Dynamic Scheduling with Real-Time Requirements

Kajeepan Umaibalan
University of Hamm-Lippstadt
Real$_t imeSystems$
Summer Semester 2023
Lippstadt, Germany
Kajeepan.umaibalan@stud.hshl.de

***Abstract*—This document discusses the topic on dynamic scheduling with real-time requirements and this paper tells us how dynamic scheduling works with a real-time system. This paper also gives a brief explanation on the topic resource constrained dynamic scheduling, Which is a real-time requirement.**

I. Introduction

Real-time solutions play a vital role in numerous industries, including aerospace, automotive, medical, and industrial automation. These industries operate under strict time constraints, requiring tasks to be completed within specific deadlines for reliable and efficient functioning. Adhering to these time specifications is critical to prevent potential consequences such as system malfunctions or compromising safety. To effectively manage task execution and optimize resource utilization while meeting stringent timing requirements, real-time hardware equipped with dynamic scheduling algorithms has emerged as a significant technique.[1]
The objective of this seminar paper is to explore the concept of dynamic scheduling and its significance in real-time hardware systems. The paper will delve into three key aspects: uniprocessor scheduling, multicore scheduling, and resource-constrained dynamic scheduling for real-time requirements. However, the primary focus will be on resource-constrained dynamic scheduling. The paper will provide an in-depth analysis of different methods and algorithms employed for scheduling tasks with limited resources in real-time systems, while also addressing the challenges associated with this approach.
In the upcoming sections, we will provide a comprehensive overview of real-time systems and delve into the intricacies of resource-constrained dynamic scheduling. Our focus will be on highlighting the characteristics and types of real-time jobs, shedding light on their unique requirements. Additionally, we will explore the advantages and disadvantages associated with resource-constrained dynamic scheduling. Furthermore, our discussion will encompass various strategies employed in resource-constrained scheduling to effectively manage limited resources during dynamic scheduling processes.

II. Dynamic scheduling

Dynamic scheduling is a dynamic and adaptive approach that involves the real-time assignment of tasks and resources while a program is in execution. Unlike static scheduling, which predetermines task assignments and resource allocations, dynamic scheduling adjusts and makes decisions on-the-go based on the system's current state and workload. This responsive approach ensures efficient task allocation, as resources are dynamically allocated to meet the changing demands of the system. By adapting to the dynamic nature of the program, dynamic scheduling maximizes system performance and optimizes resource utilization.

A. Uniprocessor Scheduling

Uniprocessor scheduling involves deciding the sequence and allocation of tasks on a single processor within a computer system. In this scenario, multiple tasks contend for the processor's resources, necessitating the use of an efficient scheduling algorithm to establish the execution order of tasks.
Various scheduling algorithms exist for uniprocessor systems, each with its own characteristics and trade-offs.

B. Multicore Scheduling

Multicore scheduling involves managing the allocation and sequencing of tasks across multiple processor cores in a computer system. In a multicore environment, tasks can be executed simultaneously on different cores, enabling parallel processing and enhancing overall system performance. An efficient multicore scheduling algorithm is crucial for effectively assigning tasks to cores and orchestrating their execution, with the goal of optimizing resource utilization and meeting system objectives.

C. Resource Constrained scheduling

Resource-constrained scheduling involves the management and arrangement of tasks in a system where resources such as Memory, Processor, IO devices, Network Bandwith are scarce or limited. In such scheduling, tasks contend for restricted resources like processors, memory, or I/O devices, and an efficient scheduling algorithm is necessary to allocate and optimize the usage of these resources.

III. Resource-Constrained Dynamic Scheduling

Scheduling tasks while considering resource constraints is a challenging and crucial problem. Resource constraints are driven by the fact that resource usage directly impacts

the circuit area in resource-intensive circuits and forms a significant portion of the overall area in most other circuits.

