

Force Directed Scheduling

1st Pisula Mayan Guruge
Hardware Software Codesign
Summer Semester 2024
Department of Electronic Engineering
Hamm-Lippstadt University of Applied Sciences
deundara-palliya-pisula-mayan.guruge@stud.hshl.de

I. ABSTRACT

Task scheduling is one of the core problems in the field of computer science and engineering. It involves providing resources to tasks to achieve an optimal outcome, optimizing it in such a way that it has a minimum finish time and maximum utilization of resources. Force Directed Scheduling (FDS) plays a critical role in Hardware-Software Codesign because it optimizes resource allocation and effectively schedules tasks. This seminar paper provides the very basics of FDS and compares its relationship with the other methodologies of scheduling, applications, benefits, and limitations. Finally, the importance of FDS, for design efficiency and electronic systems performance, will be explained through real case studies and by using mathematical modeling.

II. INTRODUCTION

Task scheduling is another typical challenge in computer systems, manufacturing, and project management. In computer systems, the problem of task scheduling involves allocating available processing resources, processors and memory, to tasks to maximize performance and high resource utilization. The main objective of this is to minimize the time for the completion of tasks while maintaining constraints on resources and task dependencies. Hardware-Software Codesign is one technique of design that combines the studies of hardware and software components to come up with a system design that ensures optimal performance. Effective scheduling is therefore needed, as it dictates which sequence the tasks are completed in. Hence, it impacts time-based constraints and resource usage. Therefore, Force Directed Scheduling emerges to be one of the potential strategies since it deals with trying to balance computing loads with minimal delays. This report provides an analysis of FDS, its methodology, applications in modern electronic design, and advantages over conventional scheduling methods. [1] [6]

A. Evolution of Scheduling Techniques

Important milestones have defined the development and advancement of hardware-software co-design scheduling approaches. The early methods were based on very basic fundamental approaches such as First Come, First Served (FCFS) and Round Robin which were simple but inefficient in complex systems. As the complexity of electronic systems increased,

several sophisticated strategies began to appear, such as Priority Scheduling, Earliest Deadline First, and Critical Path method. These techniques were targeted at overcoming the weaknesses of its predecessors by combining factors like priority, task deadlines, and dependencies. These new solutions, however, remained a problem with respect to resource utilization optimization and execution time reduction in highly integrated hardware-software setups. Therefore, the need to develop more dynamic and balanced approaches to scheduling had to be established. This led to the development of Force Directed Scheduling. Subject to the "forces" operating in task dependency and resource restrictions, FDS offered a drastic improvement over earlier methodologies and has turned out to be an essential tool within modern co-design practice.

III. HARDWARE-SOFTWARE CODESIGN

Hardware-Software Codesign is a method of development that combines hardware and software to maximize performance and functionality. Designers can use this approach to analyze the trade-offs between hardware and software implementations, as a balanced design needs to account for performance, power consumption, and cost constraints. A codesign process includes the following steps: [5]

- 1) System specification: Describe the system's requirements and constraints.
- 2) Partition: Separate the system into hardware and software components.
- 3) Hardware-Software System Co-simulation for Functional Testing.
- 4) Co-Synthesis of Hardware/Software Components.
- 5) Integrating Synthesized Components into a Complete System.

A. Advantages of Hardware-Software Codesign

The inherent advantage of hardware-software codesign is that system performance can be optimized regarding interactions between hardware and software parts. The outcome will be far more effective designs that meet strict performance, power, and cost constraints. Additionally, codesign helps in identifying and mitigating potential concerns at the very earliest stages of design, reducing the risk of an expensive redesign later. These tools also improve communication and collaboration between different design teams to come up with more coherent, better-integrated final products.

B. Challenges in Hardware-Software Codesign

Despite its benefits, hardware-software codesign presents several challenges. The necessity for strict coordination between the hardware and software teams might result in communication breakdowns and conflicts. Furthermore, the complexities of merging hardware and software components may result in lengthier design timelines and increased development costs. Performance, power, and cost restrictions continue to bring new obstacles in the design process. Effective project management, robust design tools, and thorough expertise of hardware and software domains are necessary to overcome these constraints.

