Nonlinear Predictive Control on a Heterogeneous Computing Platform

Bulat Khusainov*, Eric Kerrigan*†, Andrea Suardi*, George Constantinides*

*Department of Electrical & Electronic Engineering
†Department of Aeronautics
Imperial College London
London, SW7 2AZ, UK

†corresponding author: e.kerrigan@imperial.ac.uk

June 18, 2018

Abstract

We propose an implementation of an interior-point-based nonlinear predictive controller on a heterogeneous processor. The workload can be split between a general-purpose CPU and a field-programmable gate array to trade off the contradicting design objectives of control performance and computational resource usage. A new way of exploiting the structure of the KKT matrix yields significant memory savings. We report an 18x memory saving, compared to existing approaches, and a 6x speedup over a software implementation with an ARM Cortex-A9 processor. We also introduce a new release of Protoip, which abstracts low-level details of heterogeneous programming and allows processor-in-the-loop verification.

Keywords: Predictive control, Hardware-software co-design, Scheduling; FPGA; Optimization-based control

1 Introduction

Explicit performance optimization, systematic constraint handling and the inherent capability of dealing with nonlinearities are the main features that explain the success of Model Predictive Control (MPC) in recent decades [36]. In the MPC framework a dynamic optimization problem is solved at each sampling instant, which might restrict the application scope to systems with slow dynamics and/or render expensive implementations limited to high-performance computers.

Conventionally these challenges were addressed on the algorithmic and software levels by developing new optimization problem formulations and generating hardware-efficient code [18]. However, in addition to improvements on the software side, recent developments in reconfigurable computing allowed acceleration of predictive control algorithms on Field-Programmable Gate Arrays (FPGAs), which resulted in low-cost and resource-efficient realizations of custom quadratic programming (QP) solvers for MPC. Consider the following implementations:

- [22, 34] propose several implementations of first order-based MPC on FPGAs. The main computational kernel for this class of algorithms is matrix-vector multiplication, which has a huge parallelization potential. These papers differ in the way this operation is implemented: the former implementation is based on an adder tree, while the latter relies on multiply-accumulate units.

- [41] presents an implementation of linear MPC on a system-on-a-chip that consists of a CPU and an FPGA. Eliminating the states from the decision variables and incorporating inequality constraints in the cost function leads to an unconstrained nonlinear optimization problem, which is solved using Newton's method. For Newton's method the authors propose evaluating derivatives on the CPU and accelerating solving linear systems on the FPGA.

- [21] describes a two-stage architecture for interior point-based predictive control. Interior point algorithms have a similar computational pattern as Newton's method. However, the considered architecture is tailored for solving quadratic programming problems (in contrast to general nonlinear programming problems), which simplifies evaluation of the derivatives. The linear system solver is based on an iterative algorithm, hence the computational logic is reused efficiently.

Other examples of QP-based MPC implementations and/or architectures can be found in [16, 29, 27].

Extending a hardware acceleration approach from linear to nonlinear model predictive control (NMPC), which can be considered as the next logical step, requires mapping numerical integration algorithms on hardware, which is not a trivial task, since dynamic models are problem-dependent. As a result, instead of using systematic dynamic optimization, existing FPGA implementations of NMPC either rely on stochastic optimization [44] or approximate the offline solution with machine learning techniques [4]. These approaches cannot guarantee scalability or closed-loop performance.

Accelerating deterministic algorithms on hardware might be achieved by employing heterogeneous computing platforms that involve both a general-purpose processor with a fixed architecture and FPGA logic. For example, [35] present a heterogeneous implementation of a multiple-shooting based NMPC algorithm. The authors propose implementing integration in software while accelerating a fast gradient-based quadratic programming solver on an FPGA. The reported speedup of the heterogeneous implementation over a software realization is 1.6x and further improvement is limited, since integration and optimization algorithms have comparable computational complexity. This is a consequence of Amdahl's law [2], which states that an algorithm's speedup is limited by the part of the workload that cannot benefit from acceleration.

We present a new heterogeneous implementation of nonlinear interior-point algorithm for predictive control, that was first introduced in [25]. The main features of the proposed implementation are:

- A method for scheduling sparse matrix-vector multiplication within an iterative linear system solvers to enable significant improvements in terms of computation time vs resource usage. For the example considered, an 18x memory saving compared to existing approaches and a 6x speedup over a software implementation are reported.
- Flexible splitting of the algorithmic workload between software and hardware for trading off the computational resource usage against performance.

In addition to the initial results presented in [25], this paper presents the following extensions:

- The whole family of implicit and explicit Runge-Kutta methods are supported for ordinary differential equations integration. In contrast, the initial implementation was limited to Euler method only.
- The optimal control objective is generalized from a quadratic function to nonlinear least squares.
- The proposed controller is experimentally validated in the loop with a gantry crane high-fidelity Simscape [31] model. In [25] the controller was only validated in the loop with a nominal model.

Another contribution of the paper is a new release of the Protoip software tool [40]. Protoip allows quick prototyping and processor-in-the-loop verification of optimization algorithms on a Xilinx Zynq system-on-a-chip (SoC), which contains an ARM processor and FPGA fabric. In contrast to the previous releases, which were focused on pure FPGA implementations, the new version of Protoip allows the incorporation of both an ARM processor and FPGA. Protoip can be used both for quick testing of the proposed implementation from the MATLAB environment and for design and verification of other heterogeneous implementations.

The remainder of the paper is organised as follows: Section 2 describes the considered optimal control problem formulation; existing NMPC algorithms, with a focus on suitability for hardware implementation, are reviewed in Section 3; in Section 4 a heterogeneous computer-based implementation of NMPC is presented, followed by experimental setup description (Section 5) and experimental results (Section 6). Note, that Protoip is a part of the experimental setup and hence presented in the corresponding section. Section 7 concludes the paper.

2 Optimal Control Problem formulation

We consider nonlinear time-invariant systems that can be described as an ordinary differential equation (ODE) of the form

$$\dot{x}(t) = f_c(x(t), u(t)), \tag{1}$$

where $f_c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. We consider the nonlinear optimal control problem (OCP) with initial state $\hat{x}$ and prediction horizon T:

$$\min_{x,u,s} \quad \int_0^T ||h(x(t), u(t), s(t))||_2^2 dt + ||h_T(x(T), s_T)||_2^2 \tag{2a}$$

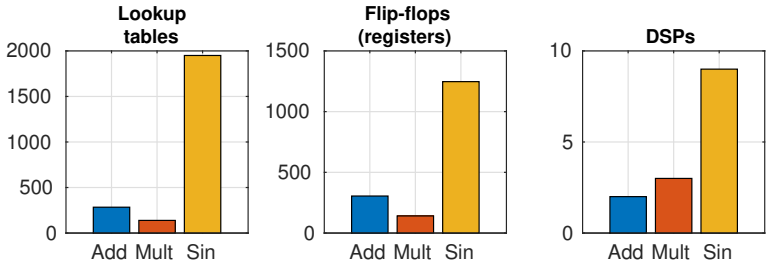

Figure 1: FPGA resource usage for different computations with single precision floating point arithmetic. Data is obtained from the vendor's high-level synthesis tool, Vivado HLS. Sine operator implementation is based on lookup tables.

$$
\begin{aligned}
\text{subject to:}\quad & x(0) = \hat{x}, && && \text{(2b)}\\
& \dot{x}(t) = f_c(x(t), u(t)), && \forall t \in [0, T] && \text{(2c)}\\
& q(x(t), u(t), s(t)) = 0, && \forall t \in [0, T] && \text{(2d)}\\
& q_T(x(T), s_T) = 0, && && \text{(2e)}\\
& x_l \leq x(t) \leq x_u, && \forall t \in [0, T] && \text{(2f)}\\
& u_l \leq u(t) \leq u_u, && \forall t \in [0, T] && \text{(2g)}\\
& s_l \leq s(t) \leq s_u, && \forall t \in [0, T] && \text{(2h)}\\
& x_{Tl} \leq x(T) \leq x_{Tu}, && && \text{(2i)}\\
& s_{Tl} \leq s_T \leq s_{Tu}, && && \text{(2j)}\\
& && && \text{(2k)}
\end{aligned}
$$

where $h : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{n_S} \to \mathbb{R}^{n_h}$ and $h_T : \mathbb{R}^n \times \mathbb{R}^{n_{ST}} \to \mathbb{R}^{n_{hT}}$. Slack trajectory s and slack variable s_T are introduced alongside with equations (2d) and (2e), which is a common technique that allows for the handling of general nonlinear inequality constraints [43]. Note that in the general case $s(T) \neq s_T$. The presented formulation can be generalized for time-varying reference tracking, which, as will be shown later, requires only changing the software part of the algorithm.