The resolution of scheduling problems under resource constraints allows for the determination of trade-off points between area and latency. However, two practical difficulties arise in this process. Firstly, the resource-constrained scheduling problem is computationally complex, and only approximate solutions are feasible for reasonably sized problems. Secondly, when dealing with circuits that are not primarily resource-dominated, the area-performance trade-off points are influenced by other factors in addition to resource constraints.[1]

Resource-constrained scheduling is an approach that aims to minimize the number of control steps required for a given design cost. The design cost can be quantified by factors such as the number of functional and storage units, the count of NAND gates, or the area occupied by the chip layout. This approach, known as resource-constrained scheduling, focuses on optimizing the allocation of resources while considering the limitations imposed by the available resources.[3]

IV. ASAP AND ALAP SCHEDULING

A. ASAP

ASAP (As Soon As Possible) scheduling is a task scheduling technique that aims to determine the earliest feasible start times for tasks within a system. The objective of ASAP scheduling is to minimize the overall completion time or latency of the entire schedule.

Scheduling involves allocating start times to tasks considering their dependencies and resource availability. The primary aim is to schedule tasks to commence at the earliest feasible moment while respecting precedence constraints and resource limitations. The scheduling process starts by identifying independent tasks that can initiate immediately. Once these tasks are scheduled, their dependent tasks become eligible for scheduling, and this process continues until all tasks are assigned start times.

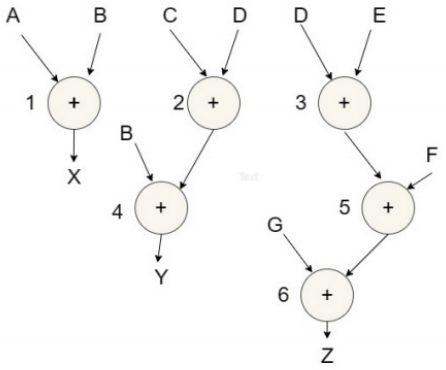

Fig. 1. Data Flow graph based on [4]

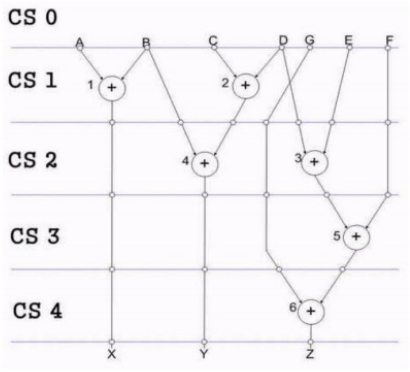

Fig. 2. ASAP Solution based on[4]

In this illustration, we present a simple example to demonstrate the algorithm. The data-flow graph depicted in Figure 1 consists solely of addition operations, serving as a basis for explaining the fundamental concept of the technique. In practical scenarios, there may be numerous operations combined together, resulting in the definition of resource constraints across multiple subsets. For our example, we assume the existence of two available adders.[3]

During the first control step (CS1), three operations, labeled as 1, 2, and 3, are eligible for scheduling. As the ASAP scheduling algorithm does not incorporate a selection category, we randomly choose two operations, namely 1 and 2, for scheduling in CS1. We then proceed to follow the algorithm to create the entire schedule. An example solution generated by ASAP scheduling is presented in Figure 2.[4]

B. ALAP

ALAP (As Late As Possible) scheduling is a task scheduling technique that aims to find the latest feasible start times for each task in a given system. The objective of ALAP scheduling is to maximize the amount of slack time available in the schedule, which represents the time that tasks can be delayed without affecting their deadlines.[3]

Scheduling considers task dependencies and system constraints to determine the latest possible start times. It starts by identifying tasks that have no successors or dependencies, and schedules them at the latest feasible time. As the scheduling process proceeds, the start times of preceding tasks are adjusted to accommodate the dependencies and constraints.[3]

By considering Figure 1 we can also create a ALAP solution. Figure 3 shows a solution of ALAP

V. THE INTEGER LINEAR PROGRAMMING MODEL

Inter Linear Programming is a mathematical approach that combines integer programming and linear programming to

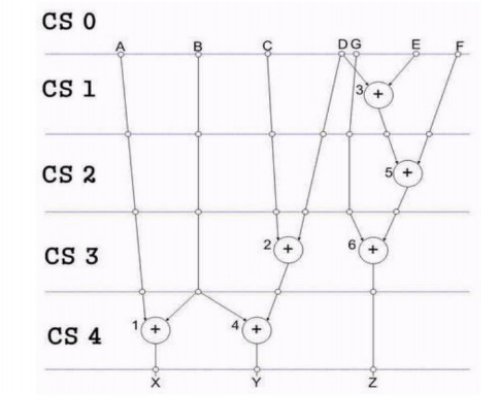

Fig. 3. ALAP Solution based on[4]

address scheduling problems. It aims to optimize the allocation of resources and time in scheduling tasks. By formulating the scheduling problem as an interlinear programming model, it becomes possible to find optimal solutions by leveraging the strengths of both integer programming and linear programming techniques.[1]

