Multi-objective Co-design for Model Predictive Control with an FPGA

B. Khusainov[1], E. C. Kerrigan[2] and G. A. Constantinides[1]

Abstract— **In order to achieve the best possible performance of a model predictive controller (MPC) for a given set of resources, the software algorithm and computational platform have to be designed simultaneously. Moreover, in practical applications the controller design problem has a multi-objective nature: performance is traded off against computational hardware resource usage, namely time, energy and space. This paper proposes formulating an MPC design problem as a multi-objective optimization (MOO) problem in order to explore the design trade-offs in a systematic way.**

Since the design objectives in the resulting MOO problem are expensive to evaluate, i.e. evaluation requires time consuming simulations, most of the classical and evolutionary MOO algorithms cannot be employed for this class of design problems. For this reason a practical MOO algorithm that can deal with expensive-to-evaluate functions is presented. The algorithm is based on Kriging and the hypervolume criterion that was recently proposed in the expensive optimization literature. A numerical example for a fast gradient-based controller design shows that the proposed approach can efficiently explore optimal performance-resource trade-offs.

I. Performance vs. time, energy and space

Model predictive control (MPC) has already proven to be an efficient and reliable solution for a wide-range of applications, most of which are characterized by relatively slow dynamics. The necessity of solving an optimization problem at each sampling instance has traditionally been the main bottleneck that prevented more significant expansion of MPC, especially in relation to fast dynamic plants. Recent developments in communication technologies and hardware processing systems have sped up online optimization solvers and hence increased the potential application scope of MPC.

Most MPC and underlying optimization algorithms are designed at a high level of abstraction without regard to the intended hardware platform. This decoupled approach usually leads to inefficient utilization of available resources [1]. In contrast, the co-design approach implies simultaneous design of both software and hardware components in order to achieve the best possible performance for a given set of available resources. However, improvement of the closed-loop performance cannot be considered as the only design objective. In practice, there is a trade-off between performance and computational hardware resource usage. Resources that are required to perform computations include time, energy and space, which are also the functions of the MPC algorithm and hardware platform. Instead of looking for one optimal design (which often does not exist due to conflicts between objectives) engineers might make a decision based on the whole series of Pareto optimal designs, i.e. designs that cannot be improved in terms of one objective without worsening at least one of the other objectives.

The problem of investigating design trade-offs is usually formalized as a multi-objective optimization (MOO) problem. The main bottleneck that prevents efficient solution of MOO design problems in relation to MPC are the properties of the objective functions. For instance, it is difficult to derive a closed-form expression and hence obtain derivative information for evaluation of an MPC closed-loop cost function or hardware platform energy consumption. Even evaluation of these black-box functions usually requires time-consuming simulations. Functions with such properties are often called *expensive functions* and, analogously, optimization problems that involve expensive objective or constraint functions belong to the class of *expensive optimization* problems. Expensive optimization algorithms aim to find optimal points with a limited number of evaluations, which is usually done by modeling expensive functions based on evaluated points. In this paper, we propose an algorithmic solution for the generation of Pareto optimal MPC designs, which is based on a hypervolume criterion described in [2]. In contrast to generic implementation [3], the proposed algorithm speeds up the solution of expensive MOO problems by (i) parallelizing the evaluations and (ii) decoupling the objective function and constraint models, so that objective functions are not evaluated for infeasible designs. This is the first application of a systematic MOO method for co-design of an MPC algorithm and underlying computational platform.

II. Optimal control problem formulation

Consider the linear time-invariant state-space model:

$$x_{k+1} = Ax_k + Bu_k, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $x_k \in \mathbb{R}^n$ is a vector of state variables and $u_k \in \mathbb{R}^m$ is an input vector. The constrained optimal control problem with horizon length N for the system (1) is described by

*The work leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no 607957 (TEMPO).

