IFAC Proceedings Volumes, Volume 45, Issue 17, 2012, Pages 60-67
4th IFAC Conference on Nonlinear Model Predictive Control 2012, (NMPC'12),
23-27 August 2012, Leeuwenhorst in Noordwijkerhout, The Netherlands.

Number Representation in Predictive Control

Eric C. Kerrigan * **Juan L. Jerez** ** **Stefano Longo** **
George A. Constantinides **

* *Department of Electrical & Electronic Engineering and Department of Aeronautics, Imperial College London, Exhibition Road, London SW7 2AZ, UK (e-mail: e.kerrigan@ imperial.ac.uk).*
** *Department of Electrical and Electronic Engineering, Imperial College London, Exhibition Road, London SW7 2AZ, UK (e-mail: juan.jerez-fullana05,s.longo,g.constantinides@imperial.ac.uk)*

Abstract: In predictive control a nonlinear optimization problem has to be solved at each sample instant. Solving this optimization problem in a computationally efficient and numerically reliable fashion on an embedded system is a challenging task. This paper presents results to reduce the computational requirements for solving fundamental problems that arise when implementing predictive controllers in finite precision arithmetic. By employing novel formulations and tailor-made optimization algorithms, this paper shows that computational resources can be reduced using very low precision arithmetic. We also present new mathematical results that enable computational savings to be made in the most numerically critical part of an optimization solver, namely the linear algebra kernel, using fixed-point arithmetic. Our theoretical results are supported by numerical results from implementations on a Field Programmable Gate Array (FPGA).

Keywords: Predictive control; Optimization problems; Number systems; Numerical methods; Embedded systems

1. INTRODUCTION

This paper is concerned with the important problem of choosing the right number system when implementing a predictive controller on an embedded control system, e.g. as in automotive, aerospace, electrical power, healthcare, communications, manufacturing and motion control applications. The number representation has a big impact on the computational efficiency and performance of the optimization solver that is used within the predictive control scheme. Decreasing the number of bits needed to do the computations, or by changing from floating-point to fixed-point, can significantly reduce the time, silicon area, cost and energy needed to compute the control action. However, numerical errors could potentially offset computational performance gains and result in poor closed-loop performance, hence a good understanding of number representation is crucial for the successful implementation of predictive controllers in demanding and safety-critical control applications.

A major driving factor in deciding what to include on a microprocessor is the area of the chip. Static and dynamic power consumption of a chip roughly grows at least linearly with area, ignoring the additional increase in power consumption due to increases in interconnection length. Furthermore, because of defects during manufacturing, not all dies manufactured are acceptable; a good rule of thumb for modern manufacturing processes is that the cost per processor die grows roughly with the *square* of the die area [Hennessy and Patterson, 2011, Chap. 1.6].

One of the key choices that an engineer has to make in order to control the area of a chip is the number representation that will be used for the arithmetic units [Koren, 2002]. For fixed-point arithmetic with bit parallel two's complement, the area of multipliers typically scales quadratically and the area of adders scales linearly with the number of bits. For floating-point arithmetic, the area of multipliers and adders usually scale somewhere between linearly and quadratically with the number of bits, since the area of the shifter in the floating-point adder scales somewhere between linearly and quadratically with the number of bits. It is clear that significant savings in cost and power requirements are possible by appropriate choice of number representation and reducing the number of bits. Of course, similar improvements in latency (time taken to complete a computation, or commonly referred to as the computational delay) and throughput (number of completed computations per unit time) are also possible by reducing the number of bits. However, the amount of improvement is implementation dependent and it is often possible to trade off area against latency or throughput.

Nearly all CPUs within modern desktop PCs provide hardware support for the IEEE-754 standard for double

⋆ This work was supported by EPSRC grants EP/G031576/1 and EP/I012036/1 and the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement number FP7-ICT-2009-4 248940, as well as industrial support from Xilinx, The Mathworks and the European Space Agency.

