Nonlinear predictive control on a heterogeneous computing platform

Bulat Khusainov [*] **Eric C. Kerrigan** [**] **Andrea Suardi** [*]
George A. Constantinides [*]

[*] *Department of Electrical & Electronic Engineering, Imperial College London, London, SW7 2AZ, UK(e-mail: b.khusainov,a.suardi,g.constantinides@imperial.ac.uk.)*
[**] *Department of Electrical & Electronic Engineering and Department of Aeronautics, Imperial College London, London, SW7 2AZ, UK(e-mail: e.kerrigan@imperial.ac.uk).*

Abstract:
Nonlinear Model Predictive Control (NMPC) is an advanced control technique that often relies on computationally demanding optimization and integration algorithms. This paper proposes and investigates a heterogeneous hardware implementation of an NMPC controller based on an interior point algorithm. The proposed implementation provides flexibility of splitting the workload between a general-purpose CPU with a fixed architecture and a field-programmable gate array (FPGA) to trade off contradicting design objectives, namely performance and computational resource usage. A new way of exploiting the structure of the Karush-Kuhn-Tucker (KKT) matrix yields significant memory savings, which is crucial for reconfigurable hardware. For the considered case study, a 10x memory savings compared to existing approaches and a 10x speedup over a software implementation are reported. The proposed implementation can be tested from Matlab using a new release of the Protoip software tool, which is another contribution of the paper. Protoip abstracts many low-level details of heterogeneous hardware programming and allows quick prototyping and processor-in-the-loop verification of heterogeneous hardware implementations.

Keywords: Nonlinear predictive control; Hardware-software co-design; Scheduling

1. INTRODUCTION

Model predictive control (MPC) is an advanced control technique that allows systematic performance optimization, constraint handling and tackling multiple input and multiple output systems (Rawlings and Mayne, 2009). MPC relies on solving optimization problems at every sampling instant, which applies tight constraints on the algorithm execution time and hence limits the application scope to slow plants.

However, recent developments in algorithms for solving online optimization problems allows applying predictive control to systems with relatively fast dynamics (Debrouwere et al., 2014). In addition to improvements on the software side, employing reconfigurable computing platforms, such as as field-programmable gate arrays (FPGAs), resulted in a further speedup of linear MPC algorithms (Jerez et al., 2012; Hartley et al., 2014). Extending a hardware acceleration approach to nonlinear model predictive control (NMPC) is not straightforward, because NMPC involves both optimization and integration, while only the former is known to be efficiently mapped onto reconfigurable platforms. For this reason existing FPGA implementations of NMPC mostly rely on stochastic algorithms (Ayala et al., 2016; Xu et al., 2016), which cannot guarantee local optimality, feasibility and closed-loop stability. 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, Peyrl et al. (2015) present a heterogeneous implementation of a multiple-shooting based NMPC algorithm. The authors propose implementing integration on software while accelerating a fast gradient-based quadratic programming (QP) 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 (Amdahl, 1967), which states that an algorithm's speedup is limited by the part of the workload that cannot benefit from acceleration.

This paper proposes and investigates a heterogeneous implementation of a nonlinear interior point algorithm for predictive control. The main features of the proposed implementation are:

- Flexible splitting of the algorithmic workload between software and hardware for trading off the computational resource usage against performance. A 10x speedup of a heterogeneous implementation compared to a pure software implementation is reported.
- A new way of exploiting the structure of the Karush-Kuhn Tucker (KKT) matrix that allows a significant memory reduction compared to the existing approach

of Boland and Constantinides (2011). For the considered example, a 10x memory saving is reported.

Another contribution of the paper is a new release of the Protoip software tool (Suardi et al., 2015). The tool 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 incorporating both an ARM processor and FPGA. Protoip can be used both for quick testing of the proposed implementation from Matlab and for design and verification of other heterogeneous implementations.