A representation of the scheduling problem under resource constraints can be achieved by utilizing binary decision variables indexed as X = Xi — i = 0, 1, ..., n; j = -1, -2, ..., A + 1. The range of the indexes is justified by considering the source and sink operations, and assuming the schedule begins on cycle 1. The variable notation is presented as a set to enhance clarity, as opposed to a matrix notation. It is important to note that the value A represents an upper bound on the latency, as the precise schedule latency is currently unknown. To compute this bound, a fast heuristic scheduling algorithm like a list scheduling algorithm can be employed.[1]

The value of binary changes to one (Xil=1) only when Operation or a process(Vi) starts in the step(l) of the schedule at time (ti). By this conclusion we can write an equation

$$_{il} = \delta_{ti}$$

. We then use

$$\Sigma_i$$

to denote the summations of all operations and we use

$$\Sigma_l$$

to denote all schedule steps to make it look simple. On corresponding unconstrained problem ASAP and ALAP Algorithms can be use to calculate the upper and lower bound on start times.
Now we consider to model resource constrained scheduling problem on constraints on binary values. The start time of each operation is unique.

$$\Sigma X_{il} = 1, i = 0, 1, ..., n$$

So the start time of any operations vi can be stated V interms of

$$X_{il}$$

as

$$t_i = \Sigma_l l.X_{il}$$

As the second step is to satisfy the sequencing relationship represented by Gs(V,E) therefore,

$$\Sigma_l l.X_{il} >= \Sigma_l l.X_{jl} + d_j, i, j = 0, 1, .., n$$

Let's revisit the sequencing graph depicted in Figure 1 In this scenario, we consider two types of resources: a multiplier and an ALU (Arithmetic Logic Unit) capable of performing addition, subtraction, and comparison operations. Both resources have a fixed execution time of one cycle. Additionally, we assume that there is a limit of 2 units for each resource type, denoted as a1 = 2 for the multiplier and a2 = 2 for the ALU.

We employ a heuristic algorithm called list scheduling to determine that the maximum latency is 4 steps. By applying the ASAP (As Soon As Possible) and ALAP (As Late As Possible) algorithms to the unconstrained model, we can establish lower and upper bounds on the start times, respectively. Interestingly, we observe that the latency calculated by the unconstrained ASAP algorithm matches the upper bound computed by the heuristic algorithm, indicating that the heuristic solution is optimal. However, for the purpose of presenting the ILP (Integer Linear Programming) constraint formulation, we temporarily disregard this observation.[1]

Let's examine the constraint sets individually using below equations. Starting with the first set, it is required that each operation initiates only once showed below,.

$$X_{0.1} = 1$$

$$X_{1.1} = 1$$

$$X_{2.1} = 1$$

$$X_{3.2} = 1$$

$$X_{4.3} = 1$$

$$X_{5.4} = 1$$

$$X_{6.1} + X_{6.2} = 1$$

$$X_{7.2} + X_{6.2} = 1$$

$$X_{8.1} + X_{8.2} + X_{8.3} = 1$$

$$X_{9.2} + X_{9.3} + X_{9.4} = 1$$

$$X_{10.1} + X_{10.2} + X_{10.3} = 1$$

$$X_{11.2} + X_{11.3} + X_{11.4} = 1$$

$$X_{n.5} = 1$$

Then we can do for resource constraints

$$X_{1.1} + X_{2.1} + X_{6.1} + X_{8.1} <= 2$$

$$X_{3.2} + X_{6.2} + X_{7.2} + X_{8.2} <= 2$$

$$X_{7.3} + X_{8.3} <= 2$$

$$X_{10.1} <= 2$$

$$X_{9.2} + X_{10.2} + X_{11.2} <= 2$$

$$X_{4.3} + X_{9.3} + X_{10.3} + X_{11.3} <= 2$$

$$X_{5.4} + X_{9.4} + X_{11.4} <= 2$$

Any set of start times that satisfies these constraints can be considered a feasible solution. In this specific scenario, the immobility of the sink node indicates that any feasible solution is also an optimal solution

By minimizing this expression, we can achieve an optimum solution that meets the real-time requirements and optimizes the scheduling outcome for the given values of c.

