POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

Evaluating FPGA-Based Convex Optimization Methods for Onboard Low-Thrust Trajectory Guidance

LAUREA MAGISTRALE IN SPACE ENGINEERING - INGEGNERIA SPAZIALE

Author: GONÇALO OLIVEIRA PINHO

Advisor: DR. ALESSANDRO MORSELLI

Co-advisors: DAVIDE PERICO, GIANFRANCO DI DOMENICO

Academic year: 2022-2023

1. Introduction

In recent years, the focus of the space industry has shifted towards miniaturizing satellites and their components. One valuable objective for future missions is the enhancement of autonomy and delegation of flight-related tasks, such as onboard guidance design, to automated systems [1]. However, the disparity in resources between OnBoard Computer (OBC) systems and ground computers has prompted research studies proposing a combination of software (SW) and hardware (HW) accelerators through Field Programmable Gate Arrays (FPGA) implementations. This compact, low-power consumption technology can integrate various functions and meet unique application requirements post-deployment, a valuable asset for missions that require in-flight adaptability. Furthermore, it provides high computational efficiency without compromising accuracy [2]. Nonetheless, the integration of the technology presents non-trivial challenges. Therefore, this thesis investigates the considerations and concerns of deploying onboard guidance algorithms that utilize convex optimization methods on reconfigurable computing hardware. The case problem at hand, solved with a Sequential Convex Programming (SCP) algorithm, is exhibited in Section 1.1. Within it, a series of sub-problems are resolved using the Embedded Conic Solver (ECOS), an interior point solver [3].

This work delves into the functionalities and architecture of Zynq devices [2], as well as the software tools used for hardware design. The defiance of utilizing hardware optimization frameworks for implementing software-oriented applications in FPGAs is examined. Afterward, an analysis and comparison of the numerical and computational results between the original algorithm and various hardware designs are conducted. The impact of fixed-point precision in similar problems is also presented. Lastly, a set of remarks is provided to aid the development of future onboard guidance applications that utilize Interior Point Methods (IPM).

1.1. Problem Formulation

An Earth to Mars minimum-fuel Space Trajectory Optimization (STO) problem was considered [4]. Assuming two-body dynamics, the resulting equations of motion of the problem were highly nonlinear. In addition, the state and control variables were coupled. These aspects posed the problem as nonlinear and nonconvex

[4]. Thus, the problem was transformed into a convex one through a series of processes that include changing of variables, relaxing control constraints, and linearizing the dynamics [1, 4]. After implementing these, a Convex Problem (CXP) was attained, followed by a SCP approach. This is a local optimization method where a sequence of convex optimal control sub-problems, defined by the equations of the CXP, is formed using the solutions from the previous iteration. Each sub-problem can be discretized into a Second-Order Cone Programming (SOCP) problem, solved employing the interior point solver ECOS [3, 4]. The SCP was the developed algorithm to be deployed in the hardware.

1.2. FPGA Overview

FPGA are integrated circuits intended for custom hardware implementation, being reconfigurable for an infinite number of times [2]. Further detailed in Section 1.2.1, each device presents a set of resources. The functional blocks of logic included in the hardware and designed by the user to meet specific functionalities are called *Intellectual Property* (IP) blocks or cores.

1.2.1 Zynq Architecture

A System-on-Chip (SoC) is a single chip capable of implementing multiple system functionalities. However, the upgrade or replacement of a component often requires the replacement of the entire chip. Therefore, Xilinx created the Zynq device, an All-Programmable SoC (APSoC) that combines a dual-core ARM Cortex-A9 processor with FPGA fabric [2]. Illustrated in Figure 1, its architecture can be broken down into:

- Processing System (PS): formed around the aforementioned processor, it is responsible for running software routines or operating systems;
- Programmable Logic (PL): the FPGA side, responsible for implementing high-speed logic, arithmetic, and data flow subsystems.

The primary link between these two components is established through Advance eXtensilbe Interface (AXI) connections. AXI is a protocol targeted at high-speed and high-frequency system designs, being fitted for FPGA applications.

