Parallel Programming on Embedded Multicore System ESP32

Universitatea Politehnica Timișoara

Marian Belean
Franz Joseph Pal

WS2019/2020
November 5, 2019

Abstract

The following documentation will focus on the principles of parallel programming in general and the mathematical background. In addition to the different parallel programming architectures, the various models for their implementation are also discussed. Moreover, in this thesis the prerequisites for mathematical calculation models, which are suitable for Parallel Programming, are elaborated.

For a practical example, the ESP32 microcontroller was chosen, an embedded multicore system. After a brief introduction to the hardware itself, further details of the project structure and the development of the application will be presented. Therefore, a short example will be explained to focus on the basics of parallel programming.

Finally, the aim of the project and the documentation is an automatic benchmark setup and a webfrontend result overview for visualization purposes, which will be discussed in more detail in the conclusion.

Declaration

I hereby certify that I have done the final thesis on my own, that I have completely and accurately stated all the aids I have used and identified everything individually, which was taken from the work of others unchanged or with modifications.

The topic of the submitted work was jointly with Mr. / Mrs. (...) (Bachelor and Master Thesis No. (...)).

Timișoara, the November 5, 2019

Signature:

Non-disclosure notice

This work contains confidential information. In spite of the anonymous presentation of the researched organisations, readers might conclude their identity. Therefore copying, quoting or publishing is not allowed without my explicit authorisation. Furthermore, disclosure of the information to anyone other than the examination board or lectors is not authorized.

Contents

Chapter 1

Introduction

Multicore systems are becoming increasingly popular as part of digitization and Industry 4.0[1] (German/EU) [2] [or 1] - also known as smart manufacoring in the USA [see 21, p1] - and are playing an important role in data processing and process automation [see 24, p294] [or 20, p1]. On the other hand, in addition to efficiency in energy consumption, performance in terms of computation time [see 24, p294] is required in almost every application field of multicore systems.

In fact, multicore hardware is not only exspecially for smart manufactoring. Nowerdays in almost every smart application like smart phones[2], wearables[3] or home automation we can find multicore embedded hardware platforms, which garantued high performance [see 4, p7], network connectivity, security and reliability [see 25, p5]. This field of application is also known as Internet of Things (IoT)[4].

Especially for embedded systems mathematical models as well as numerical solutions, which can be executed both simply and parallel, are suitable. The question arises to what extent parallel execution of different sub-tasks to calculate a problem [see 23, p4] increases the desired cost factor in terms of energy consumption [see 11, chapter 3] and computational efficiency [see 23, p4 Figure 3].

[1]add.: https://www.epicor.com/en-ae/resource-center/articles/what-is-industry-4-0/

[2]e.g. ARM based processors for mobile phones like https://www.arm.com/solutions/mobile-computing/smartphones

[3]add. information on ARM based solutions and the current trend in wearables: https://www.arm.com/solutions/wearables

[4]for additional information about Internet of Things, please see [15]

Chapter 2

Overview

2.1 Problem definition

Compared to single-core execution of tasks, multi-core embedded hardware platforms like the ESP32[1] provide the ability to develop advanced parallel computing software applications to reduce execution time and power consumption.

On the one hand, a major problem is choosing the right hardware platform to meet the cost and size factor, and moreover, whether a single-core or multi-core calculation is required. Therefore, context switching time, power consumption and total execution time must be included in the evaluation.

In order to develop an optimal solution, the hardware platform must be included in addition to the mathematical model of the problem itself. So in this case, suitable prerequisites and characteristics can be worked out in order to make an evaluation of "Parallel Computation Tasks on Embedded Multicore Systems" possible.

2.2 Objective of the documentation

The main goal of this documentation is to focus on the current parallel programming techniques, depending on the execution time in general and the required mathematical model. For this purpose, an application which can compute different sections of the Mandelbrot fractal [see 14, p11] will be developed to compare single core and multi-core calculations. Before the practical implementation, an investigation based on parallel architectures and programming models will be conducted.

The elaboration is diveded into three different chapters: In the first chapter, the results of the general research are presented [see Chapter 3]. After that, the second chapter is pointing out the practical implemenation of the developed application on the ESP32 [see Chapter 4]. In the end, the results including the webforntend and the automatic benchmark setup [see Chapter 5] will be discussed.