Also when considering c=[l,...,1], equivalises optimum solution

$$X_{6.1}+2X_{6.2}+2X_{7.2}+3X_{7.3}+X_{8.1}+2X_{8.2}+3X_{8.3}+2X_{9.2}$$
$$+3X_{9.3} + 4X_{9.4} + x_{10.1} + 2X_{10.2} + 3X_{10.3} + 2X_{10.2}$$
$$+3X_{11.3} + 4X_{11.4}$$

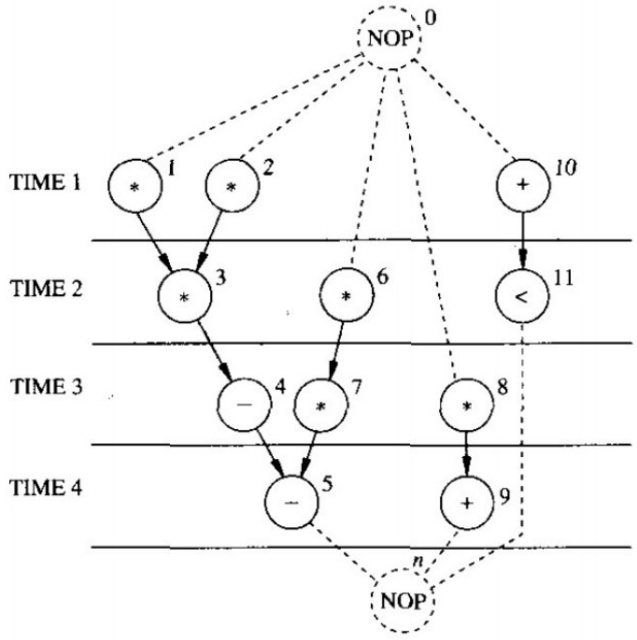

Fig. 4. Optimun schedule under resource constraints based on[1]

Considering task 6 in the equation represented as,

$$X_{6.1} + 2X_{6.2}$$

there are specific time slots available for its execution. Task 6 can be scheduled to run at time 1 or time 2. However, it cannot be executed at time 7 due to the presence of a sequential task. Task 5 depends on the output of task 6, and task 7 also follows task 6. As a result, task 6 is not able to run in time slot 3, as it would violate the sequential dependencies.

$$x_{10.1} + 2X_{10.2} + 3X_{10.3} + 2X_{11.2} + 3X_{11.3} + 4X_{11.4}$$

If we consider tasks 10 and 11, it is possible to shift task 11 to time slot 4. By doing so, it opens up an opportunity for task 10 to be moved to time slot 3.

Tasks 1 and 2 remain in time slot 1, while in time slots 2, 3, and 4, task 3 is dependent on the outputs of tasks 1 and 2. Similarly, in time slot 3, task 4 is scheduled, which relies on the outputs of tasks 1, 2, 3, and 6. Likewise, in time slot 4, task 5 is running. Considering all these dependencies, tasks 1 and 2 cannot be shifted to different time slots.

VI. Inter Linear Programming Solution

The ILP (Integer Linear Programming) formulation of constrained scheduling problems offers several advantages. Firstly, it provides an exact solution to the scheduling problems, ensuring accuracy in the results. Secondly, general-purpose software packages can be utilized to solve the ILP, making it accessible and convenient. Lastly, incorporating additional constraints and extending the problem scope, such as scheduling pipelined circuits, is relatively straightforward with the ILP formulation. However, there are drawbacks to the ILP approach, primarily due to its computational complexity. The number of variables and inequalities, as well as their tightness, impact the computational efficiency of finding a solution. Researchers have attempted to mitigate this issue by analyzing the ILP model in detail and developing tighter constraints that reduce the feasible solution space, thus decreasing computing time. Additionally, these refined constraints often yield integer or binary solutions, even when the original integer constraints are relaxed. This allows for the use of linear program (LP) solvers instead of ILP solvers, enabling faster computation of an exact solution or bounds on the solution. These bounds can serve as starting points for subsequent ILP solver applications to find an optimal solution.