3 Nonlinear predictive control algorithms

Direct solution of the continuous-time optimal control problem (2) involves two main stages: integration, i.e. solving the ordinary differential equation (ODE), and optimization. Implementing integration on an FPGA is not desirable because of the following reasons:

- The ODE (2c) may involve mathematical expressions (e.g. sine and square root) that in contrast to standard addition and multiplication operations, require significant amounts of computational resources (Figure 1) and might be unsuitable for pipelining. For examples on implementation of nonlinear operators on FPGAs refer to [9, 11].
- A vector function f_c is composed of scalar functions that might have different underlying mathematical operations and different evaluation complexities, i.e. f_c might have *irregular* structure. Irregularity potentially limits reusing computational logic and speedup by parallelization.

Optimization algorithms, on the other hand, can benefit from hardware acceleration due to (i) their iterative nature, which is beneficial for reusing computational logic, and (ii) the fact that underlying linear algebra algorithms can be efficiently mapped onto hardware [21].

Taking the above into account we consider two classes of algorithms for solving (2): shooting-based and direct transcription algorithms [5]. The common feature of shooting methods is decoupling the ODE and optimization solvers. Accelerating only the latter does not result in significant improvements, due to Amdahl's law, since the workloads of the two operations are comparable. In contrast, direct transcription algorithms transform (2) directly to a discrete OCP by approximating the ODE with algebraic equations based on numerical integration equations, i.e.

$$\min_{\substack{x_0,\dots,x_N\\ u_0,\dots,u_{N-1}\\ r_0,\dots,r_{N-1}\\ s_0,\dots,s_{N-1},s_T}} \sum_{k=0}^{N-1}\left(T_s||h(u_k,x_k,s_k)||_2^2\right) + ||h_T(x_N,s_T)||_2^2 \tag{3a}$$

$$\text{subject to:} \tag{3b}$$
$$x_0 = \hat{x}, \tag{3c}$$
$$c(x_k,u_k,r_k) = \begin{bmatrix} x_{k+1} \\ 0 \end{bmatrix}, \qquad k=0,\dots,N-1 \tag{3d}$$
$$q(x_k,u_k,s_k) = 0, \qquad k=0,\dots,N-1 \tag{3e}$$
$$q_T(x_N,s_T) = 0, \tag{3f}$$
$$x_l \le x_k \le x_u, \qquad k=0,\dots,N-1 \tag{3g}$$
$$u_l \le u_k \le u_u, \qquad k=0,\dots,N-1 \tag{3h}$$
$$s_l \le s_k \le s_u, \qquad k=0,\dots,N-1 \tag{3i}$$
$$x_{Tl} \le x_N \le x_{Tu}, \tag{3j}$$
$$s_{Tl} \le s_T \le s_{Tu}, \tag{3k}$$
$$\tag{3l}$$

where N and T_s, respectively, denote the horizon length and the sampling time,

$$r_k = \left[r_k^{(1)T} \ \cdots \ r_k^{(l)T}\right]^T \in \mathbb{R}^{nl}$$

is a vector of integrator intermediate stages and l is the number of integrator stages per sampling instant.

With a trapezoidal integrator, (3d) would be given by:

$$x_k + T_s\left(r_k^{(1)} + r_k^{(2)}\right)/2 = x_{k+1} \tag{4a}$$
$$r_k^{(1)} - f_c(x_k,u_k) = 0 \tag{4b}$$
$$r_k^{(2)} - f_c\left(x_k + T_s\left(r_k^{(1)} + r_k^{(1)}\right)/2, u_k\right) = 0 \tag{4c}$$

The OCP (3) can be transformed into an NLP of the form

$$\min_{\theta} \quad \frac{1}{2}||f(\theta)||_2^2 \tag{5a}$$
$$\text{subject to} \quad p(\theta) = 0, \tag{5b}$$
$$J\theta - d \le 0, \tag{5c}$$

where $\theta = [x_0^T \ u_0^T \ r_0^T \ s_0^T \dots x_{N-1}^T \ u_{N-1}^T \ r_{N-1}^T \ s_{N-1}^T \ x_N^T \ s_T^T]^T$. Note that J is a matrix of zeros and positive/negative ones since (3) contains only bound inequalities.

NLP (5) incorporates both integration and optimization, which potentially opens the possibility of accelerating both subproblems. Primal-dual interior-point algorithms have proven their efficiency

Algorithm 1 Primal-dual interior-point method for NLP

Initial point $[\theta_{[0]}^T, \nu_{[0]}^T, \lambda_{[0]}^T]^T : \lambda_{[0]} > 0, G\theta_{[0]} - d < 0$
Reduction parameter $0 < \sigma \leq 1$
for $k = 0$ to n_{iter} **do**
  $A_{[k]} = \begin{bmatrix} H_{[k]} + GW_{[k]}G^T & \nabla_\theta p^T(\theta_{[k]}) \\ \nabla_\theta p(\theta_{[k]}) & 0 \end{bmatrix}$
  $b_{[k]} = \begin{bmatrix} r_{dual} \\ r_{eq} \end{bmatrix}$
  Solve $A_{[k]} z_{[k]} = b_{[k]}$ for $z_{[k]} = \begin{bmatrix} \Delta\theta_{[k]} \\ \Delta\nu_{[k]} \end{bmatrix}$
  $\Delta\lambda_{[k]} = -\Lambda_{[k]}e - G^{-1}(\theta_{[k]})\mu e - G^{-1}(\theta_{[k]})\Lambda_{[k]}\nabla_\theta g(\theta_{[k]})\Delta\theta_{[k]}$
  $\alpha_{[k]} = \max_{(0,1]}\alpha : \lambda_{[k]} + \alpha_{[k]}\Delta\lambda_{[k]} > 0, G(\theta_{[k]} + \Delta\theta_{[k]}) - d < 0$
  $[\theta_{[k+1]}^T, \nu_{[k+1]}^T, \lambda_{[k+1]}^T]^T = \alpha_{[k]}[\Delta\theta_{[k]}^T, \Delta\nu_{[k]}^T, \Delta\lambda_{[k]}^T]^T + [\theta_{[k]}^T, \nu_{[k]}^T, \lambda_{[k]}^T]^T$
end for

for the numerical solution of optimal control problems [38]. Moreover, with interior-point algorithms, the structure of the KKT matrix associated with the OCP remains fixed (unlike with active set methods), which is desirable for hardware implementations [21].

The next section introduces a structure-exploiting heterogeneous implementation of an interior-point algorithm for solving (5). The implementation supports the use of Runge-Kutta family integrators for temporal discretization, considering Butcher tableau as a design parameter. However, results for memory saving, algorithm acceleration and closed loop simulations are obtained with a trapezoidal integrator (4), which belongs to the family of implicit integrators and hence allows efficient handling of stiff ODEs.

4 Algorithm and implementation details

4.1 Primal-dual interior-point algorithm

Interior-point algorithms solve NLPs by incorporating inequality constraints into the objective function using a logarithmic barrier function scaled with a barrier parameter. The modified problem is solved by performing consecutive Newton steps with or without globalization. Algorithm 1 and Appendix A outline a primal-dual interior-point algorithm for solving (5). The algorithm computes a barrier parameter based on the current complementary value and reduction parameter, which is one of the simplest and most common approaches [32]. Furthermore, there is no globalization strategy incorporated in the algorithm. This choice is justified by the fact that line search and trust region globalization algorithms involve repetitive evaluation of $f(\theta)$ and $p(\theta)$, which, as was previously mentioned, often have irregular structure and hence discourage efficient acceleration. An extensive comparative study of barrier update strategies and globalization techniques for nonlinear interior-point methods can be found in [32].

Another algorithmic choice that requires justification is the Hessian approximation method. Most of existing interior-point algorithms for large scale optimization use either Broyden-Fletcher-Goldfarb-Shanno (BFGS) [8] or Gauss-Newton approximators. The BFGS algorithm, if applied blockwise, can achieve a block diagonal structure for the Hessian approximation matrix when used for optimal control applications [20]. The Gauss-Newton algorithm, in addition, preserves sparsity within each block, which is particularly beneficial taking into account scratchpad memory limitations of reconfigurable computers. However, memory savings come at the price of narrowing the application scope to least-squares types of problems. In this work we accept this limitation and use the Gauss-Newton approach.

