

FORCE DIRECTED SCHEDULING

Nirojan Navaratnarajah

Hardware Software Codesign
Summer Semester 2023
Department of Electronic Engineering
Hamm-Lippstadt University of Applied Sciences
nirojan.navaratnarajah@stud.hshl.de

1. ABSTRACT

Task scheduling is an important problem in the field of computer science and engineering. It involves allocating resources to tasks in an optimal manner to achieve desired objectives, such as minimizing the completion time or maximizing resource utilization. In this paper, we discuss about force-directed scheduling, that is based on the concept of force-directed graph drawing. This approach leverages the principles of physics to model task dependencies as springs and compute a final schedule that minimizes the total energy of the system, and this algorithm was proposed by Paulin and Knight [1]

2. INTRODUCTION

Task scheduling is a common problem in many areas, including computer systems, manufacturing, and project management. In computer systems, task scheduling refers to the allocation of processing resources, such as processors and memory, to tasks in a way that optimizes performance and resource utilization. In this context, the objective is to minimize the completion time of the tasks, subject to resource constraints and task dependencies. Force-directed scheduling is a novel approach to task scheduling that models task dependencies as springs and computes a final schedule by minimizing the total energy of the system. The approach is based on the concept of force-directed graph drawing, which is a technique used in graph visualization to arrange nodes in a way that minimizes the energy of the system.

3. HWSW CODESIGN

Hardware-software codesign is a design methodology that involves simultaneous development and optimization of both hardware and software components of a system. It aims to find the best partitioning and allocation of tasks between hardware and software, taking advantage of the strengths of each

domain to achieve improved performance, power efficiency, and system-level optimization.

Hardware-software codesign is a promising approach that integrates software and hardware components within embedded systems. By combining software blocks (such as MCUs or DSPs) with hardware blocks (like FPGAs or ASICs), it offers advantages over traditional software-centric architectures. This methodology allows for the partitioning of applications into hardware and software tasks, accelerating computationally intensive operations in hardware while efficiently handling less intensive tasks in software.[6].

DMA (Direct Memory Access) is a transmission method that enables data transfer between external devices and system memory or between different memory locations without direct CPU control. It establishes a direct data transmission path through hardware, bypassing CPU involvement during the transfer process. This approach enhances CPU efficiency by offloading data transfer tasks to dedicated hardware. DMA operates by allowing external devices, such as I/O devices, to access system memory directly, reducing the burden on the CPU and improving overall system performance. The collaboration between hardware (DMA controller) and software facilitates seamless and efficient data transfers, exemplifying the principles of hardware-software codesign.[10]

Force-directed scheduling can be used in the context of hardware-software codesign. It helps in the allocation and scheduling decisions by balancing the workload across different resources or functional units in the design.

In the context of hardware-software codesign, force-directed scheduling can be applied at different levels:

1. Task Allocation: Force-directed scheduling can help determine which tasks should be implemented in hardware and which should be executed in software. By modeling the design as a system of particles, the algorithm can assign tasks to appropriate domains based on factors like computation requirements, communication overhead, and resource availability.

2. Task Scheduling: Once the tasks are allocated to hardware and software domains, force-directed scheduling can be used to schedule the execution of these tasks. The algorithm considers factors such as task dependencies, resource constraints, and optimization objectives to determine an optimal or near-optimal scheduling solution.

By utilizing force-directed scheduling within hardware-software codesign, designers can achieve a balanced allocation and scheduling of tasks, leading to improved system performance, reduced communication overhead, and better resource utilization. The algorithm helps in finding an efficient partitioning of tasks between hardware and software, considering factors such as task characteristics, communication requirements, and system-level objectives.

Overall, force-directed scheduling is a scheduling technique that can be employed within the hardware-software codesign process to facilitate task allocation and scheduling decisions, leading to optimized system-level designs.