While practical implementations of ILP schedulers have shown efficiency for medium-scale examples, they may struggle to solve problems with a large number of variables or constraints.[1]

A. Trade-offs between resource utilization and task deadlines

Optimizing the allocation of system resources and meeting task deadlines is a critical compromise in dynamic scheduling with real-time demands. Resource utilization involves maximizing the usage of available system resources like processors, memory, and I/O devices to achieve high efficiency and throughput. Conversely, task deadlines indicate the time restrictions within which tasks must be finished to satisfy real-time requirements.

A key trade-off arises when resource utilization is pushed to its limits, as it can lead to prolonged task execution time or resource contention, thereby putting task deadlines at risk. When system resources are heavily utilized, tasks may face delays or fail to meet deadlines due to resource bottlenecks or competition for resources. In these scenarios, it becomes essential to prioritize critical tasks or allocate additional resources to ensure timely completion of important deadlines.[6]

VII. Advantages and disadvantages of dynamic scheduling with real-time requirements

Dynamic scheduling offers the advantage of being flexible and adaptable. It can effectively handle diverse workloads and dynamically adjust task priorities, scheduling policies, and resource allocations to accommodate changing real-time requirements. This adaptability allows the system to respond promptly and effectively to dynamic scenarios. Additionally, dynamic scheduling enhances system performance by giving priority to critical tasks and allocating resources accordingly. This results in reduced latency, increased throughput, and overall improved system performance.

Disadvantages of dynamic scheduling in real-time systems. The complexity of designing and implementing dynamic scheduling algorithms to meet real-time requirements can pose a challenge. These algorithms must carefully consider stringent timing constraints, task dependencies, and resource limitations, which can significantly increase the intricacy of the system.

Furthermore, dynamic scheduling algorithms often introduce computational overhead. They require substantial computational resources to make real-time scheduling decisions, which can impact system performance and introduce latency, especially in environments with limited resources. Moreover, the dynamic nature of scheduling real-time tasks adds scheduling overhead. The system needs to continuously evaluate and update task priorities and resource allocations, potentially reducing the available processing time for actual task execution.

VIII. Applications of Dynamic Scheduling with Real-Time Requirements

Dynamic scheduling with real-time requirements finds applications in various domains where timely task execution is critical. Some notable examples; [2]

- Chemical and nuclear plant control
- Control of complex production processes
- Railway switching systems
- Automotive applications
- Flight control systems
- Environmental acquisition and monitoring
- Telecommunication systems
- Medical systems

IX. Conclusion

Dynamic scheduling with real-time requirements is a highly significant and intricate research area within computer systems and scheduling algorithms. The need to effectively allocate tasks and manage resources in real-time systems is paramount in order to meet strict timing constraints. Throughout this research paper, we have delved into numerous aspects pertaining to dynamic scheduling in the context of real-time requirements. By examining various scheduling algorithms and considering the challenges and tradeoffs involved, we have gained insights into optimizing system performance while satisfying stringent timing constraints. Moving forward, further research and innovation in scheduling algorithms, optimization techniques, and resource management strategies will continue to enhance the efficiency and effectiveness of real-time systems. By addressing the complexities and intricacies of dynamic scheduling with real-time requirements, we contribute to the advancement of reliable and high-performance computing systems across diverse domains.

References

[1] Synthesis and optimisation of digital circuits By Giovanni De Micheli ISBN:978-0-07-016333-1 Published:01 January 1994

[2] Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications By Giorgio C Buttazzo 2013.

[3] Baruch, Zoltan. “Scheduling Algorithms for High-Level Synthesis.” (2003).

[4] Implementation Of aScheduling and Allocation Algorithm for Hardware Evaluation Kangmin Chen LiTH-ISY-EX–05/3754–SE Linköping 2005

[5] Lozoya, Camilo Martí, Pau Velasco, Manel Fuertes, Josep Martín, Enric. (2013). Resource and performance trade-Offs in real-Time embedded control systems. Real-Time Systems. 49. 10.1007/s11241-012-9174-9.

X. Affidavit

I hereby confirm that I have written this paper independently and have not used any sources or aids other than those indicated. All statements taken from other sources in wording or sense are clearly marked. Furthermore, I assure that this paper has not been part of a course or examination in the same or a similar version.

Umaibalan Kajeepan
Name,Vorname
Last Name, First Name

Lippstadt 05/06/2023
Ort,Datum
Location, Date

Unterschrift
Signature