Multiprocessor scheduling and Hu's Algorithm

Amit Chakma
Dept. Electrical Engineering
Hamm-Lippstadt University of Applied Sciences
Lippstadt , Germany
amit.chakma@stud.hshl.de

***Abstract*—We came a long way through the evolution of the multiprocessor environment. Our system has had massive improvement over the last few decades and scheduling algorithms became one of our most important topics to consider about. We use algorithms to efficiently delegate tasks to the processors. The algorithm operates as a resource distributor and controller ensuring that all CPUs are used equally with maximized processor utilization within a given deadline. Hu's algorithm for multi-machine process control systems is considered in this paper. We will also check some advantages and disadvantages for this algorithm and most importantly how some popular algorithms are developed over top of this algorithm.**

I. Introduction

Nowadays most computers run on multiple processors which sometimes also known as parallel processor architecture. It has a significant impact on overall performance since it improves multitasking processes, making them faster and more efficient. Thanks to the rapid innovation of current technology, we users actually benefit a lot from these advancements. Nonetheless, even with improved computer architecture and technology, computers still require help from the software department, which acts as the brains for controlling the architecture. Nowadays software is the most important part to consider for our system and algorithms act as the backbone of our software .

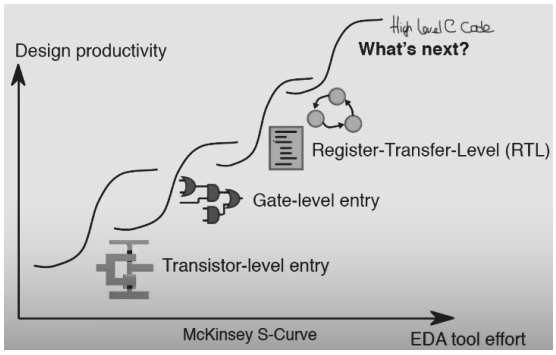

Fig. 1. Mckinsey S-curve for different synthesis level [6].

High-level synthesis is a process of converting a high-level behavior detail in C or C++ into an equivalent design in RTL level. High-level synthesis actually consists of several sub steps and the sub steps are basically pre-processing, scheduling, allocation, binding, data path controller generation and then finally RTL can be found . From RTL level, logic gate level entry can be extracted which mostly shows the gates(e.g XOR, AND gate). After designing the architecture, the model is sent to the manufacturer, which they deliver in transistor transfer entry level or we can say a processor.[fig 1]

The scheduling problem is NP complete which means it is a complex problem [4]. It is very difficult to find an algorithm which will give an optimum solution so there are no polynomial time algorithms available. Exponential time problems are still unsolvable , computer science yet could not discover this kind of problem.So here try to see whether that particular class is solvable or is still NP complete or if it is not then what is the particular subset or the subclass of this scheduling problem which is actually polynomial time solvable.

II. Methods

Hu's algorithm can generate a task execution schedule that takes the least amount of time to complete the task for a given number of processors [8]. Initial input data for generating the best task execution plan is a graph representing the order and relationships of various tasks.

A. Assumptions

To execute Hu's algorithm we need to assume:

1) All operations are of the same type or a multiprocessor executing all the operations. There will be no longer need to take different types of function units (multiplier,divider etc) since only a single multiprocessor can execute all types of operations and we can consider all the nodes are of the same type .
2) They have unit delay or delay of 1, having a single clock.
3) Sequence graph should be a tree.
   There should not be any parallel path .
   Have a single path from each vertex to the sink node. The major difference between a graph and a tree is that, in a graph we will have many paths from a node to sink node but in a tree we will have only one path from a node to the sink node. Figure shows a graph and a tree.

B. Labeling the Nodes

To understand Hu's algorithm , from the source node to the sink node, all nodes must be labeled.