[1]add. information: https://www.espressif.com/en/products/hardware/socs

Chapter 3

Parallel Programming in General

Since the 1970s [23], the decade in which the microprocessor era started, the overall performance of a processor has increased [11]. This goal was achieved by several points, including "*sophisticated process technology, innovative architecture or micro-architecture*" [see 11, Chapter 1, p2]. In fact, increasing the clock speed of a single core processor, like Moore's Law predicted [23], was usually reached by increasing the number of transistors on the chip [23]. However, this go along side with the increase in complexity [see 23, Pollack's rule], which mean, that doubling the logic of a processor result in a performance boost of only 40% [see 23, Chapter 2].

Another huge problem chip manufacturers have to deal with is leakage power [see 11, Chapter 2, p3], because the "*transistor leakage current increases as the chip size shrinks*" [see 23, p2] [see Chart 3.1]. An increase of leakage current of the transistors also result in a increase of the die's temperature [11] along side the total power consumption as well.

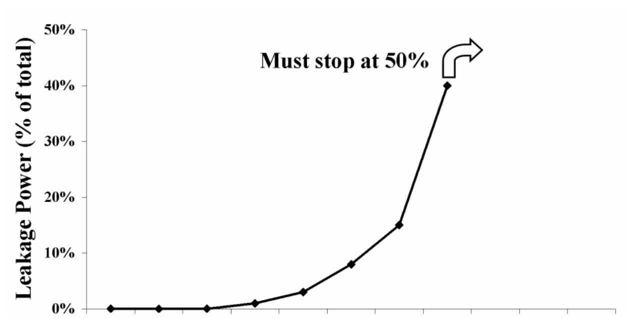

Figure 3.1: Leakage Power (% of total) vs. process technology [11]

Furthermore, a increase of the processor clock frequency to speed up the performance is only available to a suffisticating limit of 4GHz [23]. After this frequency threshold, also known as reaching the power wall, the "*power dissipation*" [see 23, p2] increases again.

Facing these types of problems such as "*chip fabrication costs, fault tolerance, power efficiency, heat dissipation*" [see 23, p3] along side with increasing processor performance, the only possible solution chip manufacturers and companies could offer was parallelism.

3.1 Basic Concept

Parallelism for processing is not something new. But due to the fact that real thread level parallelism [see Chapter 3.3.2] was only available after dual or multi-core processors were invented in 2005 [12], the topic itself and efficient software implementations are still treated in scientific work [3] [19].

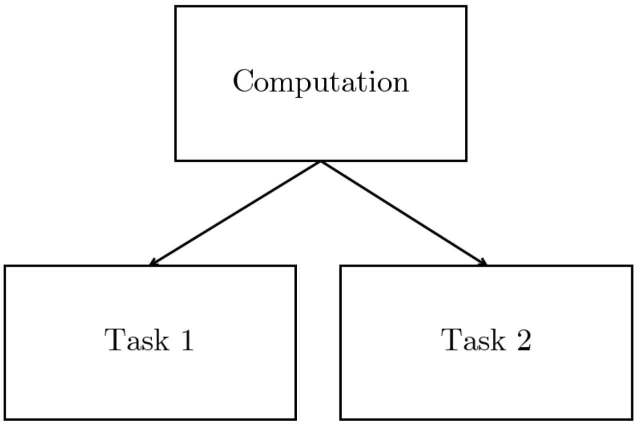

Figure 3.2: The basic concept of a simple concurrency computation.

In general, parallelism for programming means to split up a task or a computation into several sub tasks [e.g. Figure 3.2] or results, to decrease the execution time. Depending on the problem itself, these separated tasks can be independent or connected [see Chapter 3.1.4]. If we want to talk about the general concept of parallelism, we have to take a closer look to some mathematical laws, which try to describe the availability to parallel task execution and their limits.

3.1.1 Amdahl's Law

The first one, which quantifies parallelism, is called Amdahl's Law [6]. During the publication period of Amdahl's paper [5], critics claimed "*that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers*" [see 6, p80]. Of course this can be transferred on single and multi-core processors or even on multi threading [see 16, Chapter 1.3, p2], but in fact, like Amdahl claimed too, addressing hardware [6], and nowadays switching context time was not considered in this case.