IV. BASIC SCHEDULING TECHNIQUES

Scheduling is an important part of hardware-software codesign because it determines the order that which tasks are executed. The two basic methods for scheduling tasks are: ASAP (As Soon As Possible) and ALAP (As Late As Possible).

A. ASAP Scheduling

ASAP scheduling forces the start of each activity as early as possible. This technique reduces the start time of the tasks, and based on that, it generates a schedule under which the jobs execute ASAP after their dependencies are resolved. ASAP scheduling helps reduce total execution time but can also lead to poor resource utilization in a case when jobs are not well-balanced.

B. ALAP Scheduling

The objective of ALAP scheduling, on the other hand, is to start each task as late as possible without delaying the completion of a project. Such a strategy can maximize the slack time of tasks. This will balance out the use of resources and accommodate changes in the scheduling of tasks. However, ALAP scheduling may take longer to complete than ASAP scheduling.

C. Comparison of ASAP and ALAP Scheduling

The ASAP and ALAP scheduling techniques have their pros and cons. ASAP scheduling is simple and potent in project completion time reduction, and highly suitable for any time-critical applications. However, this philosophy of task execution may result in resource contention and inefficiency. On the other hand, ALAP scheduling provides more flexibility in resource utilization and may also increase project deadlines. It is important to understand that there are trade-offs between these techniques in order to apply the right technique to the right project.

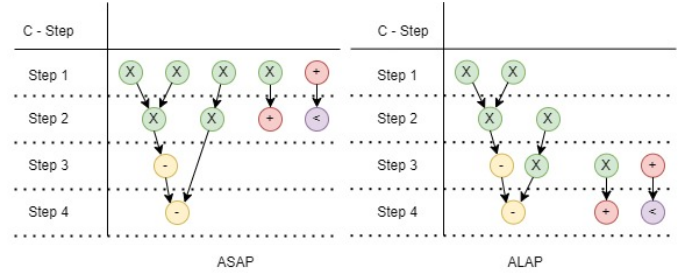


Fig. 1. ASAP and ALAP based on [7]

D. Advanced Scheduling Techniques

In addition to ASAP and ALAP, several advanced techniques have been developed. These techniques address the limitations of the above-mentioned basic methods. These techniques are:

- **List Scheduling:** This technique follows all the tasks in a preference list. Tasks are scheduled in order of preference setup. Preferences can be done on any basis like earliest start time or higher priority.
- **Graph-Based Scheduling:** This is a technique where tasks and their dependencies are modeled using DAGs (directed acyclic graphs); thereby, facilitating more sophisticated scheduling algorithms.
- **Constraint-Based Scheduling:** It is a method where the availability of resources and the timing of all critical requirements to be met are respected in the course of scheduling, thus ensuring the satisfaction of all constraints.

These advanced techniques bring flexibility and efficiency into the scheduling of such complex hardware-software codesign projects. [8]

V. HIGH-LEVEL SYNTHESIS RELATION TO FORCE DIRECTED SCHEDULING

High-Level Synthesis (HLS) transforms an algorithmic description of a design into a RTL (register-transfer level) implementation, as a means of automating the design process. Scheduling in HLS tends to be the step that dictates which order operations are to be executed and when resources are to be allocated. FDS optimizes these schedules, balancing computational loads to minimize delays.

A. The Role of High-Level Synthesis in Codesign

HLS is an important tool that is in use with hardware-software codesign because it completes the missing link between high-level design criteria and low-level hardware implementation. It automates the synthesis process for synthesis tools, which reduces the time spent in converting the description of algorithms into hardware designs. Automation of this kind could lead designers to search within a very large space of designs rapidly and hit optimal solutions, which satisfy the constraints of performance, power, and cost.