2. OPTIMAL CONTROL PROBLEM FORMULATION

We consider the nonlinear optimal control problem (OCP) with initial state $\hat{x}$ and prediction horizon T:

$$\min_{u,x} \quad \int_0^T \Big(\frac{1}{2}x^T(t)Q_c x(t) + \frac{1}{2}u^T(t)R_c u(t) + x^T(t)S_c u(t)\Big)dt + \frac{1}{2}x^T(T)P_c x(T) \tag{1a}$$

$$\text{subject to:} \quad x(0) = \hat{x}, \tag{1b}$$

$$\dot{x}(t) = f_c(x(t), u(t)), \quad \forall t \in [0, T] \tag{1c}$$

$$Jx(t) + Eu(t) \leq h, \quad \forall t \in [0, T] \tag{1d}$$

$$J_T x(T) \leq h_T, \tag{1e}$$

where $f_c : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$; $Q_c \in \mathbb{S}^n_+$, $R_c \in \mathbb{S}^m_{++}$ and $S_c \in \mathbb{R}^{n\times m}$ are state, input and cross penalty matrices, respectively. $\mathbb{S}^n_{++}(\mathbb{S}^n_+)$ denotes the set of positive (semi-)definite matrices. Path constraints are defined by $J \in \mathbb{R}^{v\times n}, E \in \mathbb{R}^{v\times m}, h \in \mathbb{R}^v$, where v is the number of inequality constraints; the terminal constraint (1e) is defined by J_T and h_T with compatible dimensions. 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. TARGET COMPUTING HARDWARE

The target hardware for the proposed implementation is a Xilinx Zynq-7000 XC7Z020 SoC with dual-core ARM Cortex-A9 and FPGA logic that contains 53200 lookup tables (LUTs), 106400 flip-flops (FFs), 220 DSP blocks and 140 block RAMs with total capacity 4.9 Mb. Communication between software and hardware is performed via an AXI interface.

Programming heterogeneous platforms is not trivial , since the subsystems are often configured using different development environments, which creates problems with communication and library reusage. Modelling languages for programming SoCs, e.g. the Mathworks HDL coder, provide a high level of abstraction, ease of building prototypes and flexibility of moving the workload between subsystems. Unfortunately, these benefits come at the price of efficiency (in terms of resource usage and computation time) of the generated code. An alternative is using C-based tools, e.g. Xilinx SDSoC, that provide a good compromise between code efficiency and design effort. However, in this case, user has to take responsibility for creating testbench files, which often slows down the design process.

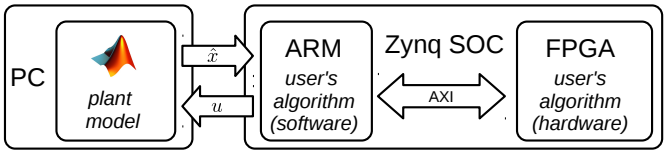

Fig. 1. Processor-in-the-loop test with Protoip.

The new release of Protoip provides infrastructure for quick prototyping and verification of online optimization algorithms using a C-based approach. Using the vendor's software on the underlying levels (Xilinx Vivado, Vivado HLS, SDK) Protoip creates template projects, software verification testbench files and processor-in-the-loop (PIL) testing facilities. A PIL test implies running the optimization algorithm on a hardware platform while simulating the controlled system on a desktop machine, which allows rapid closed-loop performance verification (see Figure 1). The tool is implemented in the Tcl scripting language and is available on the Xilinx Tcl store. A Matlab interface is provided for compatibility with state-of-the-art optimization and control toolboxes.

4. NONLINEAR PREDICTIVE CONTROL ALGORITHMS

Direct solution of the continuous-time optimal control problem (1) 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 (1c) may involve mathematical expressions (e.g. sine and square root) that require significant amounts of computational resources compared to standard addition and multiplication operations.
- Due to the non-regular structure in f_c, reusing computational logic becomes non-trivial.

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 linear algebra algorithms can be efficiently mapped onto hardware (Jerez et al., 2012).

Taking the above into account we consider two classes of algorithms for solving (1): shooting-based and direct transcription algorithms (Betts, 2010). 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 (1) directly to a discrete OCP by approximating the ODE with algebraic equations based on numerical integration equations, i.e.