4. BASIC SCHEDULING TECHNIQUES

There are two main objectives for the scheduling problem, which involve a predefined set of functional units and the duration of a control step. The first objective is to minimize the number of functional units while keeping the number of control steps fixed. This type of scheduling is known as time-constrained scheduling. The second objective is to minimize the number of control steps while considering the limitations of the design cost. The design cost can be measured by factors such as the number of functional and storage units, the number of NAND gates, or the chip-layout area. This approach is referred to as resource-constrained scheduling.[5]

4.1. ASAP scheduling

ASAP (As Soon As Possible) and ALAP (As Late As Possible) scheduling are both time-constrained scheduling algorithms. ASAP scheduling aims to schedule operations as early as possible, meaning that operations are scheduled to start in the earliest available control steps without violating any data dependencies. It focuses on minimizing the critical path, which is the longest path of dependent operations in the design. The ASAP scheduling is shown above in Figure 1 (left side).

4.2. ALAP scheduling

ALAP scheduling, on the other hand, schedules operations as late as possible within the given constraints. It aims to minimize the number of control steps by scheduling operations towards the end of the available time frame. ALAP scheduling considers the critical path in reverse, starting from the final operation and scheduling operations backward.

Both ASAP and ALAP scheduling algorithms have the goal of optimizing the performance of the design by minimizing the number of control steps required while satisfying the data dependencies and constraints. Figure 1 (right side) represents diagram represents the ALAP scheduling.

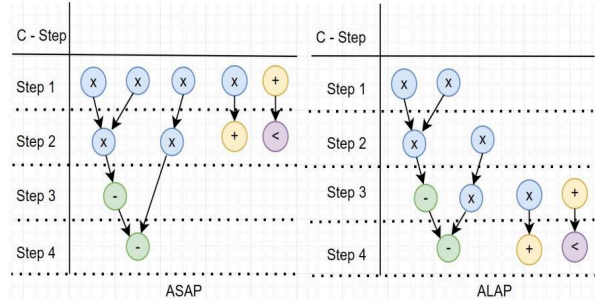


Fig. 1. ASAP and ALAP (based on [4]).

5. HIGH LEVEL SYNTHESIS AND ITS RELATION TO FORCE DIRECTED SCHEDULING

High-level synthesis (HLS) refers to the process of automatically transforming high-level behavioral descriptions (often written in high-level programming languages) into register-transfer level (RTL) descriptions, which can be implemented in hardware. HLS tools analyze the behavioral description and generate an RTL design that meets specified performance, power, and area constraints. The synthesis process involves tasks such as scheduling, allocation, binding, and optimization. Force-directed scheduling is a specific scheduling technique used within the HLS process which will be discussed in brief in the upcoming sections.

6. HOOKES LAW

Hooke's Law, first published in 1660, establishes a relationship between force and displacement in a mechanical spring. According to Hooke's Law, the force (F) exerted by a spring is proportional to its displacement (x) and is given by the equation $F = -kx$

where k represents the spring constant. Furthermore, the potential energy (PE) stored within the spring can be calculated using the equation

- $PE = kx$

Mechanical springs find extensive use in various everyday applications, such as suspension systems in beds and vehicles, where they provide mechanical support and absorb mechanical vibrations.[4]

Understanding the fundamentals of Hooke's Law is crucial for comprehending force directed scheduling, which is the focus of this paper. Force directed scheduling leverages the principles of Hooke's Law to guide the scheduling process as

shown in Figure 2. By considering tasks or activities as forces and the project timeline as a spring, force directed scheduling aims to optimize task sequencing and allocation based on the concept of forces and displacements. Throughout the paper, we will explore how Hooke's Law forms the foundation for force directed scheduling.

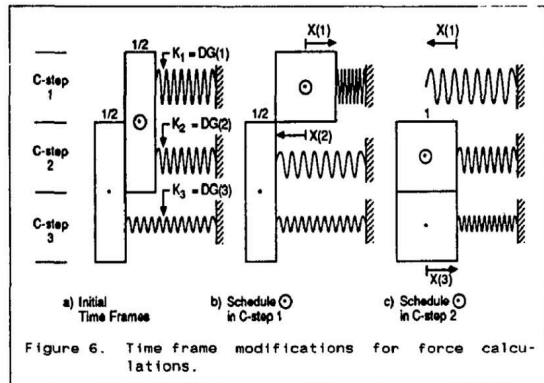


Fig. 2. Hookes Law.[4]