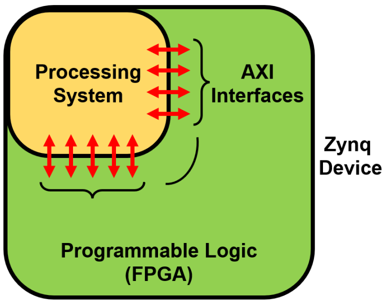

Figure 1: Simple scheme of the Zynq architecture.

Currently, three types of interfaces exist [2]:

- AXI4: high-performance interface suited for memory-mapped links;
- AXI4-lite: lightweight variant interface, used in simple low throughput memory transactions. When used, it creates a set of C driver files with functions and structures useful for controlling actions of the IP core (such as `start`, `pause`, `resume`, . . .);
- AXI4-Stream: used for high-speed streaming data scenarios.

The selected device for this work was the Xilinx Zynq ZC7Z020-1CLG484 APSoC, also known as Zedboard. Positioned as a development kit, it suited the scope of measuring the feasibility of deploying algorithms such as those explored in this work. The resources on the PL side of the board are 13300 Logic Slices, 220 DSP48E1s, and 140 Block Random Access Memory (BRAM), each of 36 Kb. Logic slices are fundamental elements for implementing various logic functions. DSP48E1s are mainly used for math-intensive operations while BRAMs refer to memory storage. The software tools used extensively throughout this work are Vivado, Vitis Integrated Design Environment (IDE) and Vitis High-Level Synthesis (HLS).

1.2.2 Optimization Techniques

Among the different approaches that could improve the computational performance of the algorithm, pipelining and dataflow have been considered. While pipelining explores an increase in the hardware throughput through software operations overlap, dataflow, and particularly the unroll technique, parallelizes functions or loops by physically increasing the number of dedicated resources.

1.2.3 Fixed-point Arithmetic

FPGA architecture revolves around fixed-point representation, therefore presenting more efficient arithmetic performances once compared to floating-point. By making a wise selection of bits for fixed cases that align with the application's required ranges, higher-quality hardware implementations are potentially achieved with fewer resources and improved performance. Section 2.3 presents the fixed-point procedure used.

2. Implementation

The minimum-fuel STO problem between Earth and Mars discussed in Section 1.1 was solved considering 100-nodes with a set of initial and final state conditions, and control constraints. First, the SCP algorithm was developed by integrating the ECOS framework. It was then compiled and executed on a desktop computer. Afterward, it was deployed in the PS of the Zedboard. No discernible difference between the two solutions was noted as Section 3.1 points out.

2.1. IP Design

2.1.1 Algorithm sub-routines

A profiling sequence was conducted to identify the execution-time bottlenecks of the SCP application. Therefore, the functions that have a higher impact on the application's execution time were prioritized for IP core development, resulting in `kkt_factor` and `sparseMV`. On one hand, `kkt_factor` computes the numeric factorization $\mathbf{A} = \mathbf{LDL^T}$, where $\mathbf{A}$ is a symmetric sparse matrix. With this process, the solution of linear systems $\mathbf{Ax} = \mathbf{b}$ can be computed at reduced computational cost thanks to a set of equations involving the permuted and lower triangular matrices $\mathbf{P}$ and $\mathbf{L}$. On the other hand, `sparseMV` performs sparse matrix-vector multiplication operations. Both are extensively used in the iterative solver.