Amdahl's Law wants "*to provide an upper limit on speedup*" [see 6, p81] in general to point out that there is a overhead [6], which can not pe implemented in parallel, but at the same time, "*apart from the sequential fraction, the remaining computations are perfectly parallelizable*" [see 6, p81].

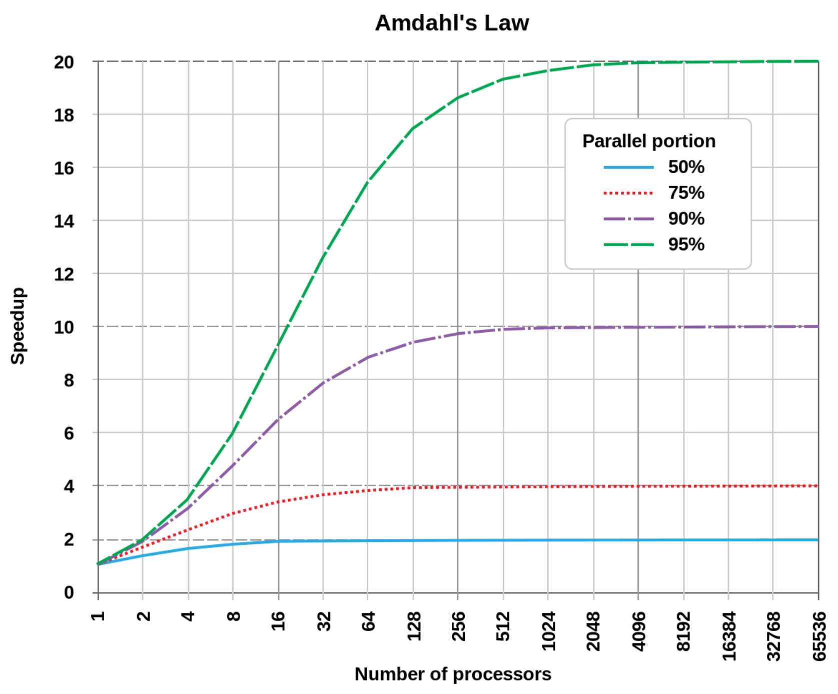

Figure 3.3: The limited speed-up of a program, which can be parallelized, depending on the number of parallel executions [10] [similar to 16, p4].

So regarding to Amdahl's Law, there has to be an upper limit of parallelism, due to the fact, that some sequential fractions still exists. In order to be able to describe this relationship, the time required to perform a calculation at once is related to the total time in parallel execution:

"*Let t_1 be the time taken by one processor solving a computational problem and t_p be the time taken by p processors solving the same problem. Finally let us denote the supposed inherently sequential fraction of instructions by f. Then, according to Amdahl, $t_p = t_1(f+(1-f)/p)$ and the speedup obtainable by p processors can be expressed as*" [see 6, p81]:

$$s = \frac{t_1}{t_p} = \frac{1}{f + (1-f)/p} \tag{3.1}$$

For example, a program which contains 90% of parallelizable code [see Figure 3.3] reaches his speed up limit at around 512 cores [formula 3.2]; in this case we substitute f= 0.2/2. After that number of processor cores, an significant speed up increase is not noticeable anymore .

$$\lim_{p\to\infty f\to 0.1}\left(\frac{t_1}{t_p}\right) = \lim_{p\to\infty}\left(\frac{1}{0.1 + (1-0.1)/p}\right) = 10 \tag{3.2}$$

Many other authors tried to claim that this upper limit of speed up, both in theory and practice, is not the final end. To proof that, only to mention a few , they took into account the "*energy per instruction (EPI)*" [Annavaram et al. in 6, Chapter 3, p81], a case study depending on "*asymmetric (or heterogeneous) chip multiprocessor architectures*" [Kumar et al. in 6, Chapter 3, p81] or even considering "*disk arrays to reduce input-output requirements*" [Patterson et al. in 6, Chapter 3, p81].