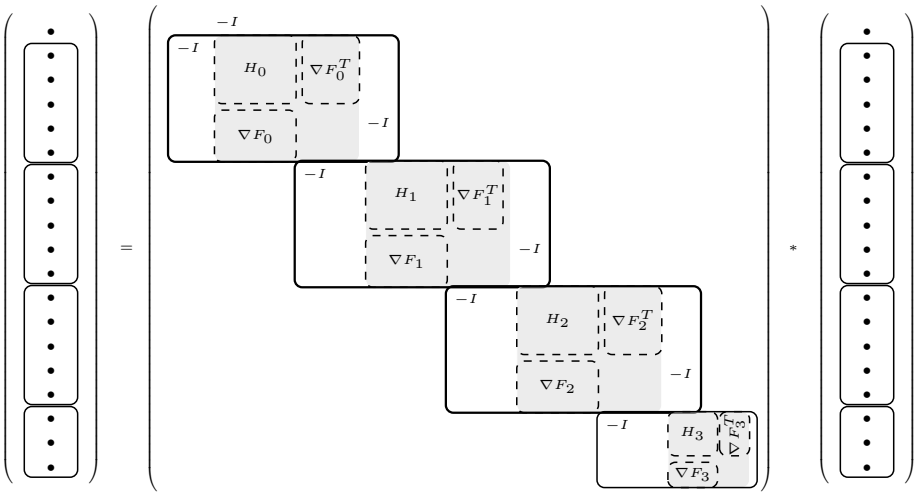

Figure 2: Structure of the KKT matrix assosiated with the problem (5). H_i and ∇F_i denote approximate Hessians and Jacobians of the corresponding sampling instant in the prediction horizon.

The main workload of the algorithm is concentrated in solving the system of linear equations. The matrix associated with the problem is symmetric and can be reordered to achieve the structure presented in Figure 2. A symmetric system of linear equations can be solved using direct methods, e.g. LDL decomposition, or iterative methods, e.g. Minimum Residual (MINRES) algorithm [33]. Decomposition algorithms often involve many division computations, which are more complicated from a hardware implementation point of view compared to addition and multiplication. Moreover, parallelizing computations is not straightforward with direct algorithms. In contrast, iterative methods mainly rely on matrix-vector multiplications, while having very few division and square root computations. In this work we use the MINRES algorithm, which can be efficiently mapped onto hardware [6] and is well studied in relation to optimal control applications [38]. The algorithm performs minimization of the residual $||b - Az||_2$ over a Krylov subspace, which is constructed iteratively by the Lanczos kernel [33]. The Lanczos kernel is based on a three-term recurrence, hence there is no need to store the entire Krylov subspace (unlike with the generalized minimal residual method [37]). From an implementation point of view, the MINRES algorithm is mostly based on addition and multiplication operations, requiring only two scalar divisions and two scalar square root computations per iteration.

4.2 Proposed implementation

Algorithm 1 with the MINRES linear solver can be visualized with the block diagram in Figure 3. We consider four different ways of splitting the workload between software and hardware:

- The entire algorithm is implemented in software, i.e. ARM Cortex-A9 processor. We denote this implementation by SW.
- Only the matrix-vector multiplication is accelerated in hardware and the rest is implemented in software (HG_1).
- The whole Lanczos kernel is accelerated in hardware and the rest is implemented in software (HG_2).
- The whole linear system solver is accelerated in hardware and the rest is implemented in software (HG_3).

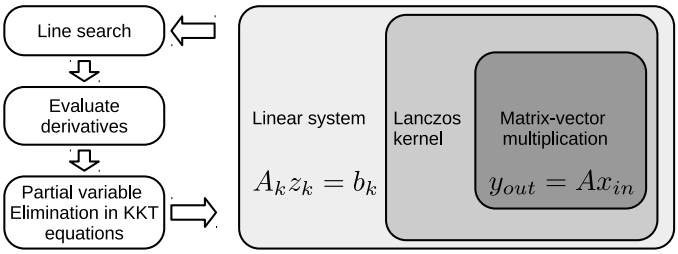

Figure 3: Algorithm 1 flow diagram with MINRES solver.

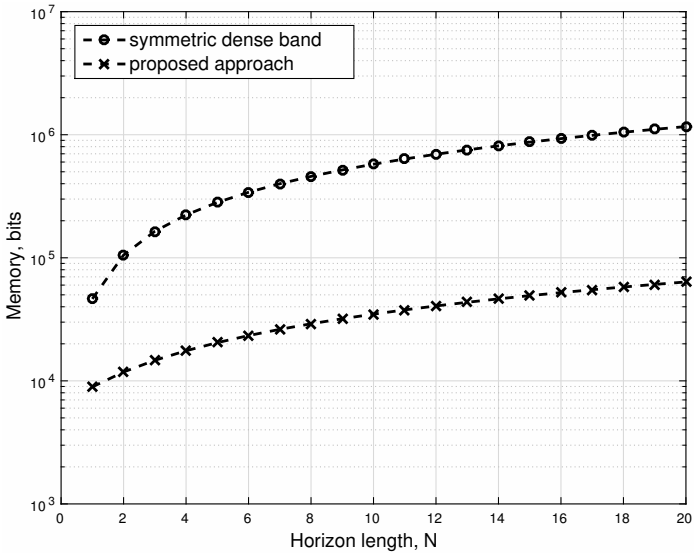

Figure 4: Memory requirements for the storing matrix on Figure 2. ODE model and OCP parameters are presented in Sections 5 and 6, trapezoidal method is used for ODE integration. Details on dense band implementation can be found in [7]. Actual memory savings may vary depending on the memory block size of a particular computing platform.

The linear system A_k changes at every interior point iteration, hence the data has to be transferred online. However, this is not a bottleneck, since the same A_k is reused multiple times within the iterative linear solver. Note that for all considered options, the Jacobians evaluations are implemented in software to avoid synthesising resource-consuming nonlinear operators. Operations that are implemented in hardware can be classified into:

- *Scalar operations*, which do not require acceleration.
- *Vector-vector operations*, which can be efficiently pipelined in hardware.
- *Matrix-vector multiplication*, which is the most resource-consuming part. Efficient implementation requires exploiting sparsity.

One possible way to exploit the structure of A_k is based on considering the matrix as banded [21]. In this case the number of parallel computations is defined by the bandwidth and cannot be changed with respect to resource availability. In this work we treat the matrix as block sparse and, in addition, exploit sparsity within each block, which can result in an 18x saving in memory (Figure 4).

Matrix-vector multiplication can potentially be parallelized by simultaneous processing of the solid line blocks in Figure 2, which correspond to different sampling instants. However, consecutive

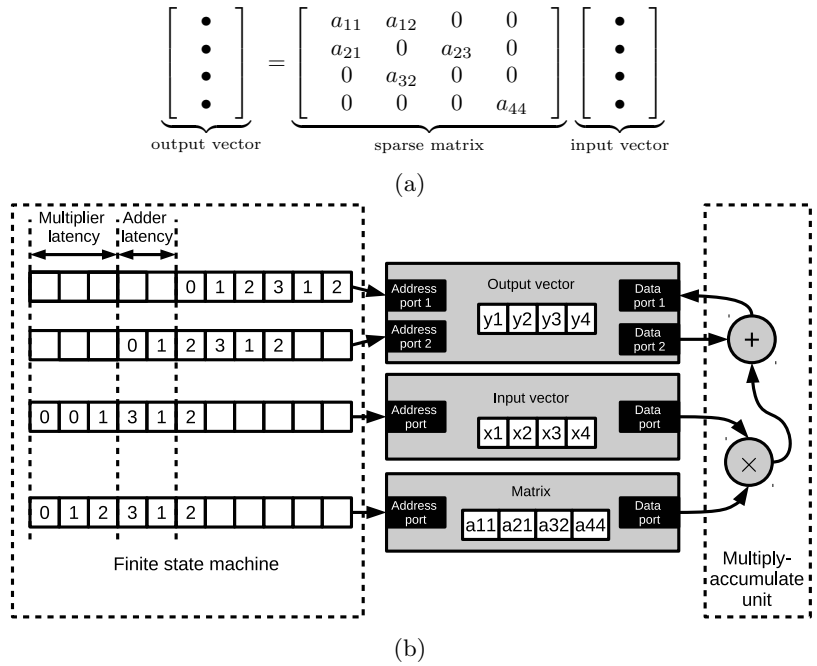

Figure 5: An example of a sparse matrix (a) and the corresponding circuit (b) for matrix vector multiplication.

blocks are coupled, i.e. each block accesses areas in the input vector associated with its neighbours, which results in restrictions on memory partitioning. It can be noted that blocks are coupled only via negative identity matrices. For a matrix-vector multiplication, handling negative identity matrices is reduced to data transfer operations (with changing the sign bit) and does not require any arithmetic operators. After negative identity matrices are processed, the remaining parts (grey area) can be parallelized, since there is no overlap in accessing input vector partitions.