Published by Elsevier. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial No Derivatives License (CC:BY:NC:ND 4.0).
The final published version (version of record) is available online at DOI:10.3182/20120823-5-NL-3013.00017. Please refer to any applicable publisher terms of use.

precision floating-point, which uses 52 bits for the mantissa (significand), 11 bits for the exponent and one bit for the sign. However, most microprocessors in embedded systems do not offer any support for double precision floating-point. Instead, they may only offer floating-point support for single precision (23 bits for the mantissa and 8 bits for the exponent) or half precision (10 bits for the mantissa and 5 bits for the exponent) or not even provide any support for floating-point at all. It is therefore possible that, because of a significant decrease in precision, a predictive control algorithm that gives reliable numerical results when implemented in the office desktop computer or laptop might give completely different results when implemented on the real embedded control system.

Because energy consumption and cost are two major concerns in embedded control systems, most microprocessors provide very good support for fixed-point arithmetic. This is because a fixed-point arithmetic unit takes up significantly less area on a chip than a floating-point unit with the same number of bits, therefore resulting in significant savings in manufacturing cost and energy requirements. Fixed-point units are also faster than floating-point units with the same number of bits, due to the fact that the radix points (binary points) of two numbers do not need to be aligned during addition or subtraction. These reductions in cost, computation time and energy come at the price of a significant reduction in the dynamic range (ratio of the largest representable number to the smallest non-zero representable number) of numbers that can be represented using fixed-point compared to floating-point. As a consequence, a predictive control algorithm that was numerically well-behaved when implemented using floating-point arithmetic might produce a significant number of numerical errors, such as overflow or underflow, when implemented using fixed-point arithmetic.

Since the choice of number representation can have a dramatic effect on the numerical robustness of a predictive controller implemented in an embedded control system, it is crucial to have a good understanding of the effect that the choice of number system and precision of the computations has on overall system performance. Rather than relying on heuristic methods for addressing this issue, it is important to develop systematic and mathematically rigorous methods that allow the implementation of predictive controllers on current and future embedded control systems with guarantees on stability and system performance. A sound theoretical basis with supporting development tools will enable the transfer of predictive control technology to current and future applications where speed, cost or energy requirements are too demanding for implementations based on double precision floating-point, thereby necessitating a switch to a different number representation.

This paper gives an overview of some recent work done at Imperial College London that aims to develop a deep understanding of the fundamental issues that are important in numerical representation in predictive control. We show that existing formulations and algorithms for solving predictive control problems result in very poor performance when a low precision or fixed-point implementation is used. We summarize some new systematic methods and theoretical results that we have developed for addressing some of these problems. These results allow for predictive controllers to be implemented in embedded control systems with dramatic reductions in cost, computation time and energy requirements, compared to existing methods.

We present results and methods that are generic and that can be applied, in principle, to most existing processors used in embedded control systems, including DSPs and embedded GPUs. However, one of the main challenges when doing fundamental research on numerical representation is to have a sufficiently flexible platform for generating numerical results in order to compare against theoretical predictions. Most microprocessors only allow one to choose from a predetermined set of number systems, which is not ideal for fundamental research in this area. It is possible to emulate different number representations in software (e.g. with the GNU MP [1] and MPFR [2] libraries for arbitrary-precision arithmetic or the MATLAB [3] Fixed-Point Toolbox) as an initial step in evaluating the effect of a given number representation on algorithm behavior. However, a software emulation approach makes it difficult to quantify how the computational resources would scale in practice if a processor does not yet exist that would provide hardware support for a non-standard number representation.

Fortunately, current Field Programmable Gate Arrays (FPGAs), which are already extensively used in a number of embedded control systems, are sufficiently flexible to allow a researcher to explore a variety of number representations, yet powerful enough to implement predictive controllers for challenging control applications. In an FPGA one can choose to implement floating-point, fixed-point or any other number representation, such as logarithmic, residue or custom number systems, with arbitrary precision. The reconfigurability of an FPGA allows experimentation with different number representations in order to get a better understanding of the trade-offs that has to be made in practice between closed-loop performance, computational hardware resources, accuracy and precision. It is for this reason that some of the numerical results presented in this paper include those from implementations on an FPGA.