3.1.2 Gustafson's Law's

Due to the fact that Gustafson's Law's is based on "the same concepts as the bases of Amdahl's law, it is more a variant, rather than a refutation" [see 6, p81]. But in fact, it is another mathematical consideration, which offers, in comparison to Amdahl's Law, no upper speed up limit regarding parallelism. Related to Gustafson, the time a single core processor needs solving the same computational problem on the sequential would be ft_p, and on the parallelizable part *(1-f)pt*$_p$. Therefore, the total amount of achievable speed up by p processors can thus be calculated

$$s = \frac{t_1}{t_p} = \frac{ft_p + (1-f)pt_p}{t_p} = f + (1-f)p \tag{3.3}$$

using the formula 3.3 above. In this case, "f is the same "inherently sequential" fraction of instructions as in the case of Amdahl's law" [see 6, p81]. In addition to that, he doesn't take the "sequential input-output requirements proportional to input and output sizes into account" [see 6, p81].

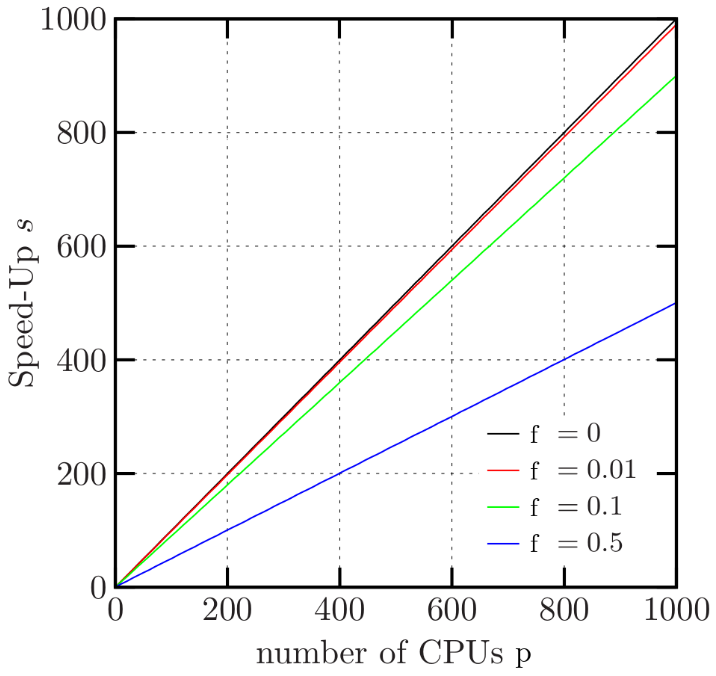

Figure 3.4: Gustafson's Law. In contrast to Amdahl we now have no upper limit to the speed up. f only determines the slope of the speed-up [13].

Regarding to [6], neither Amdahl's or Gustafson's Law are suitable to quantify parallelism in theory and practice, because both don't take into account, that "sequential fractions of computations have negligible effect on speedup if the growth rate of the parallelizable fraction is higher than that of the sequential fraction." [see 6, Chapter 7, p88].

Furthermore, [6] point out, that no simple formula governing parallelism exists. Both laws are more of an attempt to describe experimental results, and therefore understood rather as a draft rule of thumb as a law.

3.1.3 Concurrency in general

- ...[see 9, p3]
- ...[see 7, p3]
- ...

3.1.4 Principles of Parallel Computing

Emphasises design: Amdahl's law highlights the pitfalls of looking for sticking-plaster speed-ups in serial programs – design for concurrency [see 17, p4]

Aim of concurrency and their effects on program structure or implementation [see 17, p5]:

- Flexibility: Environments will be more heterogeneous.
- Efficiency: parallel for a speed-up purposes, more pitfalls (memory latency, thread overheads etc.)
- Simplicity: Parallel codes will be more complicated. All the more reason to strive for maintainable, understandable programs.

...[see 17, p11 ff.]

3.2 Definition of parallel mathematical computations

Mathematical examples [6]:

- ...[see 9, p8]
- ...[see 7, p4]
- ...[see 18, p398]

3.3 Parallel Computer Architecture

The terminology of computer architecture was invented in 1960s by the designers of IBM System to describe the structure of a computer. Computer architect's task is to write an suitable program code for the machine,keeping in mind everytime this structure of computer, understanding all the factors like state-of-the-art technologies at each design level and changing those designs tradeoffs for their specific applications. Parallel computing means the situation where tasks are separated into discrete parts that can be executed concurrently. Each part is diffused into a larger series of instructions which will be executed simultaneously on different CPUs. These kind of parallel systems have to deal with the simultaneous use of multiple computer resources that can can include either a single computer with multiple CPUs, ori a number of computers connected by a network creating a parallel processing cluster or combination of both. The crux of parallel processing are CPUs. Based on the number of instruction and data streams that can be processed simultaneously, computing systems are classified into four major categories based on Flynn's Taxonomy.
...[see 22, p9]
...examples of parallelism [see 22, p11]
...links 1)
...[see 8, p9 ff.]