7. FORCE DIRECTED SCHEDULING

7.1. Methodology

The force-directed scheduling algorithm starts by representing the tasks and their dependencies as nodes and springs in a graph. Each task is represented as a node, and its dependencies are represented as springs connecting the nodes. The algorithm then computes a final schedule by minimizing the total energy of the system, subject to resource constraints and task dependencies. The objective is to decrease the number of processors needed by reducing the level of simultaneous execution of operations assigned to them, while maintaining the same overall execution duration.[4]

The energy of the system is computed as the sum of the potential energy of each spring, which represents the deviation from the desired distance between the nodes, and the kinetic energy of each node, which represents the velocity of the node. The algorithm iteratively updates the position of the nodes and the velocity of the springs until the total energy of the system converges to a minimum. The final schedule is then computed by ordering the nodes in the order in which they are processed by the algorithm. The processing order of the nodes represents the order in which the tasks are executed in the schedule. Time-constrained scheduling is crucial for real-time system applications, such as digital signal processing (DSP) systems. In DSP, the sampling rate of the input data stream sets a time limit for executing the DSP algorithm on each data sample before the arrival of the next sample. The objective is to minimize hardware costs since the sampling rate remains fixed. The sampling rate can be represented in terms of the number of control steps needed for executing the

DSP algorithm, given the control step length.[7]

There are three techniques commonly used in time-constrained scheduling algorithms: mathematical programming, constructive heuristics, and iterative refinement. This paper will focus on a specific example of a constructive heuristic approach called the force-directed scheduling method.

The initial step involves determining both an ASAP (as soon as possible) scheduling and an ALAP (as late as possible) scheduling. By combining the results of both schedules, we can establish the possible time frames for each operation. Figure 3 presents the derived DFG (data flow graph) from this example shown in Figure 1. The raw DFG does not include scheduled operations in terms of time (control steps). In this case, the ASAP scheduling is depicted, assuming that all operations require one clock cycle and that subsequent operations cannot be scheduled in the same cycle, for the sake of simplicity[4].

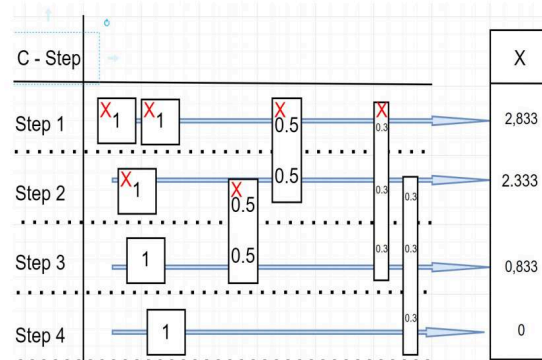


Fig. 3. Diagram Flow Graph.

The force-directed scheduling (FDS) heuristic is a widely recognized technique for scheduling tasks with specific timing constraints. This paper presents a simplified version of the FDS algorithm that focuses on minimizing the total number of functional units required for implementation. The algorithm achieves this objective by evenly distributing operations of the same type across available states. This uniform distribution ensures efficient utilization of functional units across all control steps, resulting in a high unit utilization rate. To determine the control step range for each operation, the FDS algorithm relies on both ASAP (As Soon As Possible) and ALAP (As Late As Possible) scheduling algorithms. It assumes that each operation has an equal probability of being scheduled in any control step within the determined range, while having zero probability of being scheduled in other control steps.[5]

7.2. Steps for determining the Scheduling

The overall process can be summarized as follows:

1. Determine time frames for operations.
2. Update the distribution graphs, considering any conditional dependencies.

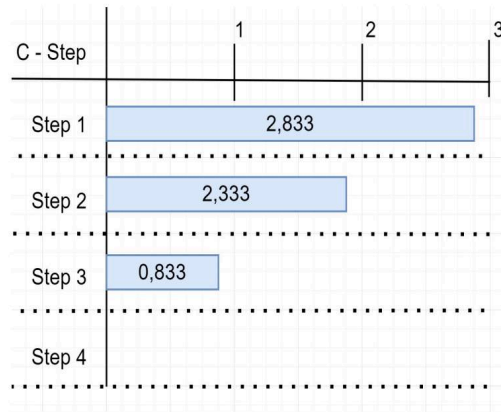


Fig. 4. Distribution graphs.