The efficiency of multiplication also depends on how sparsity within each block is handled. For example, consider the sparse matrix in Figure 5. Each non-zero element in the matrix corresponds to a multiply-accumulate (MAC) operation, i.e. each non-zero element is multiplied by a corresponding element of the input vector and added to the corresponding element of the output vector. Although the order of processing non-zero elements with floating point arithmetic might affect the final result, in practice this effect is assumed to be negligible. More importantly, this order has an impact on output vector read-write dependencies and hence pipelining possibilities. For the considered example, processing of a_{12} cannot be started before a_{11} is fully processed, since both elements require writing data to the same element of the output vector. Hence, MAC operations have to be scheduled in such a way that the distance (in terms of computer clock cycles) between processing non-zero elements from the same row is maximized. This scheduling problem can be formulated as an optimization problem

$$\max_{d, S_{sched}} \quad d \tag{6a}$$

subject to:

$$|t(a_{ij}) - t(a_{kl})| > d \qquad \forall a_{ij} \in M, \forall a_{kl} \in M : i = k, j \neq l \tag{6b}$$

$$1 \leq t(a_{ij}) \leq N_{nz} \qquad \forall a_{ij} \in M \tag{6c}$$

$$t(a_{ij}) \neq t(a_{kl}) \qquad \iff i \neq k, j \neq l \tag{6d}$$

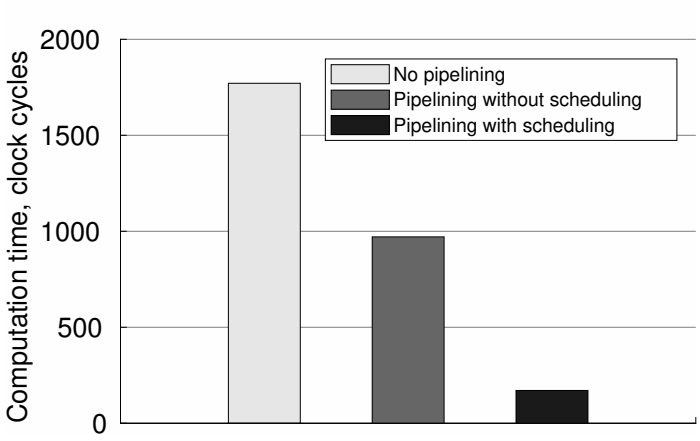

Figure 6: The impact of scheduling on computation time of a sparse matrix-vector multiplication (for each gray block in Figure 2). It is assumed that the latency for the addition and multiplication units are 6 and 5 clock cycles, respectively. The number of nonzero elements is 161, which is the case for the example described in Sections 5 and 6.

where $d \in \mathbb{Z}$ is a slack variable, $M = \{a_{11}, a_{12}, a_{21}, a_{23}, a_{32}, a_{44}\}$ is the set of non-zero elements, N_{nz} is the number of non-zero elements, $t(a_{ij})$ is the number of time instant, at which processing of a_{ij} starts, and $S_{sched} = [t(a_{11}), t(a_{12}), t(a_{21}), t(a_{23}), t(a_{32}), t(a_{44})]^T \in \mathbb{Z}^{N_{nz}}$. Since all blocks in Figure 2 have the same sparsity pattern, the scheduling problem is solved only once during design. The resulting MAC circuit for the considered example is shown in Figure 5. Note that due to symmetry of A_k only the lower triangular part is stored.

The impact of scheduling on the time taken for sparse matrix-vector multiplication is presented in Figure 6. For this case study, problem (6) was solved using the YALMIP built-in branch and bound solver [30]. If pipelining is implemented without scheduling, the *initiation interval* is equal to the adder latency, which limits potential improvement. Initiation interval is defined as the number of clock cycles that elapses between starting processing two successive non-zeros. With a systematic scheduling approach, read-write dependencies are handled optimally, which allows one to start processing a new non-zero element after the previous as quickly as possible, i.e. achieving the minimal initiation interval. For the ODE model presented in Section 5 it was possible to achieve initiation interval of one clock cycle.

Although matrix-vector multiplication can be parallelized by allocating a MAC unit to each grey block (Figure 2), we propose trading off computation time against resource usage by varying the number of MAC units, as shown in Figure 7. The number of MAC units is denoted by P.

4.3 Limitations of the proposed implementation

We highlight certain limitations of the proposed implementation:

- For matrix-vector multiplication, input and output vectors have to be partitioned in accordance with the matrix Figure 2, which, depending on the parallelism level P, might increase FPGA block memory usage. This issue can be partially addressed by placing corresponding partitions of the matrix and input vector in the same memory block, which will not affect performance, provided the computing platforms supports dual-port memory blocks. Another possible solution is implementing vector partitions using lookup tables (provided partitions are sufficiently small) instead of using block memory.

- In contrast to [7], our implementation requires control logic, which might limit FPGA clock

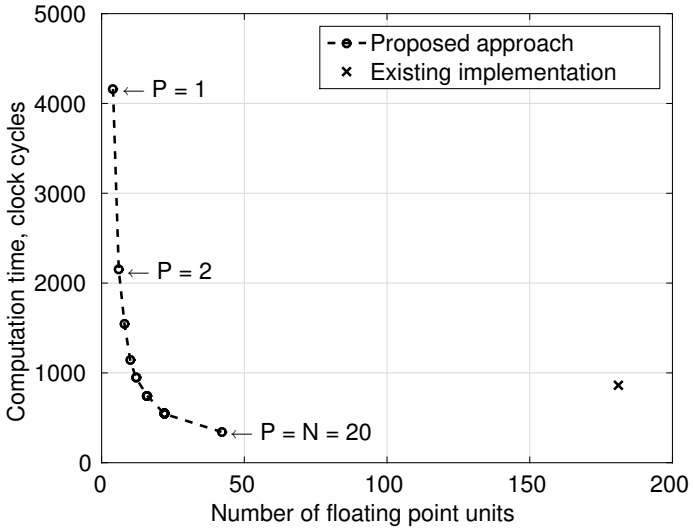

Figure 7: Trading off computational time against processor resource usage for matrix-vector multiplication (Figure 2). Each circle represents a design with a certain degree of parallelism. The floating point unit is either floating point addition or multiplication. For the considered test case, $N = 20$ and the ODE model is given in Section 5. It is assumed that the latency of the addition and multiplication units are 6 and 5 clock cycles, respectively. Details on the existing implementation can be found in Boland and Constantinides [7].

frequency. However, matrix-vector multiplication, being a part of a larger algorithm, is often not the main bottleneck for increasing clock frequency [6].

- Applying a preconditioner directly to original linear system may destroy in-block sparsity and hence eliminate memory savings. To overcome this limitation, preconditioners can be applied indirectly by performing two matrix-vector multiplications per MINRES iteration so that sparsity patterns are preserved, both for preconditioner and the original matrix [15]. In this work we use a sparsity-preserving prescaler described in [23]. Although preserving sparsity, this prescaler changes negative identity matrices into general diagonal matrices, which slightly complicates the algorithm.

5 Experimental setup

Experimental setup represents a closed-loop system that consists of a gantry crane model and a heterogeneous computer running the predictive controller. The closed-loop simulation is managed by Protoip, a software tool for rapid prototyping of online optimization algorithms on reconfigurable computing platforms. This section will provide detailed information on the experimental setup, including a new release of Protoip, heterogeneous computer and the gantry crane model.

5.1 Protoip - a new release

The initial release of Protoip [40] represents a software tool for rapid deploying and verification of online optimization algorithms on FPGAs using hardware vendor's tools on the underlying level (Xilinx Vivado, Vivado HLS, SDK). The tool is available on Xilinx Tcl Store, an open source repository provided by Xilinx. Protoip is written in Tcl, an interpreted programming language that is widely used for computer aided design. A Matlab interface is provided to allow working in

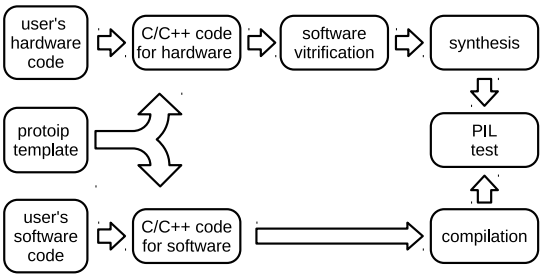

Figure 8: SoC design flow with Protoip.