1) A labeling of a sequencing graph consists of marking each vertex with the weight of its longest path to the sink, which is measured in terms of edges

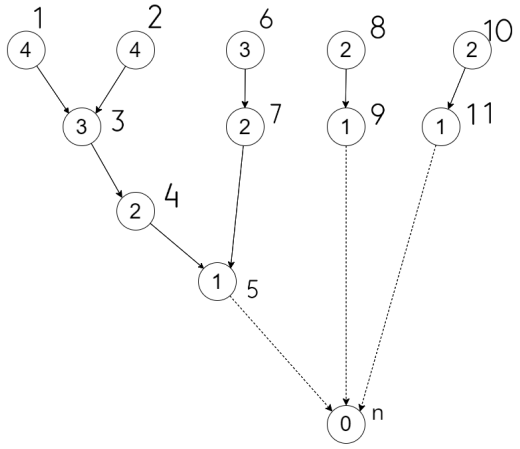

Fig. 2. A tree graph with a single route to sink node

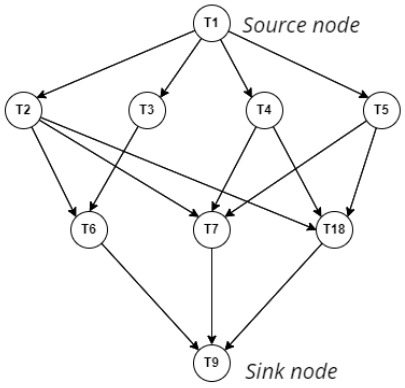

Fig. 3. A generic graph with a multiple route to sink node

2) Let us denote by $\alpha_i, i = (1, 2, 3, ...n)$.

$$\alpha = MAX\alpha_i : 1 \leq i \leq n$$

3) p(j) = Number of vertices with label equal to j.

$$p(j) = |v_i \in V : \alpha_i = j|$$

4) Latency is greater than or equal equal to the weight of the longest path.

So ,In the sequence diagram the maximum distance from the sink node is 4 so α is also 4. P(j) or p(α_n) is the number of nodes that have label j. As an example in the given graph
p(0)=1
p(1)=3
p(2)=4
p(3)=2
p(4)=2

C. Algorithm

As the algorithm is based on tree diagrams, decisions can be made locally. In the Fig.7 there are five operations that need to be scheduled at T=1. If there are 3 processors available then only 3 operations can be selected which should be the 3

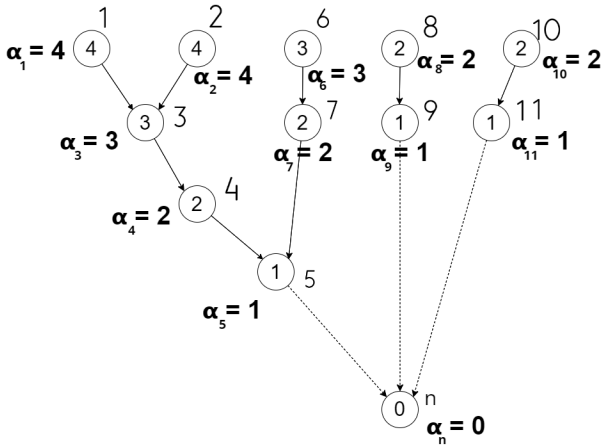

Fig. 4. The diagram shows the α or the distance from the sink node

```
HU( G_s(V, E), a ) {
    Label the vertices;
    l = 1;
    repeat {
        U = unscheduled vertices in V without predecessors
            or whose predecessors have been scheduled;
        Select S ⊆ U vertices, such that |S| ≤ a and labels in S are maximal;
        Schedule the S operations at step l by setting t_i = l ∀i : v_i ∈ S;
        l = l + 1;
    }
    until (v_n is scheduled);
    return (t);
}
```

Fig. 5. Hu's algorithm

maximum alpha at T=1. By selecting the maximum number of labels , it ensures the subsequent nodes will have the earliest possible time to schedule.