3. Calculate self-forces for each operation.
4. Calculate forces from pre/successor operations.
5. Schedule the operation and its corresponding c-step pair that yields the best outcome.

These steps are repeated until the time frame of each operation is reduced to one. The distribution graphs are updated at each iteration to reflect the current distribution of operations. The forces acting on unscheduled operations will change accordingly. The final results obtained after using the FDS algorithm for the multiplication operation is as shown in Figure 5.

7.2.1. An example calculation for force directed scheduling relating to the Hooke's Law

After having understood the ASAP and ALAP scheduling, now we know the maximum time step that is allowed in this case 4 C-steps. Each individual operation shown as small circles could either have mobility in terms of jumping to other c-steps or may be not at all depending on the dependency from the predecessor and the final time of execution, certain operations could be having a mobility of 0.5 meaning it can go from one c-step to next (up or down), or even move 2 C-steps as in our example, all without exceeding the time constraint or violating the operation dependency from predecessor.

In the example discussed below for the calculation, considering the scenarios of only the multiplication nodes as seen in Figure 1, Out of the six multiplication nodes three nodes have mobility which are either 0.5 or 0.333, which means the operation could either be scheduled in two different c-steps and three different c-steps respectively, and the rest three operations do not have mobility meaning they cannot be moved to another C-step. Figure 3 depicts this clearly. 'X' in this figure is the sum of the probability of the possible operations in each control steps, which is assumed as the force constant in Hooke's Law. The higher the value the more resource is required to complete the operation. Here comes the force directed scheduling into main concern. FDS helps distribute

these values evenly as to minimize the required resource and still stick to the time constraint.

We calculate the force for each mobile operation on each possible c-step as follows :

Lets calculate for the node with mobility in C step 1 :

- In C-Step 1 :
Force = $2.833*(1-0.5) + 2.333*(0-0.5) = +0.25$
- In C-Step 2 : $F = 2.833*(0-0.5) + 2.333*(1-0.5) = -0.25$

Now observing the values of the Net force exerted, an analogy can be applied which is, the more negative the results the better it is to be scheduled in this C-Step, this means this operation could be shifted to the C-step 2 for better results.

For the first node with mobility in C step 2 :

- In C-Step 2 : $2.333*(1-0.5) + 0.833*(0-0.5) = +0.75$
- In C-Step 3 : $2.333*(0-0.5) + 0.833*(1-0.5) = -0.75$

The FDS algorithm decides to choose to move the operation into the Control step with more negative value for the calculated force, thus into C-step 3.

For last node :

- In C-Step 1 :
 $F = 2.833*(1-0.33) + 2.333*(0-0.33) + 0.833*(0-0.33) = +0.853$
- In C- Step 2 :
 $F = 2.833*(0-0.33) + 2.333*(1-0.33) + 0.833*(0-0.33) = 0.351$

Thus FDS algorithm would consider to place this specific node to be executed in controlstep 3, as it has the least positive value.

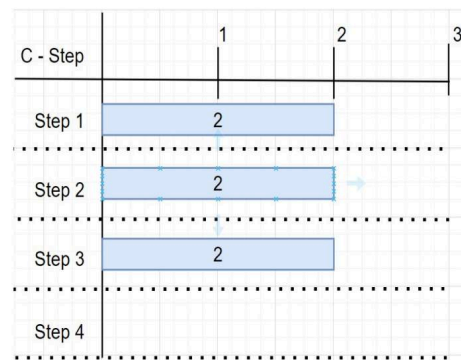


Fig. 5. Final distribution Graph after using FDS.

Figure 5 shows how FDS finalizes the scheduling by evenly distributing the tasks and minimizing the resource consumption from a maximum value of 2.833 down to 2.0. This simple example helps us understand the FDS algorithm.