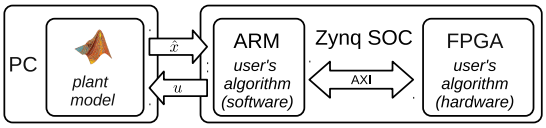

Figure 9: Processor-in-the-loop test with Protoip.

conjunction with state-of-the-art optimization and numerical integration toolboxes. The previous version of Protoip was dealing only with pure FPGA implementations, i.e. the CPU was not involved in computations. The new release adds possibility of splitting the workload between CPU and FPGA within a single chip to exploit strengths of each subsystem and trade off FPGA logic usage against algorithm performance. The design flow for heterogeneous implementations consists of two parallel branches which correspond to software and hardware parts of a given algorithm accordingly (Figure 8). Initially, the user provides two pieces of C/C++ code: software code for CPU and hardware code for FPGA. This step can be performed either manually or automated by using synthesizable code generation tools [39]. On the next stage, Protoip performs software verification of the hardware code using automatically generated testbench files, which allows detecting errors on the early stages of the design flow avoiding time consuming circuit synthesis process. Following this, FPGA circuit is synthesised and software code is compiled. On the final stage of the design flow implementation is deployed on a heterogeneous platform and Processor-in-the-loop (PIL) test is performed.

Processor-in-the-loop test implies simulating the controlled plant in Matlab environment on PC, while performing computations on an embedded platform (Figure 9). Desktop PC to embedded platform communication speed and reliability might be critical for processor- in-the-loop simulations. The previous release of Protoip was relying on UPD/Ethernet interface, which is a common and relatively fast protocol, however, having no transmission guarantees. Lack of reliability might be crucial for design space space exploration, where processor-in- the-loop simulations may run autonomously for several weeks. A possible solution to this problem is employing PCI interface, which has proven to be reliable solution for FPGA to PC communication [24]. However, PCI interface is rarely available on low-cost prototyping boards. For that reason it was decided to keep using UDP/Ethernet so that no additional hardware is introduced. To improve reliability another communication layer with error checking and retransmission mechanism was implemented on the top of UDP. As a result communication reliability was improved without changing hardware requirements.

Another feature of the release is a new way of managing Vivado, Vivado HLS and Xilinx SDK projects. In particular:

- Users get flexibility of managing projects (e.g. adding new libraries, project parameters). With the previous version of Protoip only predefined set of project parameters was supported.
- A possibility of handling multiple file C/C++ projects is added. This make Protoip more friendly in relation to code generation tools. Consider [39], a C code generation tool for model predictive control based on operator splitting methods, which uses Protoip on the underlying level and is capable generating code for FPGAs.

Summary of the new Protoip release features, i.e. contributions of this work:

- Possibility of prototyping heterogeneous implementations.

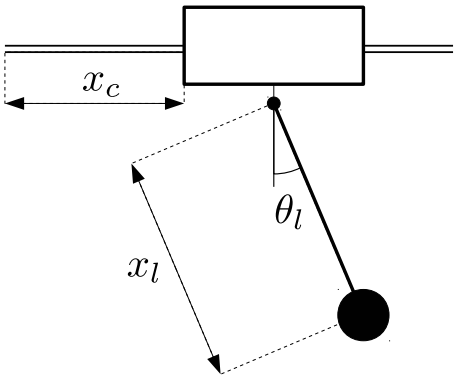

Figure 10: A schematic drawing of the gantry crane setup.

- A new communication layer on the top of UDP/Ethernet for reliable processor-in-the-loop simulations.
- A new way of managing application projects to allow working in conjunction with code generation tools.

5.2 Target heterogeneous computer

Protoip supports the entire family of Xilinx Zynq-7000 devices. The proposed implementation was tested on a Xilinx Zynq-7000 XC7Z020 SoC with dual-core ARM Cortex-A9 processor (only one core was used in this work) and Artix-7 FPGA logic that contains 53200 six-input Lookup Tables (LUTs) and 106400 Flip-Flops (FFs). In addition to general purpose logic the Artix-7 provides special purpose units: 220 DSP blocks and 140 block RAMs with total capacity 4.9 Mb, which is a relatively small amount compared to an average modern CPU RAM. Communication between software and hardware was performed via Advanced eXtensible Interface (AXI) with burst mode support that allows reading/writing of words every clock cycle. We used the Zedboard development kit [19], which supplements the Zynq platform with auxilary circuits to provide an easy access to the communication interfaces, including JTAG for programming FPGA and Ethernet for data exchange.

5.3 Gantry crane

For the purpose of closed-loop simulations, Simulink Multibody library-based [31] gantry crane model was used, which allowed incorporation of various types of friction including viscous, breakaway and Coulomb frictions. The setup consists of a payload ($m = 0.47$ kg) hanging on a rope attached to a moving cart (Figure 10). The system is actuated by two Pulse Width Modulation (PWM) controlled motors, which move the cart along the rail and lift/lower the load. Motors are equipped with internal velocity controllers, hence the inputs to the system are velocity setpoints. Cart position x_c, rope length x_l and deflection angle θ_l are measured by encoders with a resolution equal to 4096 pulses per rotation. Corresponding velocities are estimated using a second order low-pass filter cascaded with a differentiator with Laplace transform

$$Y_{est}(s) = \frac{s\omega^2}{s^2 + 2\omega\zeta + \omega^2}. \tag{7}$$

In this experiment $\omega = 20\pi$ rad/s, $\zeta = 0.7$. For the purpose of digital implementation (7) is discretized using a Tustin transformation with sampling time $T_{est} = 0.01$s. Note that estimator and controller can be sampled at different rates.

The motion of the crane with a variable cable length can be described by [14]

$$\ddot{\theta}_l x_l + 2\dot{x}_l\dot{\theta}_l + \ddot{x}_c\cos(\theta_l) + g\sin(\theta_l) = 0, \tag{8}$$

where g is the acceleration of gravity.

It is assumed that the dynamics between the motor velocity setpoints and corresponding velocities are described by the first-order lag relation with Laplace transform

$$Y_{motor}(s) = \frac{1}{\tau s + 1}, \tag{9}$$

where τ is the time constant. The following state-space model can be obtained:

$$\dot{x}_c = v_c \tag{10a}$$
$$\dot{v}_c = \frac{-v_c + u_c}{\tau_c} \tag{10b}$$
$$\dot{x}_l = v_l \tag{10c}$$
$$\dot{v}_l = \frac{-v_l + u_l}{\tau_l} \tag{10d}$$
$$\dot{\theta}_l = \theta_l \tag{10e}$$
$$\ddot{\theta}_l = \frac{1}{x_l}\left(\frac{-v_c + u_c}{\tau_c}\cos(\theta_l) + g\sin(\theta_l) + 2v_l\dot{\theta}_l\right) \tag{10f}$$

where v_c and v_l denote corresponding velocities. Time constants $\tau_c = 0.13s$ and $\tau_l = 0.07s$ were identified using sum-of-sinusoids input signal postprocessing data with the nonlinear least squares algorithm from [31].

In this work controller performance was initially verified by using the same model for predictive control and plant simulation adopting the CasADi [3] front-end of the CVODES numerical integrator [17]. Following initial verification a heterogeneous MPC controller was tested in the loop with Simscape model within the Simulink environment. Incorporating the Zynq platform into the Simulink environment was performed using the Protoip Application Programming Interface (API), which is implemented in C, and the Simulink Coder package, which allows calling external C/C++ functions.

