
3) Select a selection of SAT problems and create ants to symbolize each one.
4) Simulate ant movement and choose a subset of ants to place in the queue.
5) Create a routing design and simulate the logic. Permutations depending on the number of nets can be used to determine the routing pattern.
6) Check pheromone levels. By multiplying the pheromone values by an evaporation factor of 1, the pheromone matrix is updated.
7) Stop after all the SAT problems have been solved. Otherwise, decrease $iterCount$ and go to the next step.

In the method, we assume that global circuit routing and placement are known. In the tests, the S block flexibility was modified from 3 to $3 * w$, and the C block flexibility was set to $Fc = W$, allowing the CLB pin to link to any number of tracks. It displays Ant Colony routing findings for circuits that are large enough to be implemented on FPGAs. The results of the experiment reveal that this method to FPGA routing utilizing Ant Colony Optimization outperforms previous SAT-based FPGA routing solvers.

The ACO algorithm is used to increase the FPGA routing performance. The findings shows in Figure 6 reveal that the ACO algorithm is the best approach for routing FPGA chips because it takes less CPU time and uses the shortest wire length. Our method works by having a group of agents work together to investigate various pathways. To effectively undertake this research, a stochastic decision-making technique is developed that combines global and local heuristics. ACO has routed the channel with the fewest number of tracks in all of the circuits used in our experiment. When compared to other classical algorithms, ACO performed better and routed the circuit with the smallest channel width in all of the circuits investigated in our study.

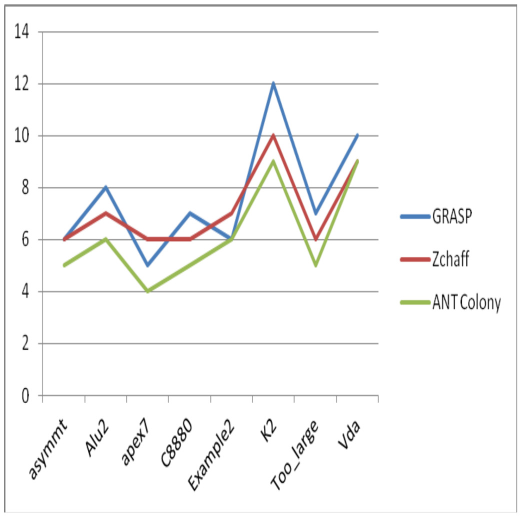

Fig. 6. Showing the comparison among Ant Colony Optimization compares with other algorithm for taking minimum CPU time to route a particular circuits [4]

In this section, the ACO algorithm has proved that it can help in synthesize and optimize in High Level Synthesis. The algorithm also can be applied in any situation that needs to find minimum path taken with least time used.

IV. Conclusion

In the nutshell, this paper shows an algorithm called Ant Colony Optimization. It is one of the algorithms that can be find in Hardware Software Codesign to be specific in scheduling. The explanation should be clear for the reader to understand more about this algorithm and can relate to High Level Synthesis. Furthermore, this paper also compares optimality of the algorithm compared to the others. The diagrams and the tables should help the reader with the visual to imagine and understand more about Ant Colony Optimization.

V. Acknowledgement

I am eternally grateful to Prof. Achim Rettberg, whose inspiration, encouragement, guidance, and support from the beginning to the conclusion helped me to gain awareness and open my eyes to the importance of integration in Hardware Software Codesign in general. I'd also like to express my gratitude, respect, regard, and benefits to everybody who helped me in any manner during the task's execution. I did everything in my power to obtain a better understanding and share it in my paper, and I hope the reader can benefit from it, particularly in this area.

References

[1] G. De Micheli, Synthesis and optimization of digital circuits. New York, N.Y.: McGraw-Hill, 1994.

[2] O. Bozorg-Haddad, M. Solgi, and H. A. Loaiciga, Meta-heuristic and evolutionary algorithms for engineering optimization. Hoboken, NJ: John Wiley amp; Sons, Inc., 2017.

[3] P. Coussy, D. D. Gajski, M. Meredith and A. Takach, "An Introduction to High-Level Synthesis," in IEEE Design Test of Computers, vol. 26, no. 4, pp. 8-17, July-Aug. 2009, doi: 10.1109/MDT.2009.69.

[4] Chopra, Vinay and Amardeep Singh. "Solving FPGA Routing using Ant Colony Optimization with Minimum CPU Time." (2011).