Scheduling Pipelined Circuits

Hochschule Hamm-Lippstadt

Ravindu.athukorala@stud.hshl.de

Summer Semester 2023

23.04.2023

Ravindu Athukorala

***Abstract*—The goal of writing this research paper is to give a clear overview about scheduling pipelined circuits. we will be discussing about the scheduling basics, scheduling techniques, performance, real world applications and about the future of pipelined circuits**

I. Introduction

Pipelining is a crucial technique employed in digital circuit design to enhance performance by increasing the throughput and reducing the latency of computations. It allows for the parallel execution of multiple instructions or operations by breaking them down into smaller stages and overlapping their execution. By dividing a complex task into simpler sub-tasks and executing them concurrently, pipelining maximizes the utilization of hardware resources and minimizes idle time. [1]

The primary motivation behind utilizing pipelining in circuit design is the ever-increasing demand for faster and more efficient digital systems. As technology advances, the need for processing large volumes of data, executing complex algorithms, and meeting stringent performance requirements becomes more prominent. Pipelining offers a scalable solution to these challenges by enabling the execution of multiple instructions simultaneously, leading to substantial improvements in system performance. [1]

Functional pipelining is a newly introduced synthesis concept in the sehwa system, specifically designed for real-time DSP algorithms. In such applications, there is typically an unlimited number of tasks to perform, and the latency between consecutive tasks must be minimized to meet high throughput requirements. [1]

During the execution of a functional pipeline, tasks are initiated at fixed intervals known as the initiation interval. The latency, measured in clock cycles, is calculated by dividing the initiation interval by the clock cycle duration. [1]

In contrast to traditional structural pipelining, functional pipelining allows stages in different control steps to share hardware resources. This characteristic increases the utilization rate of hardware resources and reduces overall hardware costs. Since the number of tasks is unlimited, we can consider them as iterations within an implicit loop known as the time loop. Therefore, each task is also referred to as an iteration. [1]

II. Current Pipeline Scheduling Techniques

The majority of existing pipeline scheduling techniques employ two basic approaches for preprocessing: ”as soon as possible” scheduling and ”as late as possible” scheduling. In ”as soon as possible” scheduling, operations in the Control Data Flow Graph (CDFG) are scheduled sequentially from the first control step to the last. An operation is considered ready when all its predecessors have been scheduled. This procedure assigns all ready operations to the current control step and proceeds to the next control step. [2]

A. ASAP(a)

In the ”as soon as possible” scheduling approach, the scheduling of operations in the Control Data Flow Graph (CDFG) follows a sequential order, progressing from the initial control step to the final one. In this method, each operation is deemed ready for scheduling only when all of its predecessors have been successfully scheduled. Once an operation satisfies this criterion, it is assigned to the current control step and subsequently moves on to the next control step. By applying this procedure, all operations that are ready for execution at any given control step are allocated to that particular step, ensuring a systematic and step-by-step progression throughout the scheduling process. This methodology allows for efficient coordination and sequencing of operations within the CDFG, facilitating the smooth execution of tasks and enabling effective utilization of available resources. [2]

B. ALAP(b)

The scheduling technique known as ”as late as possible” (ALAP) follows a procedure that is quite similar to ”as soon as possible” (ASAP) scheduling. However, there is a key distinction: ALAP scheduling arranges operations in the reverse order, starting from the last control step and moving towards the first. In ALAP scheduling, an operation is scheduled to the subsequent control step only when all of its successors have been scheduled. This approach aims to maximize the utilization of available time by delaying the execution of operations until the latest possible moment. By scheduling operations in this manner, ALAP scheduling can effectively optimize resource allocation and ensure efficient usage of available resources. To illustrate the differences between ASAP and ALAP scheduling, refer to Figure 1, which provides a visual example of how the two scheduling techniques are applied. [2]