[1]Department of Electrical & Electronic Engineering, Imperial College London, SW7 2AZ London, UK. b.khusainov@imperial.ac.uk, g.constantinides@imperial.ac.uk

[2]Department of Electrical & Electronic Engineering and Department of Aeronautics, Imperial College London, SW7 2AZ London, UK. e.kerrigan@imperial.ac.uk

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

subject to

$$x_0 = \hat{x} \tag{2b}$$

$$x_{k+1} = Ax_k + Bu_k, \quad \text{for } k = 0, 1, \dots, N-1 \tag{2c}$$

$$Jx_k + Fu_k \leq h, \quad \text{for } k = 0, 1, \dots, N-1, \tag{2d}$$

where $Q_d \in \mathbb{S}_+^n$, $R_d \in \mathbb{S}_{++}^m$ and $P_d \in \mathbb{S}_+^n$ are state, input and terminal penalty matrices accordingly and $\hat{x}$ is the current state estimate. $\mathbb{S}_{++}^n (\mathbb{S}_+^n)$ denotes a set of positive (semi-)definite matrices. Constraints are defined by $J \in \mathbb{R}^{v \times n}, F \in \mathbb{R}^{v \times m}$ and $h \in \mathbb{R}^v$, where v is the number of constraints.

III. Design objectives

We assume that the optimal control problem (2) is formulated as a quadratic programming (QP) problem and a QP solver (e.g. fast gradient, interior-point) is to be implemented on a field-programmable gate array (FPGA). The main reason behind choosing an FPGA as a target platform is customizability. Many hardware parameters, such as data representation, parallelization level or clock frequency can be easily varied on reconfigurable platforms in order to estimate closed-loop performance and computational resource usage.

FPGA design flow involves three main stages:

1) *High-level synthesis (HLS)*: Converting the C/C++ code with synthesis directives (e.g. loop unrolling or pipelining) into low level hardware description language (HDL) code.
2) *Synthesis*: The process of transforming HDL code into a netlist, which shows the connection of all logic gates and registers.
3) *Place-and-route (P&R)*: Solving a set of optimization problems in order to fit the netlist to a particular FPGA platform. The outcome of P&R is a bitstream that can be uploaded onto the FPGA.

At each stage of the design flow the objectives, which are functions of the software p_s and hardware p_h design parameter vectors, can be estimated:

1) *Closed-loop performance*. Since an FPGA typically uses a data representation that is different from standard double precision floating point, separate software-in-the-loop (SIL) and/or hardware in-the-loop (HIL) tests have to be conducted in order to verify the performance. SIL requires the HLS stage to be finished, while HIL can be performed only after complete circuit synthesis. We measure the closed-loop cost function in the following way:

$$V(\mathrm{u},\mathrm{x}) = \sum_{k=0}^{N_{sim}-1} (\frac{1}{2} \mathrm{x}_k^T Q_d \mathrm{x}_k + \frac{1}{2} \mathrm{u}_k^T R_d \mathrm{u}_k), \tag{3}$$

where u_k is the input to the plant and x_k is the state of the plant at sampling instance k. The number of simulation samples N_{sim} has to be chosen to reflect convergence or divergence of the controlled plant. A set of simulations with different initial conditions might be required in order to obtain a reliable estimate of the performance.
2) *Time*. Timing information can be obtained at the HLS stage, since the number of algorithm clock cycles is completely defined by the registers in the circuit. FPGA design considers algorithm execution time as a constraint, rather than objective, since the synthesized circuit is intended for one particular application. In contrast, for CPU programming, where several programs share the same hardware, minimizing time is an important objective.
3) *Space*. An FPGA designer often aims to minimize the amount of silicon that is required for implementing a particular control algorithm. An FPGA has the following resources: flip-flops, lookup tables, block RAMs and DSPs. We measure the silicon usage (or space usage) as the average relative utilization of each resource type, assuming uniform utilization of all resources. Estimation of this function can be obtained at any stage of the design flow with different accuracies.
4) *Energy consumption*. There is seldom a linear correspondence between circuit size and energy consumption. For instance, in some cases it could be energy efficient to create a large circuit by parallelizing all computations in order to quickly perform all calculations and switch to idle mode. Power consumption estimation can be obtained at the P&R stage.

The trade-off between the above discussed design objectives can be formalized by formulating the MOO problem:

$$\min_{p} \quad f(p) \coloneqq \begin{bmatrix} f_1(p) \\ \vdots \\ f_l(p) \end{bmatrix} \tag{4a}$$

$$\text{subject to} \quad p \in S, \tag{4b}$$

where $f(p)$ is an l-dimensional vector of design objectives and S is a q-dimensional feasible decision space. Note that for software and hardware co-design problems parameter vector involves both software and hardware design variables $p = [p_s^T \ p_h^T]^T$.