$$\min_{\substack{u_0\dots u_{N-1}\\ x_0\dots x_N\\ r_0\dots r_{N-1}}} \sum_{k=0}^{N-1}\Big(\frac{1}{2}x_k^T Q_d x_k + \frac{1}{2}u_k^T R_d u_k + x_k^T S_d u_k\Big) + \frac{1}{2}x_N^T P_d x_N \tag{2a}$$

$$\text{subject to:} \quad x_0 = \hat{x}, \tag{2b}$$
$$c(x_{k+1}, x_k, u_k, r_k) = 0, \quad k = 0, \dots, N-1 \tag{2c}$$
$$Jx_k + Eu_k \leq h, \quad k = 0, \dots, N-1 \tag{2d}$$
$$J_T x_N \leq h_T, \tag{2e}$$

where N is the horizon length, $r_k = [r_k^{(1)T} \dots r_k^{(l)T}]^T \in \mathbb{R}^{nl}$ is a vector of integrator intermediate stages and l is the number of stages per sampling instant. Note that discrete-time weight matrices ($Q_d \in \mathbb{R}^{n\times n}, R_d \in \mathbb{R}^{m\times m}, S_d \in \mathbb{R}^{n\times n}, P_d \in \mathbb{R}^{n\times n}$) should be chosen to approximate corresponding continuous-time weights. More details on computing quadrature approximations can be found in Betts (2010).

The OCP (2) can be transformed into an NLP of the form

$$\min_{\theta} \quad \frac{1}{2}\theta^T H\theta + \theta^T h \tag{3a}$$
$$\text{subject to} \quad F_{nl}(\theta) = f, \tag{3b}$$
$$G\theta \leq g, \tag{3c}$$

where $\theta = [x_0^T, r_0^T, u_0^T, \dots, x_{N-1}^T, r_{N-1}^T, u_{N-1}^T, x_N^T]^T$.

NLP (3) incorporates both integration and optimization, which potentially opens the possibility of accelerating both subproblems. Primal-dual interior point algorithms have proven their efficiency for numerical solution of optimal control problems (Shahzad et al., 2012). 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 (Jerez et al., 2012).

The next section introduces a structure-exploiting heterogeneous implementation of an interior point algorithm for solving (3). All results assume explicit Euler integrator, i.e. no intermediate integration steps are used. However, the implementation can be generalized to any Runga-Kutta family integrator by only changing the software part.

5. ALGORITHM AND IMPLEMENTATION DETAILS

5.1 Primal-dual interior-point algorithm

Algorithm 1 and Appendix A outline the steps of primal-dual interior point algorithm for solving (3). The algorithm is based on the QP algorithm in Wright (1997). The only modification is replacing linear equality constraints with nonlinear, which results in an additional Jacobian evaluation step. 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 following structure:

$$A = \begin{bmatrix}
0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-I & Q_0^* & S_0^* & A_0^{*T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & S_0^{*T} & R_0^* & B_0^{*T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & A_0^* & B_0^* & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -I & Q_1^* & S_1^* & A_1^{*T} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & S_1^{*T} & R_1^* & B_1^{*T} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & A_1^* & B_1^* & 0 & -I & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -I & Q_2^* & S_2^* & A_2^{*T} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & S_2^{*T} & R_2^* & B_2^{*T} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & A_2^* & B_2^* & 0 & -I \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & P^*
\end{bmatrix} \tag{4}$$

Algorithm 1 Primal-dual interior point method for NLP

Initial point $(\theta_{[0]}, \nu_{[0]}, \lambda_{[0]}, s_{[0]}) : (\lambda_{[0]}, s_{[0]}) \geq 0$, reduction parameter: $0 \leq \sigma \leq 1$
for $k = 0$ to n_{iter} **do**
  Evaluate $\nabla Fnl(\theta_{[k]})$
  $A_{[k]} = \begin{bmatrix} H + G^T W_{[k]} G & \nabla F_{nl}^T(\theta_{[k]}) \\ \nabla F_{nl}(\theta_{[k]}) & 0 \end{bmatrix}$
  $b_{[k]} = \begin{bmatrix} r_{dual} \\ r_{eq} \end{bmatrix}$
  Solve $A_{[k]} z_{[k]} = b_{[k]}$ for $z_{[k]} = \begin{bmatrix} \Delta\theta_{[k]} \\ \Delta\nu_{[k]} \end{bmatrix}$
  $\Delta\lambda_{[k]} = W_{[k]}(G(\theta_{[k]} + \Delta\theta_{[k]}) - g) + \sigma\mu s_{[k]}^{-1}$
  $\Delta s_{[k]} = -s_{[k]} - (G(\theta_{[k]} + \Delta\theta_{[k]}) - g)$
  $\alpha_{[k]} = \max_{(0,1]}\alpha : (\lambda_{[k]}^T, s_{[k]}^T) + \alpha(\Delta\lambda_{[k]}^T, \Delta s_{[k]}^T) > 0$
  $(\theta_{[k+1]}^T, \nu_{[k+1]}^T, \lambda_{[k+1]}^T, s_{[k+1]}^T)^T =$
  $\alpha_{[k]}(\Delta\theta_{[k]}^T, \Delta\nu_{[k]}^T, \Delta\lambda_{[k]}^T, \Delta s_{[k]}^T)^T + (\theta_{[k]}^T, \nu_{[k]}^T, \lambda_{[k]}^T, s_{[k]}^T)^T$
end for