6 Experimental results

Four implementations (software only and three heterogeneous) of Algorithm 1 were deployed and tested with Protoip. Single precision floating point data arithmetic was used, clocking the CPU and FPGA at 667 MHz and 167 MHz, respectively. The following control objectives were selected:

$$h(u,x,s) = \begin{pmatrix} x_c + x_l\sin(\theta_l) \\ x_l\cos(\theta_l) - 0.5 \\ \dot{\theta}_l \\ 10^{-4}u_c \\ 10^{-4}u_l \end{pmatrix}, \quad h_T(x_N, s_T) = \begin{pmatrix} x_c + x_l\sin(\theta_l) \\ x_l\cos(\theta_l) - 0.5 \\ \dot{\theta}_l \end{pmatrix}. \tag{11}$$

Note that this formulation has proven to be a tuning-free solution for gantry crane control, in contrast to the standard quadratic objective approach, which often relies on a time consuming tuning process [10]. Actuation limits dictated by the motor characteristics were incorporated as input constraints:

$$-0.15 \leq u_c \leq 0.15 \tag{12a}$$

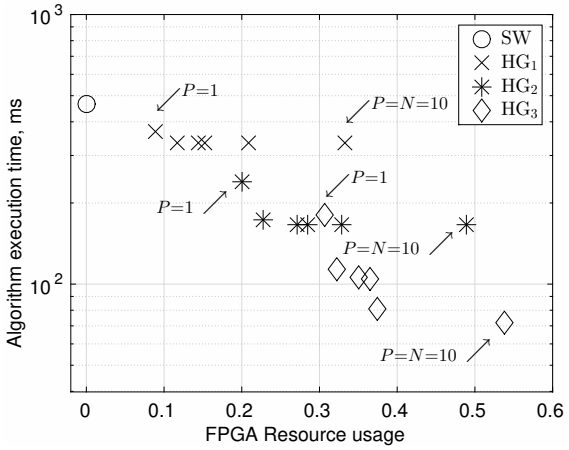

Figure 11: Trading off algorithm execution time against resource usage by splitting the workload between software and hardware. For all implementations $N = 10$.

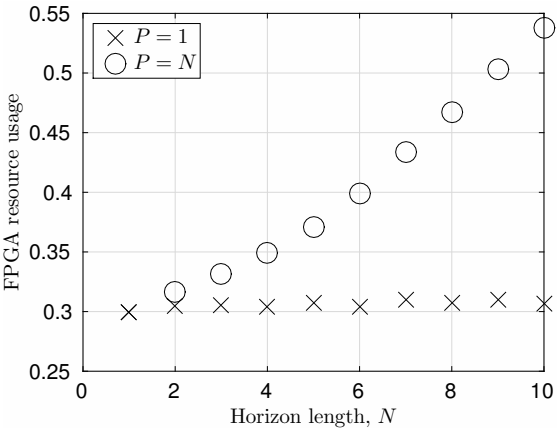

Figure 12: The impact of horizon length on FPGA resource usage with HG_3 implementation.

$$-0.15 \leq u_l \leq 0.15 \tag{12b}$$

For all designs considered $T_s = 100$ ms, $N = 10$, the number of interior-point iterations $n_{iter} = 15$ and the number of MINRES iterations was set to be equal to corresponding linear system size. Suitability of these parameters was experimentally confirmed based on closed-loop simulation results.

With the proposed implementation computation time can be traded off against resource usage either by changing the level of parallelism (i.e. the number of MAC units for matrix-vector multiplication) in the hardware accelerator or by shifting MINRES algorithm layers between software or hardware. These tradeoffs are shown in Figure 11. Resource usage is calculated as a Euclidian norm of a vector that contains the relative utilization of different FPGA resources (LUTs, FFs, DSPs and BRAMs). For each heterogeneous implementation (HG_1, HG_2 and HG_3) the parallelism factor P was varied from 1 to N. Note that although $N = 10$, there are only 6 designs for each implementation. This happens because some values of P are identified as inefficient at code generation stage. For this example any design with $P = 6$ to 9 will lead to the same performance as $P = 5$. We highlight some other conclusions that can be drawn based on Figure 11:

- Considering execution time and resource usage as two contradicting design objectives within a multi-objective optimization problem, it can be seen that Pareto-optimal designs are achieved by splitting the workload between software and hardware in different ways. This justifies the flexibility of the software-hardware splitting of the proposed implementation.

- For the HG_1 and HG_2 implementations, after reaching a certain degree of parallelism for matrix-vector multiplication, increasing parallelism further does not result in any speed up. This happens due to the fact that the Lanczos kernel (implemented on the FPGA) and the remaining part of the MINRES algorithm (implemented on the CPU) run in parallel and the overall execution time is defined by the slowest branch, which is the remaining part of MINRES.

- For the test case considered the maximum speedup of hardware-accelerated implementations over a pure software realization is 6x. Since this speedup comes at the price of significantly increased resource usage, it might be reasonable to compare the fastest design against other Pareto-optimal designs before making the final design choice.

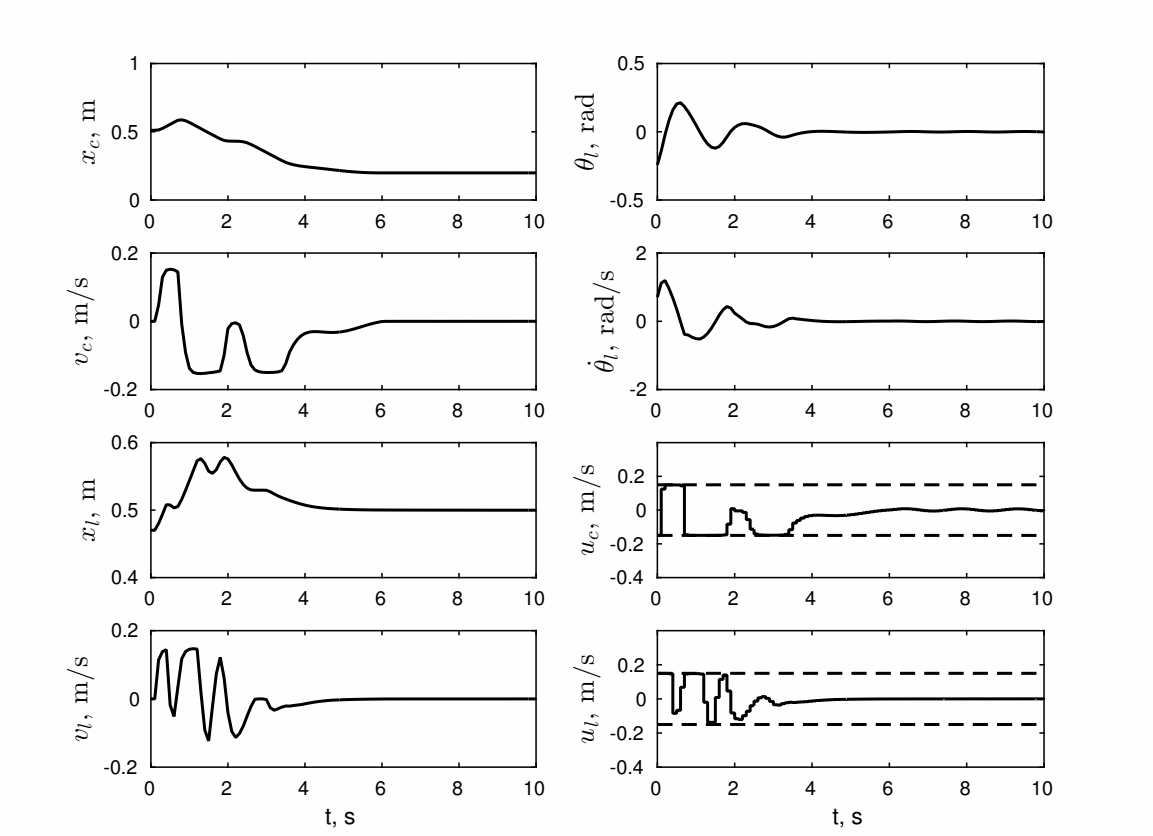

Figure 13: Processor-in-the-loop simulation results with HG_3 implementation. $N = 10$, $P = 10$.

Increasing the prediction horizon N, while improving closed-loop performance, normally results in implementations with high resource consumption. Consider Figure 12 that visualizes scaling of FPGA resource usage with the prediction horizon for two extreme cases: $P = 1$ and $P = N$. It can be observed that changing the degree of parallelism allows taking control over resource scaling. Note that for sequential processing of KKT matrix blocks ($P = 1$) resource scaling can hardly be noticed. This happens due to the fact that the memory footprint is so small that fixed-size FPGA memory blocks are not fully filled for $N = 1$ nor for $N = 10$. Further synthesis details can be found in Appendix B.

We present closed-loop simulation results obtained with an HG_3 implementation in the loop with a Simscape model (Figure 13). Starting from the initial condition $\hat{x} = [0.5\ 0\ 0.7\ 0\ -0.2\ -0.5\ -0.15\ -0.15]^T$ the controller drives the system to the desired point according to the objective (11) while satisfying constraints (12).