B. Integration of FDS with HLS

Coupling FDS with HLS will further enhance the scheduling capability of the already robust scheduling features in HLS tools to create efficient and load-balanced schedules. This is because FDS acts in response to the "forces" exerted by task dependencies and resource limits. Hence, tasks will be scheduled not to delay, but to use the resources available to them fully. Designs that fulfill such stringent performance and power standards will hence have an improved quality.

C. Benefits of Combining FDS and HLS

Several advantages exist for design flow in the combination of HLS with FDS, noted below: [3]

- 1) Improved Scheduling Efficiency: Resource usage in FDS is optimized to reduce the overall execution time.
- 2) Improved Design Quality: Therefore, by balancing forces and reducing delays, higher quality designs are reached that balance performance against power requirements.
- 3) Reduced Development Time: HLS automation, complemented by FDS's optimized scheduling, reduces the total development time and effort.

VI. HOOKE'S LAW

Hooke's Law, a physics principle, asserts that the force required to extend or compress a spring is proportionate to the distance extended or compressed. Hooke's law is represented by the equation $F=kx$. FDS employs the ideas of Hooke's Law metaphorically, viewing tasks as forces and the project timeframe as a spring.

A. Application of Hooke's Law in Scheduling

Apply Hooke's Law to scheduling by calculating the various "forces" assigned to each task based on its dependencies and available slack time. Tasks that require less time to complete can be relocated easily to balance the overall forces in the system.

B. Benefits of Using Hooke's Law in FDS

Hooke's Law, if applied to FDS, gives a coherent and intuitive way of thinking about balancing scheduling forces. The developers may get a better intuitive understanding of the trade-offs in scheduling decisions while working with tasks visualized as connected masses restrained by various springs. This also allows a precise evaluation of crucial tasks and potential bottlenecks, allowing for more targeted optimization efforts.

VII. FORCE DIRECTED SCHEDULING

Force Directed Scheduling is a technique for balancing scheduling forces given to tasks in response to resource contention and dependencies. It aims to mitigate these forces, resulting in more efficient and balanced schedules. The next section describes the approaches and stages involved in determining scheduling with FDS.

A. Methodology

The Force Directed Scheduling approach includes the following steps:

- 1) Create a directed acyclic graph (DAG) to represent tasks and their dependencies.
- 2) Compute the forces for each activity based on its dependencies and resource constraints.
- 3) Force balancing: The task start timings will be changed to minimize the system's total force level.
- 4) Schedule Generation: The process of creating the final schedule using balanced forces.

Each of these processes is in turn critical in getting the final schedule to be most optimal with respect to resource utilisation and execution time.

VIII. DETAILED STEPS FOR DETERMINING THE SCHEDULING

A. Step 1: Task Graph Representation

The first step in FDS is to represent the tasks and their dependencies in a directed acyclic graph (DAG). This graph would be beneficial for visualization purposes and in selecting the critical path containing bottlenecks. Nodes are tasks; edges are dependencies between them.

B. Step 2: Force Calculation

Step two involves calculating the forces acting on every task. These forces are defined by their interdependence and resource limitations. The earliest and latest start timings for each task (EST and LST) will be determined. The force on a task t is shown in the following equation. The EST of a task is the earliest time at which a task can start, conditioned on the completion times of its predecessor tasks. The LST denotes the latest time at which a task can start without slowing down the project. The difference between EST and LST gives the slack time available for a task. Tasks with more slack time can therefore easily be shifted to bring about a balance in the forces.

$$F(t) = \sum_{i \in \text{predecessors}(t)} w_{i,t} \cdot (EST_i - LST_i)$$

$w_{i,t}$ is the weight of the task i influencing task t , and EST_i and LST_i are the earliest and latest start times of task i , respectively.

This is an iterative process of changing task start times to rebalance forces in a way that minimizes the total force in the system, hence improving efficiency. Each iteration allows a task to be changed within its slack period, which is directed by estimated forces to decrease scheduling constraints. The process continues until the forces are balanced, which indicates that an ideal schedule has been reached. Force balancing requires complex calculations and adjustments. As a result, advanced algorithms and computing power is required.