2.1.2 Interfaces

After selecting and comprehending the operations of the functions to be implemented as IP cores, a decision is made on which interfaces to use. The IP core's interfaces involve two components: control signals, which encompass standard but crucial IP core control commands, and data signals, which are the data exchanged between the PS and PL for the IP core operation. For both IP cores and taking into account Section 1.2.1, the AXI4-lite interface fulfilled the requirements of control signals. For data signals, the AXI4-Stream was chosen due to its simpler configuration, requiring only the introduction of the AXI Direct Memory Access (DMA) IP core in Vivado. This block was responsible for correctly routing data between the PS and PL memories.
Moreover, the inputs and outputs of each function contained pointers to pointers, a coding practice not supported by Vitis HLS. Consequently, for the data exchange illustrated in Figure 2, Transmitters (TXs) and Receivers (RXs) arrays were instead created in the PS. However, as each AXI4-Stream interface must have an associated data type, the need to create distinct stream interfaces arose as the selected functions use integer and double data types. Therefore, for each AXI4-Stream interface of a particular data type, a respective TX (input) or RX (output) was created. Essentially, for the workflow, the TXs are loaded with all the data from the algorithm's variables to pass to the PL. Once the FPGA computation is finished, the RXs receive the data, which is then passed back to the algorithm's variables.

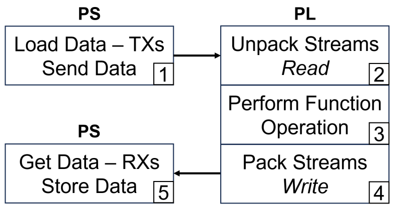

Figure 2: Scheme of data exchange between PS and PL.

2.1.3 Optimization

Referring to Figure 2, each IP core followed the same high-level operational flow (steps 2 to 4). Pipelining was used to reduce the cycle count to a minimum in steps 2 and 4. However, the primary optimization improvement stems from step 3, which depends on the function's operation and structure. While parallelization was explored during this step, it was ultimately deemed impractical for the reasons described ahead. Both functions revolve around multiplication loops which, typical of software-oriented applications, employ compact matrix storage methods, such as Compressed Column Storage (CCS). CCS involves storing a matrix $\mathbf{A}$ in 3 distinct arrays. Specifically, $\mathbf{A_x}$ stores the non-zero numerical values of $\mathbf{A}$, $\mathbf{A_i}$ stores the row indices of each entry, and $\mathbf{A_p}$ stores the index of elements in $\mathbf{A_x}$ which start a column of $\mathbf{A}$. Using this approach, the number of iterations in each multiplication loop depends on the number of elements in $\mathbf{A_x}$. The loops' bounds change based on the data utilized, resulting in a non-deterministic behavior. By contrast, FPGAs require deterministic loops to establish hardware instances of appropriate bounds. Otherwise, an arbitrary hardware instance is created, leading to inefficient code sequences as the data structure varies. Furthermore, parallel execution or pipelining becomes infeasible in cases where loop iterations have data dependencies of previous iterations. Both functions under analysis presented nondeterministic, data-dependent and strictly sequential behaviors. Therefore, the pipeline or concurrency of function operations was not possible, leading to worse hardware implementations when compared to the original algorithm as Section 3 explores.

2.2. Problem Reduction

From Figure 2, the retrieved data from the streams gets stored into local variables on the PL using BRAM resources. However, during the development of `kkt_factor`, it became noticeable that accommodating all necessary variables would necessitate a 457% increase in the available BRAM resources. Due to the high number of nodes considered, the problem presented a vast workspace of variables. For this thesis, the solution adopted was to reduce the size of the problem by considering fewer nodes, still pairing with the intent of learning insights when deploying STO algorithms in FPGAs. It has been determined that a problem with 5 nodes would be compatible with the resources of the Zedboard. However, reducing the number of nodes of the Earth to Mars problem from 100 to 5 would lead to unrealistic solutions. Therefore, a section of 5-nodes from the trajectory solution of the original Earth to Mars 100-node implementation was chosen as the new problem to solve. Accordingly, the conditions and constraints of the new problem were adjusted in the discussed SCP algorithm of Section 1.1.

2.3. Arbitrary Precision

It has been decided to implement the `sparseMV` function using both floating-point and fixed-point precision to investigate resource and performance differences. This function was chosen over `kkt_factor` due to its simpler routines. One iteration of the function under analysis was traced to calculate the required precision. From it, two implementations resulted: one providing coverage in precision up to 10^{-11} decimal places and other up to 10^{-16} decimal places. Additionally, the use of fixed-point precision was shown to impact the algorithm's convergence. Section 3.2 highlights the effects of such implementation.