Finally, we compare our implementation against other optimization-based gantry crane controllers implementations (Table 1), namely a desktop computer implementation [42] and a Programmable Logic Controller (PLC) implementation [28]:

- Implementations [42] and [28] compute one and two iterations of a Sequential Quadratic Programming (SQP) and projected gradient algorithms, respectively, in order to reduce the feedback delay, hence they do not attempt to solve the OCP directly. This idea of Real-Time Iterations (RTI) was initially proposed in [12]. Comparing closed-loop performance of the RTI approach against 'classical' nonlinear MPC is out of the scope of this paper.

Table 1: Comparison of optimization-based gantry crane controllers implementations.

	[42]	[28]	This paper
Algorithm	SQP	Projected gradient	Interior point
Number of iterations	1	2	15
ODE integration algorithm	Runge-Kutta family (not specified)	Euler (explicit)	Trapezoidal (implicit)
Objective	Quadratic	Quadratic	Nonlinear least squares
Prediction horizon, s	1	1	1
Sampling time, ms	10	2	100
Algorithm time, ms	1.09	1–1.5	96
Computing hardware	Intel Xeon 2.53 GHz quadcore	Festo PLC CECX-X-C1	Zynq-7000 XC7Z020
Maximum power consumption for the hardware platform	Processor model is not specified. E.g. for Intel Xeon X3440: 95W [1]	69W [13] (excluding I/O modules)	< 2W (for all implementations; includes CPU and FPGA)

- In this work, ODE integration is performed using an implicit integrator. For shooting-based approaches implicit integration implies solving an additional initial value problem for each integration step and is usually avoided.
- The SoC implementation presented in this work might be significantly less power hungry compared to other implementations, which is crucial for embedded control applications. However, it should be noted that Table 1 gives the *maximum* power consumption values for the listed hardware platforms, which may not correlate with the *actual* power consumption for a given algorithm.
- The proposed implementation can be accelerated further by adopting the idea of early termination of an iterative linear solver, while calculating the step direction in the interior point algorithm. The application of inexact interior point algorithms to optimal control problems was extensively studied in [38]. In contrast to iterative solvers, direct linear solvers do not allow trading off accuracy against computational resources.
- For the considered setup, the FPGA part of the chip is clocked at a lower rate compared the CPU (166 MHz vs 667 MHz). Reconfigurability of FPGAs comes at the price of a reduced clock rate. Hence, there is a potential for further acceleration by replacing the reconfigurable part by a custom non-reconfigurable platform, e.g. an Application-Specific Integrated Circuit (ASIC) with the same architecture.

7 Conclusions

This paper presented the following contributions:

- A heterogeneous implementation of an interior-point-based nonlinear predictive controller and experimental validation of the controller in the loop with a gantry crane model.

- A new release of Protoip, a software tool for quick prototyping of optimization-based controllers on heterogeneous computers. The tool can be used both for testing the proposed controller and for prototyping new algorithms.

The following conclusions can be drawn out of this work:

- Accelerating the linear algebra routines in hardware within a heterogeneous computer implementation can result in a significant speedup over a software-only implementation.
- Performance of a heterogeneous computer-based implementation can be efficiently traded off against resource usage by shifting the computational workload between the CPU and the FPGA, while varying the amount of parallelism for a given part of an algorithm might be less efficient or even without any benefits.
- Offline scheduling for sparse matrix vector multiplication allows avoiding data dependencies and hence building efficient data pipelines, which result in faster implementations.

Further work could be focused on converting the implementation to fixed-point arithmetic in order to achieve better resource usage vs performance trade-offs. The results in [23] demonstrate the efficiency of fixed point arithmetic compared to floating point for the example of a Lanczos algorithm hardware accelerator. Another possible research direction is automating the design process by formulating the NMPC design problem as multi-objective optimization problem in order to identify design trade-offs in a systematic way as in [26]. The list of possible design variables can include both hardware parameters (e.g. parallelization level) and software parameters (e.g. horizon length or number of iterations) parameters.

8 Acknowledgements

This work was funded from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO). The authors also would like to acknowledge the discussions with Mr Junyi Liu, Mr Andrea Zanelli, Dr Rien Quirynen and Dr Milan Vukov, and industrial support from Mathworks.

Appendix A Remaining details for Algorithm 1

$$g(\theta_{[k]}) \coloneqq J\theta_{[k]} - d$$
$$W_{[k]} \coloneqq \Lambda_{[k]} G^{-1}(\theta_{[k]}),$$

where $\Lambda_{[k]}$ and $G(\theta_{[k]})$ are diagonal matrices containing the elements of $\lambda_{[k]}$ and $g(\theta_{[k]})$ respectively.

$$H_{[k]} \coloneqq \nabla_\theta f^T(\theta_{[k]}) \nabla_\theta f(\theta_{[k]}), \quad \mu = -\sigma \lambda_{[k]}^T g(\theta_{[k]})$$

$$r_{dual} \coloneqq \nabla_\theta g^T(\theta_{[k]}) G^{-1}(\theta_{[k]}) \mu e - \nabla_\theta f^T(\theta_{[k]}) \nabla_\theta f(\theta_{[k]}) - \nabla_\theta p^T(\theta_{[k]}) \nu, \quad r_{eq} \coloneqq -p(\theta_{[k]})$$

Appendix B Synthesis results

The synthesis results are presented in Table 2.

Table 2: Algorithm execution times with heterogeneous implementations. For all considered implementations the CPU clock rate is 667 MHz and the FPGA clock frequency is 166 MHz. $T_s = 100$ ms, $n_{iter} = 15$, t is the algorithm execution time.

N	P	SW	HG_1					HG_2					HG_3				
		t, ms	t, ms	LUT	FF	DSP	block RAM	t, ms	LUT	FF	DSP	block RAM	t, ms	LUT	FF	DSP	block RAM
1	1	9	9	2242	3242	8	11	9	6236	8429	20	26	7	8469	11723	43	33
2	1	26	23	2653	3726	10	12	19	6684	8793	20	26	13	8868	12071	43	33
2	2	-	22	2727	3859	10	15	17	6745	8972	23	33	11	8960	12136	43	40
3	1	51	44	2646	3836	10	11	33	6502	8758	20	26	23	8910	12084	43	33
3	3	-	39	3249	4485	15	19	27	7080	9202	26	40	16	9277	12264	43	47
4	1	85	72	2637	3868	10	11	51	6542	8817	20	26	37	8786	12143	43	33
4	4	-	64	3689	5059	20	25	39	7617	9852	31	47	22	9636	12472	43	54
5	1	127	106	2576	3817	10	12	72	6432	8762	20	27	53	8976	12062	43	34
5	5	-	94	4006	5714	25	31	52	8203	10706	36	55	29	10037	12851	43	62
6	1	178	148	2547	3821	10	12	98	6439	8759	20	27	72	8671	12064	43	34
6	6	-	131	4424	6266	30	36	68	8705	1135	41	62	36	10542	13154	46	69
7	1	238	197	2582	3849	10	13	127	6503	8814	20	28	95	9064	12172	43	35
7	7	-	173	4841	6931	35	41	89	9324	12100	46	69	44	11077	13914	51	76
8	1	306	252	2564	3850	10	13	167	6499	8820	20	28	121	8820	12172	43	35
8	8	-	222	5200	7488	40	46	112	9820	12759	51	76	53	11489	14573	56	83
9	1	382	307	2590	3863	10	13	197	6503	8825	20	28	150	9093	12179	43	35
9	9	-	273	5651	8045	45	51	136	10461	13401	56	83	62	12182	15222	61	90
10	1	467	368	2616	3865	10	13	239	6499	8828	20	28	181	8787	12174	43	35
10	10	-	335	6005	8593	50	56	166	10906	14044	61	90	72	12693	15864	66	97

References

[1] Intel Xeon Processor X3440. https://ark.intel.com/products/42928/Intel-Xeon-Processor-X3440-8M-Cache-2.53-GHz. Accessed: 2018-01-27.

[2] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM. doi: 10.1145/1465482.1465560. URL `http://doi.acm.org/10.1145/1465482.1465560`.

[3] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.

[4] H. Ayala, R. Sampaio, D. M. Muñoz, C. Llanos, L. Coelho, and R. Jacobi. Nonlinear model predictive control hardware implementation with custom-precision floating point operations. In *2016 24th Mediterranean Conference on Control and Automation (MED)*, pages 135–140, June 2016. doi: 10.1109/MED.2016.7535908.

[5] J.T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, second edition, 2010.