3.3.1 Flynn's Taxonomy of Parallel Architectures

Flynn's classification was first elaborated and proposed by Michael Flynn in 1966 and represents a scheme which is based on the notion of information stream. The term 'stream' defines a sequence or flow of either one of both existent types of information which flows and are operated into a processor: instructions or data. Instruction stream defines the sequence of instructions performed by CPU, as in the same time the data stream defines the data traffic exchanged between the memory and CPU.His taxonomy left aside the machine's structure for classification of parallel computers and took over a whole new concept focusing on multiplicity of instructions and data streams observed by the CPU during the execution. (IDK HOW TO MAKE A LIST)The major four categories are the followings: SISD (single-instruction,single-data) systems It designs an sequential computer which exploits no parallelism in either the instruction stream nor data stream. An SISD computing system is a uniprocessor machine capable of single stream executions. SIMD(single-instruction, multiple-data) systems It designs a multiprocessor machine capable of executing a single instruction stream on multiple different data streams. Instructions can be executed sequentially, such as by pipelining, or in parallel by multiple functional units. MISD (multiple instruction streams, single data stream) systems It designs a multiprocessor machine capable of executing different instructions streams on the same data stream. MIMD (multiple instruction streams, multiple data streams) It designs a multiprocessor machine capable of executing multiple instructions streams on multiple data streams. This architectures include multi-core superscalar processors and distributed systems.
...[see 9, p5]
...[see 22, p13]
...[see 8, p4]

...[see 5, p2]
...[see 7, p15-p26]

3.3.2 Thread Level Parallelism

...[see 22, p24]
...[see 17, p14]

3.4 Parallel Programming Models

Steps to evaluate a proper parallel design [see 17, p6]:

1. Finding Concurrency
2. Algorithm Structure
3. Supporting Structures
4. Implementation Mechanisms

3.4.1 Classification of Parallel Programming Models

3.4.1.1 Process Interaction

...[see 9, p4]

3.4.1.2 Problem decomposition

...[see 22, p105 ff.]

Chapter 4

Project documentation

4.1 Concept development

...

4.2 Project structure

...

4.3 Simple mathematical computation examples for Parallel Programming

...

4.4 Class diagramm

...

4.4.1 C++ Backend benchmark

...

4.4.2 Vuejs Frontend

...

4.5 Benchmark setup

...

Chapter 5

Conclusion

...

Appendix A

Additional documents

Bibliography

[1] Plattform Industrie 4.0. "Positionspapier, Leitbild 2030 für Industrie 4.0 - Digitale Ökosysteme global gestalten". In: *Plattform Industrie 4.0* (Apr. 2019). URL: `https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Positionspapier%20Leitbild.pdf?__blob=publicationFile&v=5`.

[2] Plattform Industrie 4.0. *Was is Industrie 4.0?* 2019. URL: `https://www.plattform-i40.de/PI40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html`.

[3] Umut A. Acar and Guy E. Blelloch. *Algorithms: Parallel and Sequential.* Pittsburgh, USA: Carnegie Mellon University, Department of Computer Science, Feb. 2019.

[4] Tosiron Adegbija et al. "Microprocessor Optimizations for the Internet of Things: A Survey". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP (June 2017), pp. 1–1. DOI: `http://dx.doi.org/10.1109/TCAD.2017.2717782`.

[5] G.M. Amdahl. *Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of Spring Joint Computer Conference.* New York: ACM, 1967, pp. 483–485.

[6] Frank Devai. "The Refutation of Amdahl's Law and Its Variants". In: Sept. 2018, pp. 79–96. ISBN: 978-3-662-58038-7. DOI: `http://dx.doi.org/10.1007/978-3-662-58039-4_5`.