3. Results

This section compares the results obtained from the algorithm deployed in the PS and different hardware designs. As a notation, "SW version" stands for the original algorithm only using software operations, whereas "HW version" denotes a combination of SW and HW capabilities by incorporating one of the developed IP cores into ECOS. Seen from Sections 3.2 and 3.3, separate designs for each IP core were created since the FPGA implementations yield lower performance efficiency when compared to the purely software-based application. Otherwise, their combination into a single design would result in an even less efficient outcome. Lastly, the designs were clocked at 666.67 MHz for the Central Processing Unit (CPU) and 100 MHz for the PL, this last one being coherent with the frequency of the IP cores.

3.1. Problem Solution

Table 1 presents the results of the 5-node problem and characteristics of the designs considered in this work from a mean of 1000 test runs. The 4th column represents the number of SCP iterations required to solve the problem while the Acceleration Factor (AF), defined in equation (1), is a metric to compare the performance of different implementations. The measurements of the number of cycles were possible thanks to the inclusion of the AXI Timer IP core.

$$AF = \frac{\#\ \text{Clock Cycles for SW version}}{\#\ \text{Clock Cycles for HW version}}. \qquad (1)$$

Design Version	Objective Function (-)	Fuel Mass (kg)	Iterations k (-)	AF (-)
SW (reference)	0.1778736542714191	15.064871	2	-
HW `kkt_factor` - floating-point	0.1778736542714191	15.064871	2	0.357
HW `sparseMV` - floating-point	0.1778736542714191	15.064871	2	0.721
HW `sparseMV` - fixed-point (10^{-11} precision coverage)	0.1778841011968801	15.064836	3	0.526
HW `sparseMV` - fixed-point (10^{-16} precision coverage)	0.1778532160822693	15.064839	3	0.589

Table 1: Earth Mars 5-nodes problem solution for SW application and diverse HW designs.

The 1st row of Table 1 displays the solution for the 5-node application problem solely using software routines of the Zedboard PS. No discernible distinctions from running the algorithm on a desktop computer were observed up to the displayed precision.

3.2. Floating-Point Designs

The 2nd and 3rd rows of Table 1 present equal results to those of the SW version. This outcome was expected as both IP cores use IEEE standard double floating-point precision, avoiding any loss of range or precision in the FPGA [2]. However, the AF of both is below 1, meaning that the HW versions perform worse once compared to the SW version. First, it was inferred that the data transfer between the PS and PL had a negligible impact on the performance. Secondly, a significant reduction in the HW performance was caused by the address of data and stream management operations (steps 1, 2, 4, and 5 of Figure 2). Indeed, by removing the number of cycles associated with these processes, the AF of each HW design was increased to 0.583 and 0.842, respectively. The still inferior performance efficiency can be attributed to the presence of nondeterministic and data-dependent loops, which prevented the use of optimization techniques. Nonetheless, prior studies have presented efficient hardware designs of numeric factorization and matrix-vector multiplication processes. These are characterized by structures and methods qualified for FPGAs. Even so, introducing them into the algorithm would be a complex process requiring extensive changes to the ECOS framework.
To mitigate the additional cycles required by FPGA operations such as data address, transfer, and stream management, a possible workaround is to deploy algorithm sections enclosing multiple successive functions in a single IP block, rather than their separate implementation. These should, anyhow, present a significant impact on the application's execution time, as well as deterministic and data-independent behaviors to exploit the device's capabilities.