C. Step 4: Schedule Generation

Once the forces are balanced, the output schedule is then generated. The schedule will then ensure that all tasks are executed in the best manner, not violate resource constraints and reduce execution time as much as possible. The final schedule assures an optimal order of task execution, that sustains resource usage while considering task dependencies. [6]

IX. ADVANTAGES OF FDS ALGORITHM

The following are some of the advantages of Force Directed Scheduling over other scheduling techniques.

A. Optimized Resource Utilization and Reduced Execution Time

FDS ensures effective resource usage by load balancing and minimizing idle time. Since it granularly distributes tasks over all available resources, the chances of bottlenecking and underutilized components are largely reduced. This is especially important for highly optimized systems where resource requirements are very stringent.

FDS reduces the total time required for the completion of a task by perfectly balancing the forces and re-scheduling the tasks. It leads to faster execution overall because the scheduling is done in such a way that it minimizes delays. This feature would be especially useful for time-critical applications where deadlines must be met.

B. Flexibility and Scalability

FDS is adaptable to many applications and systems with unique constraints, making it suited for diverse electronic systems. Whether the system is basic or complex, FDS can always be customized to meet individual scheduling requirements. This is a significant advantage in the broader field of hardware-software codesign. FDS can handle complex systems, including many tasks that have dependencies, therefore it will be especially useful for large designs. As the systems become more complex, the capability that FDS has in handling and optimizing schedules becomes very important. This makes FDS scalable, and its effectiveness is retained by increasing the scope and scale of the design.

C. Improved Performance

FDS improves the electronic system for better throughput and lower latency by balancing forces. Improved scheduling makes for smoother operation and higher overall system performance. This is particularly important in systems where performance is one of the key aspects to measure success. [6]

X. APPLICATIONS OF FORCE DIRECTED SCHEDULING

A. Embedded Systems

Considering embedded systems, FDS can enhance the performance to a large extent as the resources are usually limited and high efficiency is expected. Since the scheduling of tasks has been optimized by FDS, it ensures that the available limited resources are optimally utilized for the overall functioning of the system.

B. Real-Time Systems

Real-time systems are used in scenarios, where precise timing and predictable performance are required, real-time systems often benefit from the use of FDS. In such instances, the algorithm naturally prioritizes time-balanced tasks, reducing delays and completing time-constrained deadlines of crucial real-time requirements for the application.

C. Parallel Processing

FDS allows load balancing over the processors in parallel processing systems, where many tasks must be executed at once. With the proper distribution of tasks over the available processors the execution time will be reduced and the performance will increase significantly.

D. High-Performance Computing

High-performance computing systems can use FDS to allow full efficiency and throughput applications. This enhanced scheduling within FDS offers greater resource utilization and faster execution within high-performance environments. [5]

XI. CASE STUDIES

A. Case Study 1: Embedded System Design

FDS was used in the embedded system design project to optimize task scheduling in a resource-constrained situation. These results confirm that FDS is effective for use in embedded systems since it exhibited a significant increase in resource utilization and a reduction in overall execution time.

B. Case Study 2: Real-Time System Implementation

In a real-time system implementation project, applying FDS was necessary to guarantee the completion of all real-time constraints. Its balanced scheduling provided predictable reliability in performance, fulfilling all timing requirements that ensured an improved efficiency of the system.

C. Case Study 3: Parallel Processing Optimization

FDS was used to distribute the load among multiple CPUs. Improved scheduling resulted in a significant performance boost: jobs were completed more effectively, reducing overall execution time. [3]

XII. CHALLENGES AND LIMITATIONS

A. Complexity and Computational Overhead

One of the limitations of FDS lies in its inherent complexity, mainly for huge systems with many tasks and dependencies. It is evident that an iterative process of balancing forces and schedules needs efficient algorithms and normally massive computational power; this could pose problems in scenarios where computing resources are characterized by low intensity. Due to FDS's high computational costs, it can be too expensive for systems with many tasks. If FDS is utilized too frequently, it affects the system's efficiency.