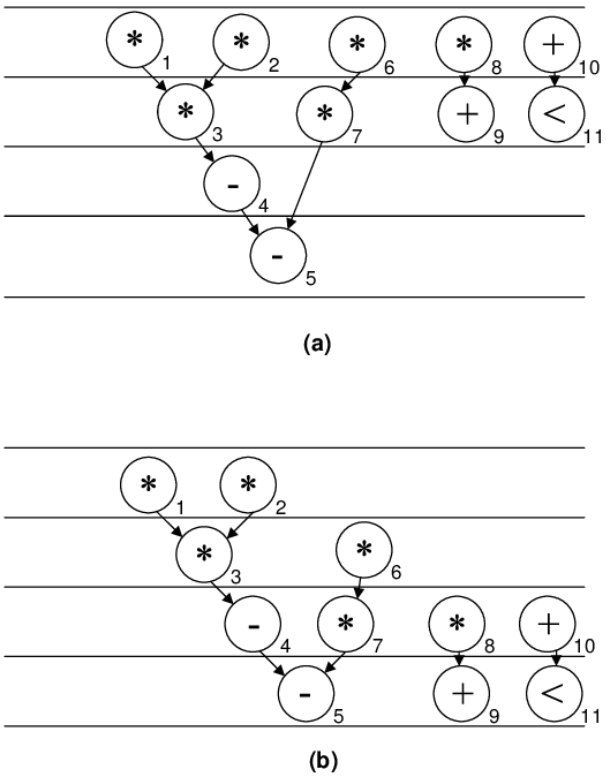

Fig. 1. ASAP/ALAP Example [3]

C. Time Constraint Scheduling

The time-constrained scheduling problem can be described as the task of finding an optimal schedule with the least cost, given a maximum number of control steps and a set of constraints. In this problem, the cost primarily focuses on the function units used in the data path, such as interconnections, registers, and function units themselves. However, for simplicity, we will only consider the cost of function units in this particular formulation, while taking other factors into account in the next section. [2]

To minimize the cost of function units, it is desirable to fully utilize all available function units in the system. This means that operations of the same type should be evenly distributed across all control steps. Our model aims to achieve this objective by minimizing the maximum number of operations of the same type within each control step. By distributing the workload evenly and avoiding excessive concentration of operations, we can optimize the utilization of function units and ultimately minimize the overall cost associated with their usage. [2]

D. Resource Constraint Scheduling

The resource-constrained scheduling problem can be formally defined as follows: given a specified maximum number of resources, the objective is to determine the fastest possible schedule that satisfies the given set of constraints. In this context, the resources refer to the number of function units available, such as adders, multipliers, ALUs (Arithmetic Logic Units), and buses. While registers and interconnections also contribute to the overall system area, specifying them as resource constraints can be complex. [2]

To tackle the resource-constrained scheduling problem, a comprehensive approach involves four substeps:

1) List Scheduling: This step aims to determine an upper limit on the number of control steps required for the schedule. By carefully considering the available resources, the goal is to optimize the scheduling process while ensuring that the number of control steps does not exceed the specified limit. [2]

2) Modified ASAP (As Soon As Possible): This substep involves determining the earliest possible time at which each operation can be executed. By considering the dependencies and constraints, the objective is to schedule the operations as early as possible, minimizing any potential delays or dependencies. [2]

3) Modified ALAP (As Late As Possible): In this substep, the focus is on determining the latest possible time at which each operation can be executed. By considering the dependencies and constraints, the goal is to schedule the operations as late as possible while still meeting the overall timing requirements. [2]

4) Integer Linear Programming (ILP): The final substep involves formulating an Integer Linear Programming problem to minimize the number of control steps required for the data path. By leveraging mathematical optimization techniques, the objective is to find an optimal schedule that minimizes the number of control steps while satisfying all the given constraints. [2]

By following these substeps in a resource-constrained scheduling approach, it becomes possible to devise an efficient schedule that maximizes resource utilization and minimizes the overall execution time of the system.

III. Functional Pipelining

The presence of a pipelined data path enables concurrent execution of multiple tasks. In this scenario, two consecutive tasks can begin at a specific interval known as the latency of the pipelined data path. [4]

In the context of the given latency 1, the operations occurring in control steps j + pl (where p = 0, 1, 2, ...) are executed

simultaneously. It is important to note that these simultaneously executed operations cannot utilize the same function units. As a result, constraint (2) is revised to accommodate this requirement. [4]

$$\sum_{p=0}^{\lfloor (s-j)/l \rfloor} \sum_{o_i \in FU_{lk}} x_{i,j+pl} \leq M_{lk}, \qquad \text{for } 1 \leq j \leq l,\ 1 \leq k \leq m$$

Fig. 2. Equation []

IV. Loop Folding

Loop folding and functional pipelining share a common objective in improving the performance of computations by leveraging parallelism. However, there are important distinctions between the two techniques. [2]