The objective functions of the MOO problem (4) have the following properties:

1) Absence of derivative information. There is no analytical expressions for accurate estimation of the design objectives.
2) Long function evaluation time. Each evaluation of the listed functions requires time-consuming simulations to be performed.
3) Noisiness. The same HLS code may result in different resource usage values depending on a vendor's software version or other factors that are not taken into account by conventional models.

Tackling the MOO problem (4) with the above described properties is discussed in the next section.

IV. Multi-objective problem definition and solution

A. Notations and definitions

In the context of problem (4), a point $y' = f(p') : p' \in S$ is said to dominate $y'' = f(p'') : p'' \in S$ iff $\forall i \in \{1, \dots, l\} : y'_i \leq y''_i$ and $y' \neq y''$, where $y'_i = f_i(p')$ and $y''_i = f_i(p'')$. The shorthand notation for this is $y' \prec y''$. Analogously, $y' \preceq y''$ means $\forall i \in \{1, \dots, l\} : y'_i \leq y''_i$.

The Pareto frontier is the set of all feasible non-dominated, i.e. Pareto optimal, points. $P(U)$ is the set of feasible non-dominated points for a given set of evaluated points U, i.e. the current approximation of the Pareto frontier.

B. Handling objective functions in (4)

The MOO problem can be scalarized, i.e. transformed into a single-objective problem, so that the solution of the scalarized problem is a single point on the Pareto front. The classical example of scalarizing is aggregating the objectives into a single function, although more sophisticated techniques might be used (e.g. [4]). This class of algorithms does not exploit similarities between consecutive single-objective problems, which results in increased number of expensive evaluations. Moreover, the resulting single-objective optimization problems may have unpleasant properties, due to the fact that expensive-to-evaluate objectives of the original MOO often appear in the constraints of scalarized problems.

An alternative way of tackling (4) is approximating the Pareto frontier directly. The review paper [5] outlines two classes of derivative-free algorithms for direct approximation of the Pareto frontier: direct multisearch methods (DMS), which is a multi-objective generalization of direct search algorithms, and evolutionary algorithms. Unfortunately, these approaches are not suitable for MPC design problems, since they typically require a relatively large amount of evaluations to converge.

In order to reduce the number of evaluations, [6] proposed a systematic way of building and updating a *surrogate model* of the original function. The main idea is to model the function using Gaussian process regression (Kriging) with a limited number of sampling points and to update the model by sampling new points with the highest *expected improvement* (EI). For a single function optimization expected improvement is defined as the expectation of the objective function decrease after sampling the certain new point. Maximizing the EI allows automatic balancing between exploration (improving the accuracy of unexplored regions) and exploitation (improving the regions that are likely to contain an optimal point).

For MOO the Pareto frontier approximation quality, and hence improvement, is often measured by means of the *hypervolume*. The hypervolume is defined as a set of points in the objective space $\{y \preceq y^{ref} \in \mathbb{R}^l : \exists y' \in P : y' \preceq y\}$, where y^{ref} is selected by a designer. In other words, a hypervolume is a set of points that (i) are dominated by at least one of the evaluated points and (ii) dominate the reference point. The Lebesgue measure of a hypervolume $L(P, y^{ref})$ defines the quality of the Pareto frontier approximation (Figure 1). The improvement in the hypervolume after sampling a new point y^{new} is given by:

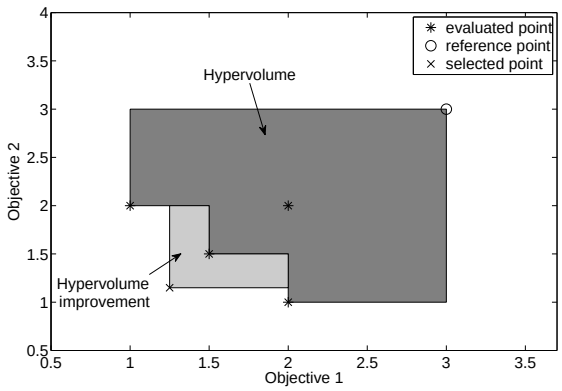

Fig. 1. Hypervolume space as a measure of the Pareto frontier approximation quality. Dark grey is the current hypervolume, light grey is the improvement after evaluating a new point.