By noting that the actual scheduling of a particular operation may be anywhere between its ASAP cycle and its ALAP cycle, one can draw a time frame diagram (Figure 3). Here

the width of the box containing a particular operation represents the probability that the operation will be eventually placed in a given time slot. A useful heuristic is to assume uniform probability of placing an operation in any feasible control step. The area of each operation is always one but it is 'stretched' along its time frame. Overall the application of the FDS algorithm in this example helps us minimize the resource consumption, at the same time also not affect the maximum time constraint by optimally scheduling the tasks.

8. ADVANTAGES OF FORCE DIRECTED SCHEDULING ALGORITHM.

Force-directed scheduling is an approach to scheduling tasks that uses principles of physics and optimization algorithms. It offers several advantages, including flexibility to accommodate changes and uncertainties, resource optimization to minimize idle time and improve resource utilization, conflict resolution to handle overlapping requirements, visual representation for better understanding and analysis, real-time adaptation to respond to new tasks or constraints, identification of parallel task execution opportunities to increase efficiency, scalability for handling large and complex projects. Overall, force-directed scheduling provides an efficient and adaptive method for scheduling tasks in dynamic environments. The force-directed scheduling algorithm has been implemented in various design systems, and it has demonstrated superior results compared to list scheduling. However, the algorithm's execution times tend to be lengthy when dealing with large graphs, which restricts its practical applicability for larger designs.[8]

9. RESULTS AND CONCLUSION.

The force-directed scheduling algorithm has been evaluated on several benchmark datasets and has been shown to produce schedules that are comparable to or better than the schedules produced by existing task scheduling algorithms. The algorithm has been shown to be effective in minimizing the completion time of the tasks, subject to resource constraints and task dependencies.

In this paper a novel approach to task scheduling, called force-directed scheduling, that is based on the concept of force-directed graph drawing was discussed. This approach models task dependencies as springs and computes a final schedule by minimizing the total energy of the system. The results of the evaluation show that the force-directed scheduling algorithm produces schedules that are comparable to or better than the schedules produced by existing task scheduling algorithms.

10. REFERENCES

[1] E. T. Lee and J. B. Schrage, "An algorithm for scheduling independent tasks," *Operations Research*, vol. 22, pp. 458-

475, 1974.

[2] R. G. Johnson, "Heterogeneous parallel processing," *Communications of the ACM*, vol. 26, pp. 832-844, 1983.

[3] F. Brandenburg, "Force Synthesis and optimization of digital circuits.

[4] 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.

[4] C. K. Lee, S. C. Tan, F. F. Wu, S. Y. R. Hui and B. Chaudhuri, "Use of Hooke's law for stabilizing future smart grid — The electric spring concept," 2013 IEEE Energy Conversion Congress and Exposition, Denver, CO, USA, 2013, pp. 5253-5257, doi: 10.1109/ECCE.2013.6647412.

[5] 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>

[6] M. Fons, F. Fons and E. Canto, "Hardware-Software Codesign of a Fingerprint Alignment Processor," 2007 14th International Conference on Mixed Design of Integrated Circuits and Systems, Ciechocinek, Poland, 2007, pp. 661-666, doi: 10.1109/MIXDES.2007.4286246.

[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>

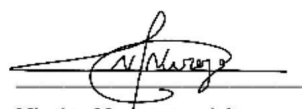
[8] De Micheli, G. Synthesis and Optimization of Digital Circuits.

[9] W. F. J. Verhaegh, E. H. L. Aarts, J. H. M. Korst and P. E. R. Lippens, "Improved force-directed scheduling," *Proceedings of the European Conference on Design Automation*, Amsterdam, Netherlands, 1991, pp. 430-435, doi: 10.1109/EDAC.1991.206441.

[10] C. Xia and G. Yang, "Implementation of DMA driver development under Linux," 2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA), Chongqing, China, 2021, pp. 721-724, doi: 10.1109/ICIBA52610.2021.9688147.

11. AFFIDAVIT - NIROJAN NAVARATNARAJAH

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.



Nirojan Navaratnarajah
Lippstadt, 05.06.2023