Loop folding involves executing multiple iterations of a loop concurrently, while considering data dependencies between the iterations. In contrast, functional pipelining focuses on executing multiple independent instances of a function concurrently, without any data dependencies between them. This distinction leads to differences in the achievable latency and resource requirements. [2]

In functional pipelining, the latency of a pipelined data path can be made arbitrarily small as long as there are sufficient resources available. Each instance of the function can progress through the pipeline independently, resulting in a reduced overall execution time. As long as resources are unlimited, functional pipelining allows for high parallelism and efficient resource utilization. [2]

On the other hand, loop folding takes into account data dependencies between loop iterations. This means that the execution of loop iterations needs to consider the dependencies and ensure correct ordering of operations. The latency, or loop length, in loop folding depends on the number of available resources and the structure of the Data Flow Graph (DFG) representing the computation. The presence of data dependencies introduces constraints on the parallel execution of loop iterations, and the achievable latency is influenced by the need to maintain correct data flow. [2]

While both loop folding and functional pipelining aim to exploit parallelism, the consideration of data dependencies in loop folding adds complexity to the execution. It requires careful analysis and handling of dependencies to ensure correctness and achieve optimal performance. The achievable latency in loop folding is affected by both the available resources and the characteristics of the computation, as represented by the DFG. [2]

Consider a loop with a fixed number of iterations, n_1, and the execution delay is equal to the product of the number of iterations and the loop body's latency, λ_1. We can potentially introduce pipelining within the loop body itself. By using a local data introduction interval, δ_1, that is smaller than the latency, λ_1, the overall execution delay of the loop can be approximately n_1 multiplied by δ_1, which is smaller than n_1 multiplied by λ_1. It's important to account for the overhead involved in starting the pipeline. Therefore, the modified equation for the loop execution delay is $(n_1 + \left\lfloor \frac{\lambda_1}{\delta_1} \right\rfloor) - 1$, where $\left\lfloor \frac{\lambda_1}{\delta_1} \right\rfloor$ represents the integer division of λ_1 by δ_1. This modified equation considers the additional cycles required to initiate and manage the pipeline, providing a more accurate estimation of the loop execution delay. [4]

Loop folding offers a way to decrease the latency of a scheduled sequencing graph that includes loops, even if they are nested. By recognizing the hierarchical structure of sequencing graphs, loop folding can be seen as the process of introducing pipeline stages within the loop body. The objective is to minimize the execution delay of the iteration vertex at the higher level in the hierarchy. In other words, loop folding aims to overlap and parallelize the execution of loop iterations to optimize performance. [4]

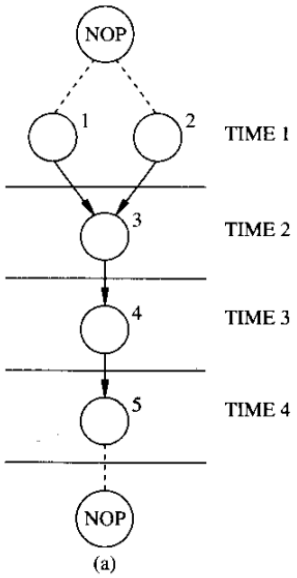

Fig. 3. Sequence graph of loop body [4]

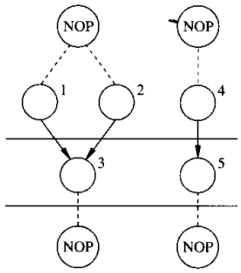

Fig. 4. Sequencing graph after loop foalding [4]

V. Applications

- Digital Signal Processing
- Microprocessors
- High Performance Computing
- Network And Communication Systems

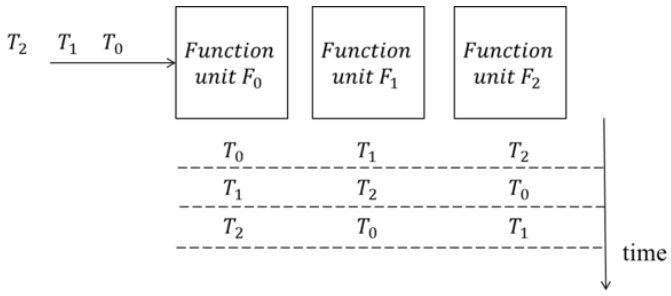

Fig. 5. DSP Pipelining Example [5]