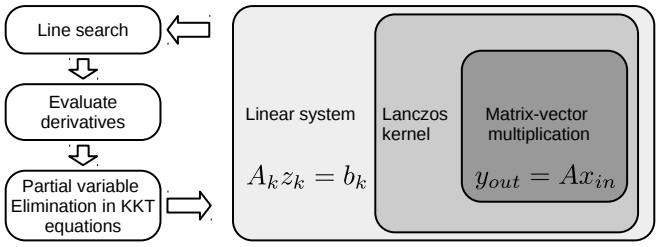

Fig. 2. Algorithm 1 flow diagram with MINRES solver.

where

$$\begin{bmatrix} Q_i^* & S_i^* \\ S_i^{*T} & R_i^* \end{bmatrix} = \begin{bmatrix} Q_d & S_d \\ S_d^T & R_d \end{bmatrix} + [J \ E]^T W_i^{part} [J \ E], \tag{5a}$$
$$P^* = P_d + J_T^T W_N^{part} J_T \tag{5b}$$

and W_i^{part} is a diagonal matrix containing corresponding diagonal elements of $W_{[k]}$. A_i^* and B_i^* correspond to Jacobians of the equality constraints, i.e.

$$A_i^* = \nabla_{x_i} c(x_{i+1}, x_i, u_i, r_i), \tag{6}$$
$$B_i^* = \nabla_{u_i} c(x_{i+1}, x_i, u_i, r_i). \tag{7}$$

A symmetric system of linear equations can be solved using direct methods, e.g. LDL decomposition, and iterative methods, e.g. MINRES. 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 (Boland and Constantinides, 2008) and is well studied in relation to optimal control applications (Shahzad et al., 2012).

5.2 Proposed implementation

Assuming that Algorithm 1 is implemented using the MINRES linear solver it can be visualized with the block diagram in Figure 2. We consider four different ways of splitting the workload between software and hardware:

- The entire algorithm is implemented in software.
- Only the matrix-vector multiplication is accelerated in hardware and the rest is implemented in software.
- The whole Lanczos kernel is accelerated in hardware and the rest is implemented in software.
- The whole linear system solver is accelerated in hardware and the rest is implemented in software.

Note that for all considered options, the Jacobians are evaluated 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 (Jerez et al., 2012). 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 that, as will be seen in the case study in Appendix B, can lead to a 10x saving in memory.

Matrix-vector multiplication can potentially be parallelized by simultaneous processing of the blocks in (4) highlighted by solid lines. However, consecutive blocks are coupled via negative identity matrices, i.e. each block accesses areas in the input vector associated with its neighbours, which applies restrictions on memory partitioning. For matrix-vector multiplication, handling negative identity matrices is reduced to data transfer operations with changing the sign bit. 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

$$\underbrace{\begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}}_{\text{output vector}} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & 0 & a_{23} & 0 \\ 0 & a_{32} & 0 & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix}}_{\text{sparse matrix}} \underbrace{\begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}}_{\text{input vector}}. \quad (8)$$

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. The order of processing non-zero elements does not affect the final result, however, it does affect 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

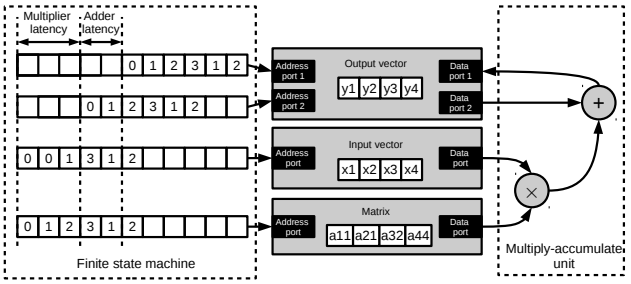

Fig. 3. MAC circuit for the example in (8).