Section 2 of this paper is concerned with the case when the predictive control implementation uses a very small number of bits. The main point made here is that the choice of discretization method is a key factor in the behavior of a predictive control algorithm implemented in low precision. The method commonly used in predictive control applications to compute an equivalent discrete-time model of the continuous-time system, namely the shift form, can be numerically sensitive to round-off error; an off-the-shelf optimization algorithm would not be able to detect and correct for any errors in the data, because information is lost prior to solving the optimization problem. We propose a novel formulation of the optimization problem, based on the delta operator approach of Middleton and Goodwin [1986], which allows the computation of an equivalent discrete-time model that is less susceptible to numerical errors compared to the shift form. In order to solve this optimization problem, we also outline a novel algorithm

[1] http://gmplib.org/
[2] http://www.mpfr.org/
[3] http://www.mathworks.com/

that pays attention to the order in which computations should be done in order to minimize the effect of numerical errors and avoid any increase in computational resources or latency. This section is based on results in Longo et al. [2012] and the reader is referred to that paper for a more in-depth discussion.

Section 3 is based on results in Jerez et al. [2012a,b]. and discusses what happens when implementing the most computationally expensive and numerically critical part of an optimization algorithm, namely the computation of the search direction, using a fixed-point representation. In particular, we discuss the use of the minimal residual (MINRES) method for solving the resulting set of linear equations and implement the Lanczos kernel using fixed-point arithmetic. The main challenge of fixed-point over floating-point is to develop a computationally tractable method to a priori determine tight bounds on the dynamic range of the variables, which is not possible using existing methods. This section presents a novel preconditioner that guarantees that the eigenvalues of the preconditioned matrix has eigenvalues inside the unit disk. This allows us to analytically derive tight bounds on all the variables in the Lanczos process, in order to determine a priori where to place the radix point (binary point) such that numerical errors due to overflow are avoided.

The paper draws some conclusions in Section 4.

2. LOW PRECISION ARITHMETIC

Double- or single-precision floating-point representation may be unnecessarily precise for a given application, where precision would be better traded in for improving more important aspects, such as speed, cost and energy consumption. Reducing the number of mantissa bits can significantly reduce the hardware resources required and the latency of a predictive control algorithm, as can be seen in Figure 1, which shows the resources required for an implementation of the predictive control algorithm of Rao et al. [1998] on an FPGA.

The problem with reducing the number of bits is that existing predictive control algorithms may give unacceptable results when using a very low precision. Figure 2 shows that the closed-loop response of a 5-bit implementation of Rao et al. [1998], which uses the shift form to obtain a discrete-time model, may give an unacceptable response compared to a double precision implementation of Rao et al. [1998]. However, Figure 2 also shows that a 5-bit implementation based on solving the optimization problem outlined below, which is based on obtaining a discrete-time model using the delta operator approach of Middleton and Goodwin [1986], produces trajectories that almost perfectly overlap with the ones from a 52-bit shift implementation.

The reason for the problem with using the shift form, which is the common method used to discretize the continuous-time system in existing predictive control formulations, is easily understood. Consider the continuous-time LTI plant model

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{1}$$

where $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$. Suppose the input signal $u(\cdot)$ is piecewise constant, e.g. when there is a zero-

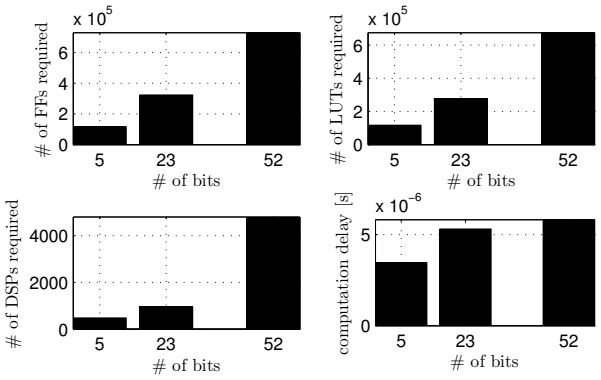

Fig. 1. FPGA resources required for double (52 bits), single (23 bits) and custom (5 bits) floating-point implementations for a predictive control algorithm with 6 states, 2 inputs and a horizon length of 200 steps. The number of Flip-Flops (FF), Look-Up-Tables (LUTs), Digital Signal Processing (DSP) units required and the latency (calculated at a clock frequency of 200 MHz, assuming 10 interior-point iterations) are given.