[7] Gabriel Edgar. *An Introduction to Parallel Computing.* URL: `http://www2.cs.uh.edu/~gabriel/courses/mpicourse_03_06/Introduction.pdf`.

[8] David Emerson. *Introduction to Parallel Computing Using Advanced Architectures and Algorithms.* Sept. 1997, pp. 1–40. URL: `https://www.researchgate.net/publication/321151707`.

[9] Prof. Robert van Engelen. *Parallel Programming Models.* URL: `https://www.cs.fsu.edu/~engelen/courses/HPC/Models.pdf`.

[10] Daniels220 at English Wikipedia. *SVG Graph illustrating Amdahl's law.* [Online; accessed October 29, 2019]. Apr. 2008. URL: `https://commons.wikimedia.org/w/index.php?curid=6678551`.

[11] Pawel Gepner and Michal Kowalik. "Multi-Core Processors: New Way to Achieve High System Performance." In: Jan. 2006, pp. 9–13. DOI: `http://dx.doi.org/10.1109/PARELEC.2006.54`.

[12] Computer Hope. *Computer processor history.* 2019. URL: `https://www.computerhope.com/history/processor.htm`.

[13] Harald Koestler, C. Moeller, and F Deserno. “Performance Results for Optical Flow on an Opteron Cluster Using a Parallel 2D/3D Multigrid Solver”. In: (Jan. 2006). URL: https://www.researchgate.net/publication/236892123.

[14] Nigel Lesmoir-Gordon. “THE MANDELBROT SET, FRACTAL GEOMETRY AND BENOIT MANDELBROT - The Life and Work of a Maverick Mathematician”. In: *Medicographia* 34 (June 2012), p. 353. URL: https://www.researchgate.net/publication/270285889.

[15] Sheik Dawood M. “Review on Applications of Internet of Things (IoT)”. In: (Dec. 2018). URL: https://www.researchgate.net/publication/329672903.

[16] Khaled Mashfiq. *NONLINEAR EARTHQUAKE ENGINEERING SIMULATION USING PARALLEL COMPUTING SYSTEM*. Dec. 2012. DOI: http://dx.doi.org/10.13140/RG.2.2.21215.87208.

[17] Tim Mattson, Beverly Sanders, and Berna Massingill. “Patterns for Parallel Programming”. In: (Sept. 2004).

[18] Zhang N. “A Novel Parallel Scan for Multicore Processors and Its Application in Sparse Matrix-Vector Multiplication”. In: *IEEE Transactions on Parallel and Distributed Systems* 23.3 (Mar. 2012), pp. 397–404. DOI: http://dx.doi.org/10.1109/TPDS.2011.174.

[19] Kota Sujatha et al. “Multicore Parallel Processing Concepts for Effective Sorting and Searching”. In: *IEEE* (2015). DOI: http://dx.doi.org/10.1109/SPACES.2015.7058238.

[20] Karlsruhe Institute of Technology. “Multi-core processors for mobility and industry 4.0”. In: *PHYSORG* (2016). URL: https://phys.org/news/2016-12-multi-core-processors-mobility-industry.html.

[21] Klaus-Dieter Thoben, Stefan Wiesner, and Thorsten Wuest. “”Industrie 4.0” and Smart Manufacturing – A Review of Research Issues and Application Examples”. In: *International Journal of Automation Technology* 11 (Jan. 2017), pp. 4–19. DOI: http://dx.doi.org/10.20965/ijat.2017.p0004.

[22] Gudula Rünger Thomas Rauber. *Parallel Programming for Multicore and Cluster Systems*. Germany: Second Edition, Springer Verlag, 2013.

[23] Balaji Venu. “Multi-core processors - An overview”. In: (Oct. 2011). URL: https://www.researchgate.net/publication/51945986.

[24] Dragan Vuksanović, Jelena Vešić, and Davor Korčok. “Industry 4.0: the Future Concepts and New Visions of Factory of the Future Development”. In: Jan. 2016, pp. 293–298. DOI: http://dx.doi.org/10.15308/Sinteza-2016-293-298.

[25] Yousaf Zikria et al. “Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution”. In: *Sensors* 8 (Apr. 2019), pp. 1–10. DOI: http://dx.doi.org/10.3390/s19081793.