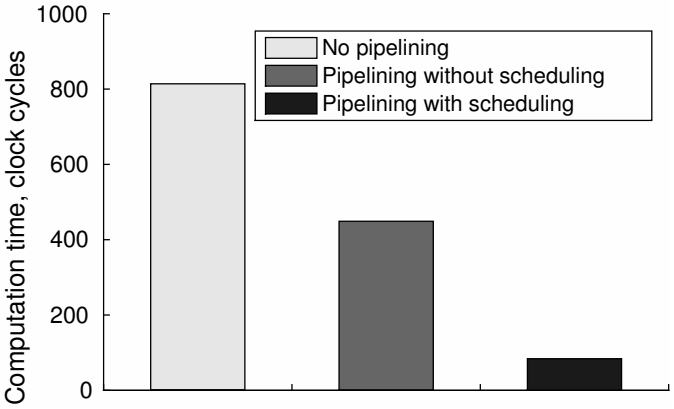

Fig. 4. The impact of scheduling on computation time of a sparse matrix-vector multiplication (for each gray block in (4)). It is assumed that the latency for the addition and multiplication units are 6 and 5 clock cycles, respectively. The number of nonzero elements is 74, which is the case for the example in Appendix B.

be formulated as an optimization problem

$$\max_{d, S_{sched}} \quad d \quad (9a)$$

subject to:

$$|t(a_{ij}) - t(a_{kl})| > d \quad \forall a_{ij} \in M, \forall a_{kl} \in M : i = k, j \neq l \quad (9b)$$

$$1 \leq t(a_{ij}) \leq N_{nz} \quad \forall a_{ij} \in M \quad (9c)$$

$$t(a_{ij}) \neq t(a_{kl}) \quad \iff i \neq k, j \neq l \quad (9d)$$

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 queue number of a_{ij} 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 (4) have the same sparsity pattern, the scheduling problem is solved only once. The resulting MAC circuit for the considered example (8) is shown in Figure 3. 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 4. For this case study the problem (9) was solved using the YALMIP built-in branch and bound solver (Lofberg, 2004). If pipelining is implemented without scheduling, the initiation interval is equal to the adder latency, which limits potential improvement. With a systematic scheduling approach, read-write dependencies are avoided, which allows one to start processing a new non-zero element every clock cycle, i.e. achieving an initiation interval of one.

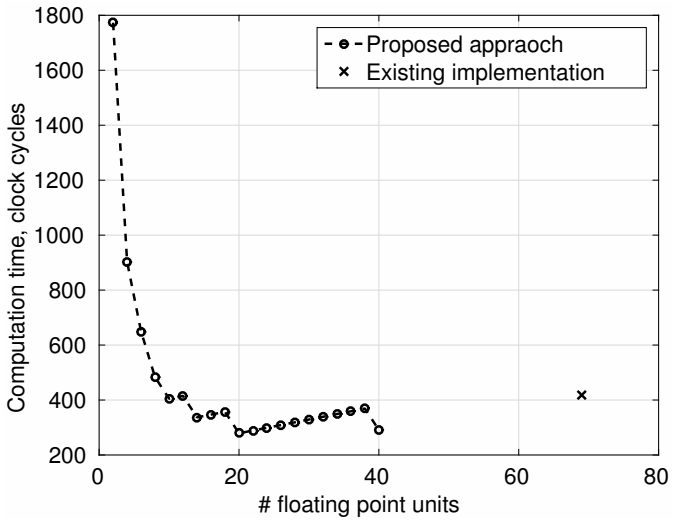

Fig. 5. Trading off computational time against processor resource usage for matrix-vector multiplication (4). Each circle represents a design with a certain degree of parallelism. The floating point unit is either floating point addition or multiplication. For the considered test case, $N = 20$ and the ODE model is given in Appendix B. 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 Jerez et al. (2012).

Although matrix-vector multiplication (4) can be parallelized by allocating a MAC unit to each grey block, we propose trading off computation time against resource usage by varying the number of MAC units, as shown in Figure 5.