$$I(P, y^{new}) \coloneqq L(P \cup \{y^{new}\}, y^{ref}) - L(P, y^{ref}). \quad (5)$$

Based on (5), the expectation of improvement after evaluating a new point p is

$$\mathrm{EI}(p) = \int_{y \in \mathbb{R}^l} I(P, y) \mathrm{PDF}_p(y) dy, \quad (6)$$

where PDF is the probability density function of the objectives that comes from the Kriging model. A calculation procedure for computing EI (6) for a given Kriging model is explained in [2].

Similarly to the vast majority of interpolation methods, Kriging models the approximation as a weighted sum of the values surrounding approximated point. The weights are assigned based on a *covariance function*, which is typically a decreasing function of the distance between two points. A detailed discussion on choosing and tuning the covariance function is presented in [7].

The important distinctive feature of Kriging is the calculation of estimation error (variance σ) alongside with the estimation itself (expectation E). An example of a Kriging model for FPGA resource usage is presented in Figure 2. The 95% confidence interval was calculated under assumption of a normal distribution based on the variance: $\mathrm{E} \pm 1.96\sqrt{\sigma}$.

Calculation of expectation and variance for a given scalar function $g(\cdot)$ is performed in the following way. Assume there is a vector of evaluated points $z = [z_1 \dots z_r]^T$, where $z_i = g(p_i)$. In order to approximate the function value z_* at p_*, covariance matrices have to be constructed:

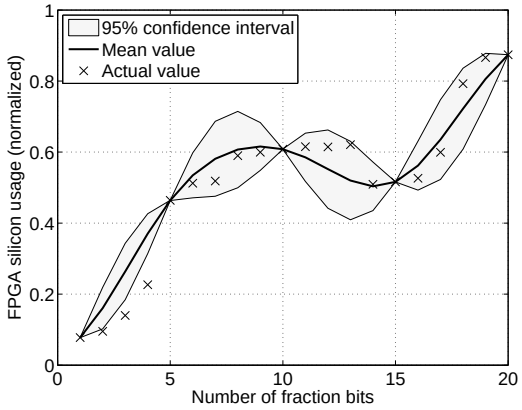

Fig. 2. Kriging model: FPGA silicon usage as a function of number of fraction bits for data representation.

$$
\begin{aligned}
K &\coloneqq \begin{bmatrix} \operatorname{cov}(p_1,p_1) & \cdots & \operatorname{cov}(p_1,p_r) \\ \vdots & \ddots & \vdots \\ \operatorname{cov}(p_r,p_1) & \cdots & \operatorname{cov}(p_r,p_r) \end{bmatrix}, \\
K_* &\coloneqq \begin{bmatrix} \operatorname{cov}(p_*,p_1) & \cdots & \operatorname{cov}(p_*,p_r) \end{bmatrix}, \\
K_{**} &\coloneqq \begin{bmatrix} \operatorname{cov}(p_*,p_*) \end{bmatrix},
\end{aligned} \tag{7}
$$

where cov is covariance function. Based on covariance matrices, expectation and variance can be calculated:

$$
\begin{aligned}
\mathrm{E}(z_*) &= K_* K^{-1} z, \\
\sigma(z_*) &= K_{**} - K_* K^{-1} K_*^T.
\end{aligned} \tag{8}
$$

The described procedure applies only for zero mean functions, but can be generalized for other functoins as described in [7]. We suggest using dedicated software packages for modelling, e.g. GPML, which is based on [7].

After the initial model is built, the expression for computing the EI can be derived. Following this, the point p^* with the highest EI is selected and evaluation of the objective functions is performed. The whole process is repeated until the algorithm converges (Algorithm 1). Note that a global derivative-free optimizer (e.g. the Nelder-Mead algorithm [8] or simulated annealing) is employed for maximizing $\mathrm{EI}(p)$.