[6] David Boland and George A Constantinides. An FPGA-based implementation of the MINRES algorithm. In *2008 International Conference on Field Programmable Logic and Applications*, pages 379–384. IEEE, 2008.

[7] David Boland and George A. Constantinides. Optimizing memory bandwidth use and performance for matrix-vector multiplication in iterative methods. *ACM Trans. Reconfigurable Technol. Syst.*, 4(3):22:1–22:14, August 2011. ISSN 1936-7406. doi: 10.1145/2000832.2000834. URL `http://doi.acm.org/10.1145/2000832.2000834`.

[8] William C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991. doi: 10.1137/0801001. URL `http://dx.doi.org/10.1137/0801001`.

[9] F. de Dinechin, M. Joldes, B. Pasca, and G. Revy. Multiplicative square root algorithms for FPGAs. In *2010 International Conference on Field Programmable Logic and Applications*, pages 574–577, Aug 2010. doi: 10.1109/FPL.2010.112.

[10] Frederik Debrouwere, Milan Vukov, Rien Quirynen, Moritz Diehl, and Jan Swevers. Experimental validation of combined nonlinear optimal control and estimation of an overhead crane. *IFAC Proceedings Volumes*, 47(3):9617 – 9622, 2014. ISSN 1474-6670. doi: 10.3182/20140824-6-ZA-1003.01674. 19th IFAC World Congress.

[11] J. Detrey and F. de Dinechin. Floating-point trigonometric functions for FPGAs. In *2007 International Conference on Field Programmable Logic and Applications*, pages 29–34, Aug 2007. doi: 10.1109/FPL.2007.4380621.

[12] Moritz Diehl, Hans Georg Bock, and Johannes P. Schlder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005. doi: 10.1137/S0363012902400713. URL `https://doi.org/10.1137/S0363012902400713`.

[13] *Modular controllers CECX*. Festo, May 2015.

[14] M. Fliess, J. Levine, and P. Rouchon. A simplified approach of crane control via a generalized state-space model. In *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*, pages 736–741 vol.1, Dec 1991. doi: 10.1109/CDC.1991.261409.

[15] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997. doi: 10.1137/1.9781611970937. URL http://epubs.siam.org/doi/abs/10.1137/1.9781611970937.

[16] E.N. Hartley, J.L. Jerez, A. Suardi, Jan M. Maciejowski, E.C. Kerrigan, and G. Constantinides. Predictive Control using an FPGA with Application to Aircraft Control. *IEEE Transactions on Control Systems Technology*, 22(3):1006–1017, May 2014.

[17] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[18] B. Houska, H.J. Ferreau, and M. Diehl. An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range. *Automatica*, 47(10):2279–2285, 2011. doi: 10.1016/j.automatica.2011.08.020.

[19] Inc. ZedBoard (Zynq evaluation and development) hardware user's guide, September 2012.

[20] Dennis Janka, Christian Kirches, Sebastian Sager, and Andreas Wächter. An SR1/BFGS SQP algorithm for nonconvex nonlinear programs with block-diagonal hessian matrix. *Mathematical Programming Computation*, 8(4):435–459, 2016. URL http://dx.doi.org/10.1007/s12532-016-0101-2.

[21] J. L. Jerez, K. V. Ling, G. A. Constantinides, and E. C. Kerrigan. Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry. *IET Control Theory & Applications*, 6:1029–1041(12), May 2012. ISSN 1751-8644.

[22] J. L. Jerez, S. Richter, P. J. Goulart, G. A. Constantinides, E. C. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *Automatic Control, IEEE Transactions on*, 59(12):3238–3251, Dec 2014. ISSN 0018-9286. doi: 10.1109/TAC.2014.2351991.

[23] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. A low complexity scaling method for the lanczos kernel in fixed-point arithmetic. *IEEE Transactions on Computers*, 64(2):303–315, Feb 2015. ISSN 0018-9340. doi: 10.1109/TC.2013.162.

[24] F. A. Khan, Z. Hafeez, A. Mirza, and Q. u. Ain. Design of FPGA based DAQ card using PCI express protocol. In *2011 IEEE 14th International Multitopic Conference*, pages 211–216, Dec 2011. doi: 10.1109/INMIC.2011.6151475.

[25] B. Khusainov, E.C. Kerrigan, A. Suardi, and G.A. Constantinides. Nonlinear predictive control on a heterogeneous computing platform. IFAC World Congress 2017. URL http://hdl.handle.net/10044/1/45094.

[26] B Khusainov, EC Kerrigan, and GA Constantinides. Multi-objective co-design for model predictive control with an FPGA. In *European Control Conference 2016*. IEEE, 2016. URL http://hdl.handle.net/10044/1/30637.

[27] G. Knagge, A. Wills, A. Mills, and B. Ninness. ASIC and FPGA implementation strategies for model predictive control. In *2009 European Control Conference (ECC)*, pages 144–149, Aug 2009.

[28] B. Kpernick and K. Graichen. PLC implementation of a nonlinear model predictive controller. *IFAC Proceedings Volumes*, 47(3):1892 – 1897, 2014. ISSN 1474-6670. 19th IFAC World Congress.

[29] K. V. Ling, S. P. Yue, and J. M. Maciejowski. A FPGA implementation of model predictive control. In *2006 American Control Conference*, pages 6 pp.–, June 2006. doi: 10.1109/ACC.2006.1656502.

[30] J. Lofberg. YALMIP : a toolbox for modeling and optimization in matlab. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pages 284–289, Sept 2004. doi: 10.1109/CACSD.2004.1393890.

[31] *MATLAB version 9.2 (R2017a)*. The Mathworks, Inc., Natick, Massachusetts, 2015.

[32] J. Nocedal, A. Wächter, and R. A. Waltz. Adaptive barrier strategies for nonlinear interior methods. Technical Report RC 23563, IBM Watson Research Center, Yorktown Heights, NY, USA, 2005.

[33] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975. doi: 10.1137/0712047. URL `http://dx.doi.org/10.1137/0712047`.

[34] Helfried Peyrl, Alessandro Zanarini, Thomas Besselmann, Junyi Liu, and Marc-Alexandre Bochat. Parallel implementations of the fast gradient method for high-speed MPC. *Control Engineering Practice*, 33(Supplement C):22 – 34, 2014. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2014.08.010. URL `http://www.sciencedirect.com/science/article/pii/S0967066114002093`.

[35] Helfried Peyrl, Hans Joachim Ferreau, and Dimitris Kouzoupis. A hybrid hardware implementation for nonlinear model predictive control. *IFAC-PapersOnLine*, 48(23):87 – 93, 2015. ISSN 2405-8963. doi: http://dx.doi.org/10.1016/j.ifacol.2015.11.266.

[36] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009. ISBN 9780975937709.

[37] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7 (3):856–869, 1986. doi: 10.1137/0907058. URL `http://dx.doi.org/10.1137/0907058`.

[38] Amir Shahzad, Eric C. Kerrigan, and George A. Constantinides. A stable and efficient method for solving a convex quadratic program with application to optimal control. *SIAM Journal on Optimization*, 22(4):1369–1393, 2012. doi: 10.1137/11082960X. URL `http://dx.doi.org/10.1137/11082960X`.

[39] H. Shukla, B. Khusainov, E.C. Kerrigan, and C.N. Jones. Software and hardware code generation for predictive control using splitting methods. IFAC World Congress 2017. URL `http://hdl.handle.net/10044/1/45093`.

[40] A. Suardi, G. A. Constantinides, and E. C. Kerrigan. Software development kit for FPGA: A fast FPGA prototyping tool for embedded optimization. In *European Control Conference*, 2015.

[41] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare. A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5):1006–1017, Sept 2009. ISSN 1063-6536. doi: 10.1109/TCST.2008.2004503.

[42] M. Vukov, W. Van Loock, B. Houska, H. J. Ferreau, J. Swevers, and M. Diehl. Experimental validation of nonlinear MPC on an overhead crane using automatic code generation. In *2012 American Control Conference (ACC)*, pages 6264–6269, June 2012. doi: 10.1109/ACC.2012.6315390.

[43] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. ISSN 1436-4646. doi: 10.1007/s10107-004-0559-y. URL `http://dx.doi.org/10.1007/s10107-004-0559-y`.

[44] F. Xu, H. Chen, X. Gong, and Q. Mei. Fast nonlinear model predictive control on FPGA using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, 63(1):310–321, Jan 2016. ISSN 0278-0046. doi: 10.1109/TIE.2015.2464171.