3.3. Fixed-Point Designs

The 4th row of Table 1 indicates a loss of precision in both the objective function and fuel mass. In Section 2.3, although values above the selected precision are kept for the sample of data collected, a loss of decimal places still occurs. As later demonstrated, this impacts subsequent computations in ECOS that require more decimal places than those retained by the chosen precision. This aspect has further implications in the design as the AF of 0.526 indicates a decrease when compared to the floating-point design. Indeed, the decrease in precision led to a greater number of iterations k to reach the solution. These strictly related to a heightened number of ECOS iterations, increasing the overall computational time. To express this hypothesis, a second fixed-point design with greater precision coverage was considered.
The effect can be observed by controlling the quantity of `sparseMV` function calls as it links to the solver's iteration count. Specifically, the 3rd, 4th, and 5th rows of Table 1 incorporated 1806, 3021, and 2585 `sparseMV` function calls, respectively. A significant increase in the solver's iteration count was witnessed in both HW versions once compared to the SW version. Furthermore, the design of the 5th row of Table 1 approximates to the number of iterations of the

SW version by covering a broader range of the values utilized by ECOS, despite not being sufficient. In fact, the author of the solver states a need for single floating-point precision when targeting FPGA implementations to cover the dynamic range of numbers that arise from the IPM [3]. This is a frequent concern in FPGA implementations of problems that utilize IPMs, where the employment of fixed-point precision is not recommended [5]. Therefore, for the aforementioned reasons, the later described performance efficiency of fixed-point arithmetic is surpassed by the need to cover the range of values that IPM solvers present. These conclusions align with the AF values of the last 3 rows of Table 1.

To control the behavior differences between floating-point and fixed-point arithmetic, a standalone test problem of the `sparseMV` function was constructed. From it, the HW designs of the 3rd and 4th rows of Table 1 required 11181 and 5680 clock cycles, respectively, to solve the problem, confirming the performance efficiency of fixed-point implementations. The SW version of this function only required 1263 clock cycles. The discrepancy between SW and HW versions corroborated the difficulty of implementing software-oriented applications in FPGAs when optimization techniques are not employed.

Moreover, despite Section 2.2 highlighting the board's resource limitation when deploying onboard guidance algorithms, the performance gap between FPGA and SW implementations should scale positively with the size of STO problems solved by IPM [3]. Lastly, another approach for performance improvement is increasing the IP core frequency to the maximum value that still meets the hardware design timing. Notice, however, increasing the IP core frequency can limit the concurrency of functions due to the aforementioned timing constraints and vice versa.

4. Conclusions

The present work concludes that deploying onboard guidance algorithms in FPGAs is a complex assignment for a conjunction of motives. Highly impacted by the board's resources, the optimization, and structure presented by software-oriented applications pose a demanding transition into hardware. Furthermore, the selection of IP cores shall target sub-routines of the algorithm with high execution time significance that convey deterministic and data-independent behaviors. These enable the parallelism and pipeline benefits FPGAs introduce. If the mentioned attributes are not present, the function's structure and implementation must be adapted accordingly. However, this can lead to complicated algorithm redesign due to the intricacy of IPMs. A suggested procedure is the development of single IP cores that incorporate sections of the algorithm characterized by multiple sequential functions. Thus, the added clock cycles that FPGAs require for operations like data transfer and address are counteracted by its concurrency benefit. Nevertheless, these sections should display relevance to the execution time and the aforementioned attributes. Finally, for onboard guidance algorithms utilizing IPMs, floating-point precision must be employed in IP core designs to cover the imposed range of computational accuracy.

References

[1] Andrea Carlo Morelli et al. Robust Low-Thrust Trajectory Optimization Using Convex Programming and a Homotopic Approach. *IEEE Transactions on Aerospace and Electronic Systems*, 58(3):2103–2116, 11 2022. doi: 10.1109/TAES.2021.3128869.

[2] Louise H. Crockett et al. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014. ISBN 099297870X.

[3] Alexander Domahidi et al. ECOS: An SOCP solver for embedded systems. *2013 European Control Conference, ECC 2013*, pages 3071–3076, 07 2013. doi: 10.23919/ECC.2013.6669541.

[4] Zhenbo Wang et al. Minimum-Fuel Low-Thrust Transfers for Spacecraft: A Convex Approach. *IEEE Transactions on Aerospace and Electronic Systems*, 54(5):2274–2290, 2 2018. doi: 10.1109/TAES.2018.2812558.

[5] Junyi Liu et al. FPGA implementation of an interior point method for high-speed model predictive control. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, 2014. doi: 10.1109/FPL.2014.6927473.