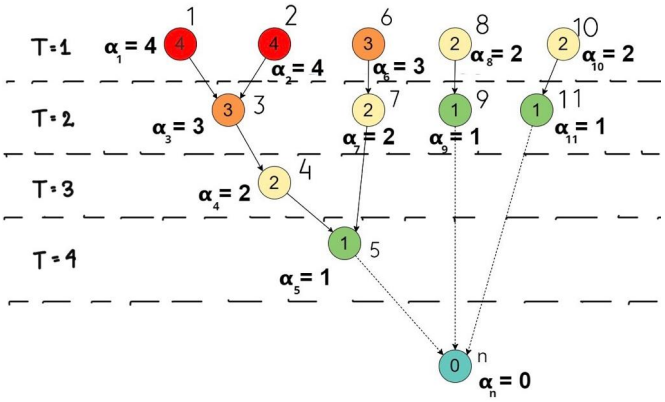

Fig. 6. The diagram shows the number of operations available in four different time set

Now, let us take a look at scheduling
At t=1 ,

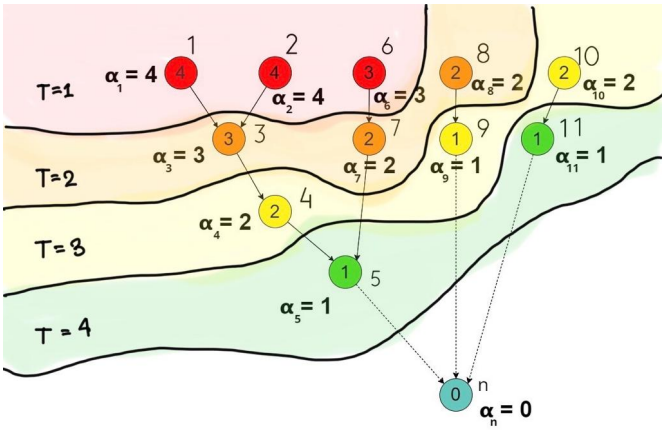

Fig. 7. The diagram shows the scheduling by the algorithm

```
LIST_L( G_s(V, E), a ) {
    l = 1;
    repeat {
        for each resource type k = 1, 2, ..., n_res {
            Determine candidate operations U_{l,k};
            Determine unfinished operations T_{l,k};
            Select S_k ⊆ U_{l,k} vertices, such that |S_k| + |T_{l,k}| ≤ a_k;
            Schedule the S_k operations at step l by setting t_i = l ∀i : v_i ∈ S_k;
        }
        l = l + 1;
    }
    until (v_n is scheduled);
    return (t);
}
```

Fig. 8. The Heuristic Algorithms for Minimize Resource Under latency Constrain

U = {8,10,3,7,9,11,4,5}
S = {1,2,6}
At t=2 ,
U = {10,9,11,4,5}
S = {3,7,8}
At t=3 ,
U = {11.5}
S = {4,9,10}
At t=4 ,
U = {11,5}
S = {11,5}

Hu's algorithm is a very powerful algorithm , it provides an easy and optimum pathway for scheduling . As shown in the diagram 3 resource constraints $\bar{a}$ are required for 4 time steps λ and which is the minimum. Also in other way it can be said 4 time steps are required for 3 resource constraints which makes the algorithm very powerful and optimum. Proving optimum schedule with minimum resource usage under latency constraints (MRLC) and also the minimum-latency schedule under resource constraints (MLRC)[1].

III. Related works

A. Heuristic Algorithms for MLRC

Heuristic Algorithms is a list based scheduling algorithm which is based on Hu's algorithm. It is mostly used on generic graphs but can on be used on tree . Most of the time heuristic algorithms are applied for practical purposes. In the algorithm priority orders are created on some priority function/operations. All the nodes or the operate operations are available in the sequence graph. After identifying the priority function , priority order is developed.

1) The candidate operations :
   those operations of type K whose predecessors have already been scheduled early enough (completed at time step l).
2) The unfinished operations :
   those operations of type k that started at earlier cycles but whose execution has not finished at time step l (multi-cycle operations).
3) Priority list :
   List operators according to some heuristic urgency measure Common priority list: labeled by position on the longest path in decreasing order scheduling

One of the important priority functions is the distance from the sink node and that is what is used for (MLRC) problems. Longest distance from the sink node is calculated and priority value can be found. Based on the priority value , we make an order of this node and maintain a priority list which is based on the decreasing value.

For a given resource bound we need to optimize the latency. We start at time step 1 or l = 1 . For every time step , for each type of resource we identify the ready list $U_{l,k}$ and the running list $T_{l,k}$ and we schedule the running operation.[Fig.8] In the next step we find a subset of the ready operation such that your resource bound is satisfied and this selection is based on the priority. We will run until our sink node is executed.