6. SYNTHESIS AND SIMULATION 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 100 MHz, respectively. The benchmark plant model and OCP parameters are given in Appendix B. For the considered example, the pure software implementations were not able to compute optimal solution within one sampling instant $T_s = 150$ ms and $n_{iter} = 15$, as can be seen in Table 1. Speeding up certain parts of the computational workload on the FPGA and allocating two MAC units for matrix-vector multiplication resulted in a reduced computational time and hence implementing the algorithm online.

The trade-off between the closed-loop performance and processor resource usage for the feasible designs from Table 1 is illustrated in Figure 6. We measure closed-loop performance based on a PIL simulation with the cost function

$$V(\mathrm{u},\mathrm{x}) = \sum_{k=0}^{N_{sim}-1} \left(\frac{1}{2}\mathrm{x}_k^T Q_d \mathrm{x}_k + \frac{1}{2}\mathrm{u}_k^T R_d \mathrm{u}_k + \mathrm{x}_k^T S_d \mathrm{u}_k\right), \quad (10)$$

where u_k is an input to the plant and x_k is a state of the plant at sampling instance k. The number of simulation samples N_{sim} is chosen to reflect convergence or divergence of the controlled plant. Resource usage

Table 1. Algorithm execution times with heterogeneous implementations. Feasible designs are highlighted with a gray background.

N	Software, ms	Mat-vec multiplication on FPGA, ms	Lanczos kernel on FPGA, ms	Linear solver on FPGA, ms
2	416	144	78	48
3	600	227	125	68
4	1043	373	195	109
5	1316	502	265	135
6	2033	695	366	191

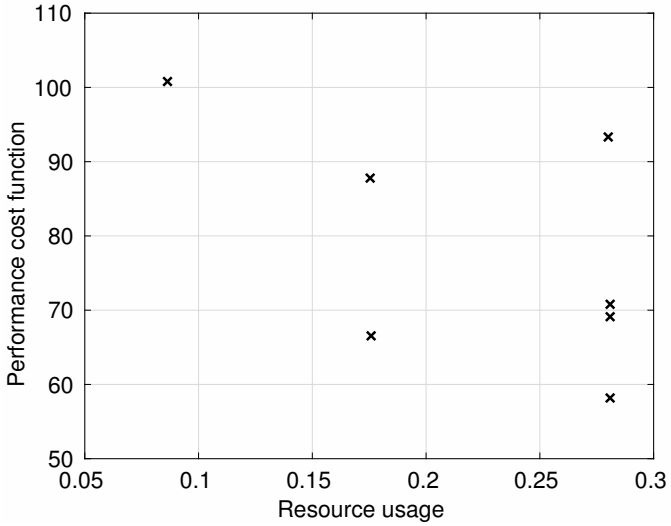

Fig. 6. Trading off closed-loop performance against resource usage by splitting the workload between software and hardware.

is measured as geometric mean of relative utilization of each resource type, namely LUT, FF, block RAM and DSP. Three out of seven considered designs are Pareto-optimal. The notable point is that Pareto-optimal designs are achieved by splitting the workload between software and hardware in different ways, which justifies the flexibility of the software-hardware splitting of the proposed implementation.

7. CONCLUSIONS

This paper presented a heterogeneous implementation of an interior point-based nonlinear predictive controller. Accelerating the linear algebra routines in hardware resulted in a significant speedup over a software-only implementation. Moreover, it was shown that performance can be efficiently traded off against resource usage by shifting the computational workload between the CPU and the FPGA.

Further work will be focused on 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 Khusainov et al. (2016). The list of possible design variables can include both hardware (e.g. parallelization level) and software (e.g. horizon length or number of iterations) parameters.

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).

REFERENCES

Amdahl, G.M. (1967). 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), 483–485. ACM, New York, NY, USA. doi:10.1145/1465482.1465560. URL http://doi.acm.org/10.1145/1465482.1465560.

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

Betts, J. (2010). *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, second edition.

Boland, D. and Constantinides, G.A. (2008). An FPGA-based implementation of the minres algorithm. In *2008 International Conference on Field Programmable Logic and Applications*, 379–384. IEEE.