VI. Future Directions

A. Nested Loop Folding

Current loop folding techniques primarily concentrate on folding single time loops. The objective of loop folding is to identify a regular initiation pattern that minimizes the control area. When dealing with nested loops, most existing techniques suggest fully expanding all inner loops before folding, referred to as the full expansion method. However, the data path resulting from the full expansion approach can become burdened with intricate interconnections and multiplexing, leading to an increase in controller area. This method may not be optimal in terms of turnaround time and resource costs. Therefore, there is a need to develop a more efficient algorithm for folding nested loops of a general nature. [1]

B. Pipeline Scheduling With System Partitioning

Pipelined designs often exceed the capacity of a single chip, necessitating system partitioning to divide the behavioral description into manageable parts that can fit on individual chips. The main goal of system partitioning is to achieve a time-correct design, which requires performing the partitioning task prior to scheduling.To optimize for time efficiency, it is more advantageous to perform system partitioning and pipeline scheduling concurrently. Nevertheless, simultaneously executing these two tasks for large problems may not be practical.Because of that reason,there is a need for efficient heuristic techniques that can handle the simultaneous system partitioning and pipeline scheduling, enabling the design process to be both time efficient and feasible for large-scale problems. [1]

VII. conclution

In conclusion, the scheduling of pipelined circuits plays a crucial role in optimizing the performance and efficiency of complex computational systems. Through careful analysis, modeling, and algorithmic techniques, researchers and practitioners have made significant advancements in addressing the challenges associated with scheduling pipelined circuits.

One key consideration in pipelined circuit scheduling is the management of data dependencies and resource constraints. Efficient scheduling algorithms aim to minimize resource conflicts, balance workloads, and maximize parallelism within the pipeline stages. Techniques such as list scheduling, ASAP (As Soon As Possible), ALAP (As Late As Possible), and ILP (Integer Linear Programming) have been developed to address these challenges effectively.

Additionally, time and resource constraints are crucial factors in scheduling pipelined circuits. Meeting timing deadlines while optimizing resource utilization is a critical objective. Researchers have explored various techniques, including time-constrained scheduling algorithms, to find optimal schedules that satisfy given time constraints and minimize costs.

Furthermore, loop folding has emerged as an effective technique for improving the performance of pipelined circuits. By concurrently executing multiple loop iterations and exploiting parallelism, loop folding reduces loop control overhead and increases computation throughput. However, careful consideration of data dependencies and resource allocation is necessary to ensure correct execution and maximize the benefits of loop folding.

In summary, scheduling pipelined circuits involves addressing data dependencies, resource constraints, and timing considerations. Through the development of sophisticated algorithms and techniques like loop folding, researchers have made significant progress in optimizing the performance and efficiency of pipelined systems. However, there are still opportunities for further research to explore novel scheduling strategies and to address emerging challenges in the ever-evolving landscape of pipelined circuit design and optimization.

VIII. Acknowledgment

I would like to express my sincere gratitude to all those who have contributed to the completion of this research paper on scheduling pipelined circuits. I would like to acknowledge the support and guidance of our research advisors and Professor Achim Rettberg, whose expertise and insights were invaluable in shaping this study. Lastly, I would like to acknowledge the collective effort of the research community in advancing the field of pipelined circuit scheduling, as their work has been instrumental in shaping and inspiring my research endeavors

References

[1] Yu-Chin Hsu and Yuang-Long Jeang, "Pipeline scheduling techniques in high-level synthesis," Sixth Annual IEEE International ASIC Conference and Exhibit, Rochester, NY, USA, 1993, pp. 396-403, doi: 10.1109/ASIC.1993.410746.

[2] C. -T. Hwang, J. -H. Lee and Y. -C. Hsu, "A formal approach to the scheduling problem in high level synthesis," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 10, no. 4, pp. 464-475, April 1991, doi: 10.1109/43.75629.

[3] ResearchGate, "Figure 3.2: (a) ASAP scheduling (b) ALAP scheduling," [Online]. Available: https://www.researchgate.net/figure/a-ASAP-scheduling-b-ALAP-schedulingfig12281567647.

[4] Mitcheli, G.D. '5.6', in Synthesis And Optimization Of Digital Circuits.

[5] Pipelining (DSP implementation) (2022) Wikipedia. Available at: https://en.wikipedia.org/wiki/Pipelining

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.

Athukorala,Ravindu

Name,Vorname
Last Name, First Name

Lippstadt,05.06.2023

Ort,Datum
Location, Date

Unterschrift
Signature