B. Heuristic Algorithms for MRLC

For MRLC our time constrain is fixed and we need to find optimum resources for our problem.Initially we assume that for all the operators there are minimum numbers of instance/resources available in the hardware. Later when scheduling , if we find that we need to increase our resources for the given latency , we can increase the number of resources. We always start with minimum resources and later if we find critical computation then we increase our resource bound. Initially we check our λ by ALAP algorithm , whether we can schedule the operations within the given time step. In the algorithm it can be seen if $T_0 < 0$ which says if its not a feasible latency bound we cannot schedule any operations. After checking λ we start the main algorithm with l=1 or time step = 1 for all resources K = 1,2,3...n_{res} e.g adder, multiplier. We will then identify the ready list which is the operation that is ready to be scheduled in this time step $T_{l,k}$.The most important thing in the algorithm is to identify the slack in the ready list.

```
LIST_R( G(V, E), λ̄ ) {
    a = 1;
    Compute the latest possible start times t^L by ALAP ( G(V, E), λ̄);
    if ( t_0^L < 0 )
        return (∅);
    l = 1;
    repeat {
        for each resource type k = 1, 2, ..., n_res {
            Determine candidate operations U_lk;
            Compute the slacks {s_i = t_i^L − l ∀v_i ∈ U_lk};
            Schedule the candidate operations with żero slack and update a;
            Schedule the candidate operations requiring no additional resources;
        }
        l = l + 1;
    }
    until (v_n is scheduled);
    return (t, a);
}
```

Fig. 9. The Heuristic Algorithms for Minimize latency Under Resource Constrain

From ALAP we can identify t_i and slack is used to rank the operations, where the slack is the difference between the latest possible start time t_i and the index of the schedule step under consideration l. Lower the the slack,higher the priority it gets . After we determine the slack value of all operations, we identify those operations which are critical or slack 0 . Based on the slack we increase the number of resources , so that other running operation can be scheduled in the same time step. If there are no sun critical operations and if there are some resources available because it has been increased in previous steps, we will select a subset of those ready operations based on their priority.At the end we can step to the next time step (L = L + 1) until our sink node is scheduled .

IV. Conclusion

Hu,s algorithm is one of the most simple and basic algorithm for task scheduling for tree . Hu's algorithm Both MLRC and MRLC constrains and some of the most used practical algorithms are based on Hu's algorithm. As a disadvantage, it is necessary to consider that this algorithm does not provide preemptive schedules or interruptions and sometimes not as practical to use . One other disadvantage is equal duration restriction of all operations, it can lead to some system idle time, but sometimes it is simply impossible to avoid. Heuristic Algorithm is based on Hu"s algorithm. Heuristic Algorithms are very much practical , provides fast and feasible short-term solutions for a optimum scheduling. As we have seen the approach for MLRC and MRLC problems which is very effective for such problems. But also the algorithm too much conditions and sometimes it is not guaranteed to have a optimum solution for a given problem .

References

[1] de Micheli, G., & de Micheli, G. (1994). Synthesis and Optimization of Digital Circuits. McGraw-Hill Education. pp 202-207

[2] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2001). Scheduling Computer and Manufacturing Processes. Springer Publishing.

[3] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory: a survey. Ann. Discrete Math. 5. 287-326. 1979.

[4] Brucker, P. (2007). Scheduling algorithms. Springer.

[5] Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley Professional.

[6] Hofmann, K. (n.d.). Success factors design abstraction synthesis and re-use - ppt download. SlidePlayer. Retrieved May 16, 2022, from https://slideplayer.com/slide/15529244/

[7] Chernigovskiy, A. S., Tsarev, R. Y., Kapulin, D. V. (2017). Scheduling algorithms for automatic control systems for technological processes. In Journal of Physics: Conference Series (Vol. 803, No. 1, p. 012028). IOP Publishing.

[8] T.C. Hu, —Parallel Sequencing and Assembly Line Problems,‖ Operations Research, vol. 9(6), pp. 841-848, 1961.

[9] O. Sinnen, —Reducing the solution space of optimal task scheduling,‖ Computers and Operations Research, vol. 43 (1), pp. 201-214, 2014.