Boland, D. and Constantinides, G.A. (2011). Optimizing memory bandwidth use and performance for matrix-vector multiplication in iterative methods. *ACM Trans. Reconfigurable Technol. Syst.*, 4(3), 22:1–22:14. doi:10.1145/2000832.2000834. URL http://doi.acm.org/10.1145/2000832.2000834.

Debrouwere, F., Vukov, M., Quirynen, R., Diehl, M., and Swevers, J. (2014). Experimental validation of combined nonlinear optimal control and estimation of an overhead crane. *IFAC Proceedings Volumes*, 47(3), 9617 – 9622. doi:10.3182/20140824-6-ZA-1003.01674. 19th IFAC World Congress.

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

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

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

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

Peyrl, H., Ferreau, H.J., and Kouzoupis, D. (2015). A hybrid hardware implementation for nonlinear model predictive control. *IFAC-PapersOnLine*, 48(23), 87 – 93. doi:http://dx.doi.org/10.1016/j.ifacol.2015.11.266.

Rawlings, J. and Mayne, D. (2009). *Model Predictive Control: Theory and Design*. Nob Hill Pub.

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

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

Wright, S. (1997). Applying new optimization algorithms to model predictive control. In *Fifth International Conference on Chemical Process Control – CPC V*, 147–155. CACHE Publications.

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

Appendix A. REMAINING DETAILS FOR ALGORITHM 1

$$\mu_{[k]} \coloneqq \frac{\lambda_{[k]}^T s_{[k]}}{\text{length}(s_{[k]})}, \quad W_{[k]} \coloneqq \Lambda_{[k]} S_{[k]}^{-1},$$

$$r_{eq} \coloneqq -(F_{nl}(\theta_{[k]}) - f), r_{dual} \coloneqq -(H + G^T W_{[k]} G)\theta_{[k]} - \\ - h - \nabla F_{nl}^T(\theta_{[k]})\nu_{[k]} - G^T(\lambda_{[k]} - W_{[k]} g + \sigma\mu_{[k]} s_{[k]}^{-1})$$

where $\Lambda_{[k]}$ and $S_{[k]}$ are diagonal matrices containing the elements of $\lambda_{[k]}$ and $s_{[k]}$ respectively.

Appendix B. GANTRY CRANE MODEL AND OCP PARAMETERS

$$\begin{aligned}
\dot{x}(1) &= x(2),\ \ \dot{x}(3) = x(4),\ \ \dot{x}(5) = x(6),\ \ \dot{x}(7) = x(8) \\
\dot{x}(2) &= -\left(0.22\,\sin(x(5))\,x(6)^2 + 15.85\,u(1) - 44.17\,x(2)\right. \\
&\quad \left. + 4.61\cos(x(5))\,\sin(x(5))\right)/\left(0.47\cos(x(5))^2 - 3.84\right) \\
\dot{x}(4) &= -\left(0.22\,\sin(x(7))\,x(8)^2 + 15.85\,u(2) - 43.02\,x(4)\right. \\
&\quad \left. + 4.61\cos(x(7))\,\sin(x(7))\right)/\left(0.47\cos(x(7))^2 - 2.13\right) \\
\dot{x}(6) &= \left(0.22\,\cos(x(5))\,\sin(x(5))\,x(6)^2 + \right. \\
&\quad + 37.67\sin(x(5)) + 15.85u(1)\cos(x(5)) - \\
&\quad \left. - 44.17\,x(2)\cos(x(5))\right)/\left(0.22\cos(x(5))^2 - 1.80\right) \\
\dot{x}(8) &= \left(0.22\,\cos(x(7))\,\sin(x(7))\,x(8)^2 + \right. \\
&\quad + 20.89\sin(x(7)) + 15.85u(2)\cos(x(7)) - \\
&\quad \left. - 43.02\,x(4)\cos(x(7))\right)/\left(0.22\cos(x(7))^2 - 1.00\right)
\end{aligned}$$

$$\hat{x} = [0,0,0,0,\pi/3,0,\pi/4,0]^T, S_d = 0_{n,m}$$
$$Q_d = \text{diag}(10,0,10,0,10,0,10,0), P_d = Q_d$$
$$R_d = \text{diag}(0.01, 0.01), J = 0_{v,n}, \text{where } v = 4$$
$$E = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}^T, d = [2\ 2\ 2\ 2]^T$$