B. Adaptability to Dynamic Changes

Because FDS is designed for static scheduling, it can be difficult to change system parameters during runtime. This static limitation of FDS can restrict the efficiency of such systems where tasks and dependencies change very frequently.

C. Resource Constraints

While FDS maximizes resource utilization, it may fail in situations where resource constraints are severe. This limits the algorithm's balancing forces and schedule optimization to only accessible resources, which may not always produce the desired outcomes. [8]

XIII. FUTURE DIRECTIONS

A. Integration with Machine Learning

Integration of FDS into a machine learning-based system is likely to enhance the former's effectiveness in handling the most complex scenarios of scheduling. In that respect, machine learning techniques could be used to make predictions about the scheduling decisions in real-time to enhance the performance and adaptability of the FDS.

B. Real-Time Adaptation

This benefits FDS by extending its applicability to real-time scheduling, hence methods to accommodate such needs should be developed. Real-time data-based dynamic scheduling techniques can improve the working of FDS in systems where tasks and dependencies do change but not very frequently.

C. Hybrid Scheduling Approaches

Hybrid approaches where merging FDS with other scheduling algorithms can provide a robust solution for complicated systems. Hybrid techniques can leverage the power of different scheduling algorithms to improve both performance and flexibility.

D. Improved Algorithms

Improving the algorithms for force calculation and balancing can greatly reduce the computing overhead of FDS. Further advances in this area may require the use of FDS on a substantially broader set of systems.

E. Quantum Computing Integration

Quantum algorithms potentially boost the performance of FDS as they solve complex scheduling problems more effectively than classical algorithms.

F. Enhanced Visualization Tools

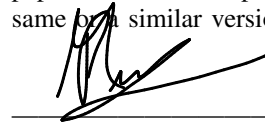
Proper visualization tools in FDS are required to help designers comprehend and understand scheduling decisions. The dependencies of tasks and forces are graphically and explicitly presented tools, which allows us to easily identify and enhance any scheduling bottlenecks. [3]

XIV. CONCLUSION

Force Directed Scheduling is a powerful technique in Hardware-Software Codesign that achieves the goal of optimizing resources while reducing execution time. Because FDS balances opposite forces on tasks, this strategy produces more effective scheduling results than other conventional techniques. This paper discusses the fundamentals of force-directed scheduling methodology, as well as its applications and benefits, focusing on why this technology is important in modern electronic design. As electronic systems improve and become more sophisticated, FDS plays an increasingly important role in maintaining their efficiency and success.

XV. AFFIDAVIT - DEUNDARA PALLIYA PISULA MAYAN GURUGE

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.



Deundara Palliya Pisula Mayan Guruge
Lippstadt, 03.07.2024

REFERENCES

- [1] Lee, E. T., Schrage, J. B. (1974). An algorithm for scheduling independent tasks. *Operations Research*, 22, 458-475.
- [2] Johnson, R. G. (1983). Heterogeneous parallel processing. *Communications of the ACM*, 26, 832-844.
- [3] Brandenburg, F. Force Directed Scheduling: A Survey. *Journal of Scheduling*, 15(1), 2012, pp. 3-24.
- [4] Liu, J. W. S. (2000). *Real-Time Systems*. Prentice Hall.
- [5] Kress, R., Thiele, L. (2002). *Models and Algorithms for Embedded Real-Time Systems*. Springer.
- [6] P. G. Pauline and J. P. Knight, "Force-Directed Scheduling in Automatic Data Path Synthesis," 24th ACM/IEEE Design Automation Conference, Miami Beach, FL, USA, 1987, pp.195-202, doi: 10.1145/37888.37918.
- [7] Scheduling Algorithms for High-Level Synthesis Zoltan Baruch Computer Science Department, Technical University of Cluj-Napoca , URL - <http://users.utcluj.ro/~baruch/papers/Scheduling-Algorithms.pdf>
- [8] Micheli, Giovanni. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Professional, 1994.