Algorithm 1 evaluates only one point at each iteration, while a designer might have computational facilities for parallel evaluation of a number of objective functions. In order to parallelize the algorithm we propose introducing the inner loop to Algorithm 1, which results in Algorithm 2. At each inner loop iteration the model is updated under the assumption that the actual value of the function is equal to the mathematical expectation $\mathrm{E}(\cdot)$ regardless of the standard deviation. After the required number of points $N_{parallel}$ is obtained, expensive evaluation can be performed in parallel in order to update the model with actual values of the objective functions.

C. Handling constraints in (4)

In relation to MPC controller design, the feasible set S for the MOO problem (4) can often be described by the following constraints:

Algorithm 1: Generic hypervolume-based unconstrained expensive MOO algorithm.

```
Evaluate objective functions at initial points;
do
    Update Kriging model;
    Identify P;
    Derive EI(p);
    p* = arg max_p EI(p);
    Evaluate new point p*;
while EI(p*) > ε;
```

Algorithm 2: Parallel implementation of hypervolume-based unconstrained expensive MOO algorithm.

```
Evaluate objective functions at initial points;
do
    Update Kriging model;
    for i = 1 : N_parallel do
        Identify P;
        Derive EI(p);
        p*(i) = arg max_p EI(p);
        Update Kriging model under assumption
        f(p*(i)) = E(f(p*(i)));
    end
    Evaluate new point(s) {p*(1), ..., p*(N_parallel)} in
    parallel;
while EI(p*(1)) > ε;
```

- *Bounds on design parameters*. The upper and lower bounds on design variables define the search space. Obviously these are cheap analytical constraints.
- *Upper bounds on design objectives.* Controller designer usually aims to stay within some bounds on performance and resource usage. This requirement can be satisfied by setting the appropriate coordinate of the reference point y^{ref} to an upper bound of the corresponding objective function.
- *Latency constraint.* The fundamental constraint of any online optimization algorithm is an upper bound on execution time.

Among the listed constraints, the only one that requires separate consideration is the latency constraint. When an online optimization algorithm for MPC is coded using an HLS tool, latency information can be obtained only after performing high-level synthesis. Hence the latency constraint is an *expensive constraint*. There are at least two ways of dealing with expensive constraints presented in the literature: (i) incorporating expensive constraints into the objective function [9] and (ii) using dedicated models known as support vector machines (SVM) [10]. The first approach can distort the objective function model, since the original covariance function may not be suitable for augmented objective. Using complicated support vector machines is not justified for handling one single constraint and may complicate the overall MPC design algorithm.

Algorithm 3: Constrained parallel implementation of hypervolume-based expensive MOO algorithm.

1 Evaluate objective functions at initial points;
2 **do**
3 Update Kriging model;
4 **for** $i = 1 : N_{parallel}$ **do**
5 Identify P;
6 Derive EI(p);
7 **do**
8 Update feasibility model S_{model};
9 $p^*(i) = \arg\max_p \text{EI}(p)$ s.t. $p \in S_{model}$;
10 Evaluate feasibility;
11 **while** $p^*(i) \notin S$;
12 Update Kriging model under assumption $f(p^*(i)) = \text{E}(f(p^*(i)))$;
13 **end**
14 Evaluate new point(s) $\{p^*(1), \ldots, p^*(N_{parallel})\}$ in parallel;
15 **while** $\text{EI}(p^*(1)) > \epsilon$;

We propose building a separate feasibility model based on Gaussian process classification. For classification problems, instead of estimating the function value, the model estimates the probability of belonging to a certain class. Obviously, if for a certain point p the probability of being feasible is higher than the probability of being infeasible, the point is assumed to be feasible. This approach has two benefits from an implementation point of view. Firstly, since feasibility and objective functions models are completely decoupled, they can be updated independently. For example, it is possible to update the feasibility model with HLS data, while objective functions can be evaluated with more accurate and time-consuming full circuit synthesis information. Secondly, since the nature of the Gaussian process classification problem is similar to regression, often the same software tools can be used for classification and regression.

The overall constrained MOO procedure is summarized in Algorithm 3. After initial points are evaluated, both objective function and feasibility models are updated. Following this, the point for consecutive evaluation is derived taking into account the feasibility model S_{model}. After that, feasibility information is obtained by simulation in order to verify the information provided by the classifier. If the derived point p^* is infeasible, the feasibility model is updated in the inner loop until a feasible point cannot be computed. Objective functions are evaluated on the last step of the loop when points with proven feasibility are obtained.

V. Experimental results

We solve the optimal control problem (2) in condensed form using Nesterov's fast gradient method (FGM) with a constant step size scheme [11]. The necessity of calculating the projection makes FGM practically suitable only for input constrained systems. The algorithm was implemented with fixed-point arithmetic on Artix-7 programmable logic of a Xilinx Zynq-7020 system-on-a-chip (SoC). Fixed-point implementation of FGM for MPC is discussed in [12].

A mass-spring-damper system model with 10 states and 5 inputs was employed to investigate the design trade-offs. For the presented examples, HLS was used to identify design feasibility while full circuit synthesis was performed to get silicon usage and power consumption information. Closed-loop performance was assessed by means of HIL simulation using FPGA prototyping software [13].

We consider a co-design problem with two design variables:

- Horizon length N. The range from 1 to 10 was selected for evaluation.
- Number of fraction bits b for fixed point data representation. The range from 1 to 20 was selected for evaluation. The number of integer bits was chosen to avoid overflow errors and kept constant.

Simulation results show that the cost function (3) monotonically decreases with respect to both horizon length and number of fraction bits. It was also observed that a certain desired closed-loop performance can be achieved using different combinations of software and hardware design variables. For instance, the design $N = 2$ and $b = 1$ leads to similar closed-loop performance as the design $N = 1$ and $b = 4$ in terms of closed-loop cost function (3). However, both parameters affect FPGA silicon usage as well. According to the report from the synthesis tool the former design is 1.5 times more efficient in terms of FPGA silicon utilization. We choose the closed-loop performance and FPGA silicon usage to be the contradicting design objectives for the MOO problem. The MOO algorithm aims to identify the designs that provide the best trade-off between performance and FPGA silicon usage.

Before implementing the algorithm, a full design exploration was performed in order to identify the true set of Pareto optimal designs. Obviously, enumerating all possible combinations is time consuming and not scalable. For this relatively small example, exploration of 200 designs took approximately 54 hours and identified all 29 Pareto optimal designs. The explored designs are presented in Figure 3. Implementing Algorithm 2 with $N_{parallel} = 2$ allowed significant reduction of expensive function evaluations; after an initial sampling of 9 points, which was performed on the bounds and middle points of the parameter space, it took 26 evaluations to get a good approximation of the Pareto frontier (Figure 3).

The impact of parallelizing the evaluations on the convergence speed of Algorithm 2 is presented in Figure 4. Three implementations with different levels of parallelization are compared in terms of the hypervolume. The set of points for initial evaluation (line 1 of the algorithm) is the same for all considered cases. It could be observed that parallelizing the evaluations, i.e. evaluating $N_{parallel}$ points simultaneously during 1 time unit, improves the rate of convergence. However, since in the inner loop the model is updated without actual evaluations (line 8) the quality of the model decays with increasing $N_{parallel}$. This explains the

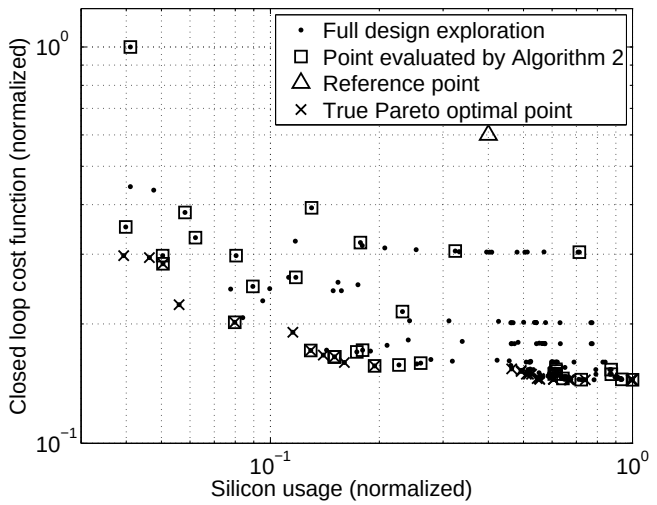

Fig. 3. The design space for the MOO problem with two objectives.

fact that improvement from $N_{parallel} = 1$ to $N_{parallel} = 2$ is higher compared to improvement from $N_{parallel} = 2$ to $N_{parallel} = 3$.

After Algorithm 2 was verified, constrained implementation (Algorithm 3) was tested. For the considered example, it was observed that for up to 30% of the outer loop iterations (Algorithm 3) the feasibility model required updates within the inner loop. This means that the algorithm requires up to 30% less expensive objective functions evaluations compared to the approach when constraints are incorporated in objective functions.

The main implementation challenge for the proposed algorithm is choosing an appropriate covariance function for the Kriging. For the considered application it was required to explore a small region of the design space in order to derive a suitable covariance function.

VI. Conclusions and future work

Multi-objective co-design allows (i) efficient exploitation of the computational hardware platform capabilities and (ii) systematically treating the design trade-offs. This paper proposed a practical hypervolume criterion-based algorithm for MPC multi-objective co-design. Experimental results show that using this approach allows avoiding a large number of non-Pareto optimal and infeasible MPC designs.

Future work includes exploiting the coupling between the nature of the design problem and MOO algorithm. For instance, closed-loop cost function properties, which were recently studied in [14], can be used for improving the accuracy of the regression model and hence algorithm convergence speed. Another direction for further research is deriving analytical bounds for objective functions in order to reduce the evaluation time, i.e. the time of a single iteration.

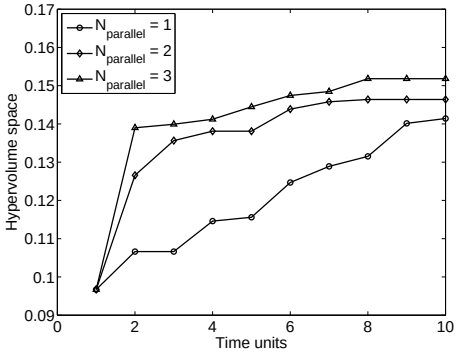

Fig. 4. The effect of parallelization in Algorithm 2.

References

[1] E. C. Kerrigan, "Co-design of hardware and algorithms for real-time optimization," in *Control Conference (ECC), 2014 European*, June 2014, pp. 2484–2489.

[2] A. Emmerich, "The computation of the expected improvement in dominated hypervolume of Pareto front approximations," Leiden University, The Netherlands, Tech. Rep. LIACS TR-4-2008, 2008.

[3] M. Tesch, J. Schneider, and H. Choset, "Expensive multiobjective optimization for robotics," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 973–980.

[4] G. Kirlik and S. Sayn, "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems," *European Journal of Operational Research*, vol. 232, no. 3, pp. 479 – 488, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/ S0377221713006474

[5] A. Custodio, M. Emmerich, and J. Madeira, "Recent developments in derivative-free multiobjective optimisation," *Computational Technology Reviews*, vol. 5, pp. 1 – 30, 2012.

[6] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[7] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[8] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-free Optimization*, ser. MOS-SIAM series on optimization. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.

[9] K. Holmstrm, "Tomlab – an environment for solving optimization problems in matlab," in *Proceedings for the Nordic MATLAB conference*, 1997, pp. 27–28.

[10] G. Gao, C. Sun, J. Zeng, and S. Xue, "A constraint approximation assisted pso for computationally expensive constrained problems," in *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, June 2014, pp. 1354–1359.

[11] Y. Nesterov, *Introductory lectures on convex optimization : a basic course*, ser. Applied optimization. Boston, Dordrecht, London: Kluwer Academic Publ., 2004. [Online]. Available: http://opac.inria.fr/record=b1104789

[12] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *CoRR*, vol. abs/1303.1090, 2013. [Online]. Available: http://arxiv.org/abs/1303.1090

[13] A. Suardi, E. C. Kerrigan, and G. A. Constantinides, "Fast FPGA prototyping toolbox for embedded optimization," in *Control Conference (ECC), 2015 European*. IEEE, 2015, pp. 2589–2594.

[14] V. Bachtiar, E. C. Kerrigan, W. H. Moase, and C. Manzie, "Continuity and monotonicity of the MPC value function with respect to sampling time and prediction horizon," *Automatica*, vol. 63, pp. 330 – 337, 2016.