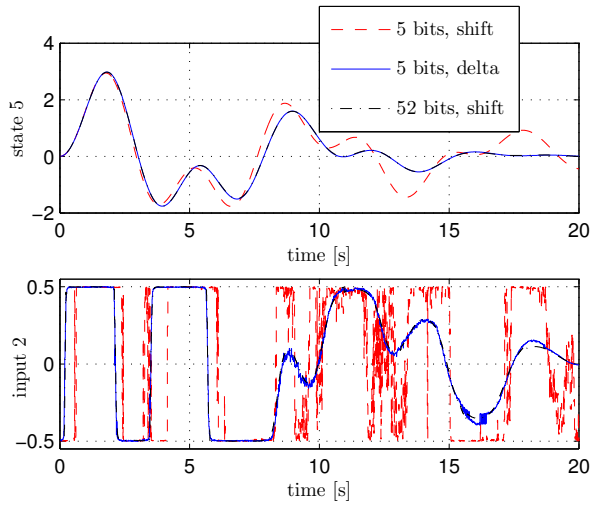

Fig. 2. Sample state and input trajectories of the closed-loop response of a benchmark spring-mass system with 6 states, 2 inputs, horizon length of 2 s and sample period of 10 ms.

order hold at the output of the controller, with a sampling period h such that

$$u(t) = u(ih), \quad \forall t \in [ih, ih + h),$$

where $i \in \mathbb{N}_0$ is the sample instant. We will adopt the more convenient notation $x_i := x(ih)$ and similarly for other vectors.

The equivalent discrete-time/sampled-data model in the so-called *shift* form (as implemented in, for example, MATLAB's `c2d` function) is given by

$$x_{i+1} = \underbrace{e^{A_c h}}_{=:A_s} x_i + \underbrace{\left[\int_0^h e^{A_c(h-\tau)} B_c d\tau\right]}_{=:B_s} u_i, \tag{2}$$

where, following from the definition of the matrix exponential, the exponential of the matrix $A_c h$ is given by

$$e^{A_c h} = I + A_c h + \frac{(A_c h)^2}{2!} + \frac{(A_c h)^3}{3!} + \dots. \quad (3)$$

It is important to sit back here and note the well-known (but not often appreciated) fact that

$$\lim_{h\to 0} A_s = I \quad \text{and} \quad \lim_{h\to 0} B_s = 0.$$

This is not nice, because it implies that the discrete-time matrices do not converge to the continuous-time matrices as the sample period tends to zero. This is not just an interesting theoretical curiosity, but has very important implications for implementation of predictive controllers in finite precision arithmetic.

The above equations suggest that significant problems might occur in practice when implementing a controller using finite precision arithmetic. If the product $A_c h$ in (3) results in a matrix with entries much smaller than one, then the transition matrix A_s in (2) will be a matrix where the elements on the diagonal are the summation of 1 with a much smaller number, hence the significant bits of the coefficients will be truncated and some of the information contained in $A_c h + \frac{(A_c h)^2}{2!} + \frac{(A_c h)^3}{3!} + \dots$, which is where the plant dynamics are represented, might be lost.

In practice, a system does not need to be sampled at very fast rates before truncation or round-off errors become a problem when using a small number of bits to represent the data. Sample times of the order of 10 to 20 times less than the dominant time constant of the system are often sufficient for numerical problems to arise, and sampling slower than this is usually only done if computation time is an issue.

2.1 An Alternative Discrete-Time Representation

One way to address the above numerical problem is to use an alternative representation for the discrete-time model. A very good candidate is the *delta* operator approach [Goodwin et al., 2010, Middleton and Goodwin, 1986]. Consider substituting (3) into (2) and rewriting it as

$$\frac{x_{i+1} - x_i}{h} = A_\delta x_i + B_\delta u_i, \quad (4a)$$

where

$$A_\delta := \Omega A_c, \quad B_\delta := \Omega B_c, \quad (4b)$$

and

$$\Omega := \frac{1}{h}\int_0^h e^{A_c \tau} d\tau = I + \frac{A_c h}{2!} + \frac{A_c^2 h^2}{3!} + \dots \quad (4c)$$

$$= \frac{1}{h}\begin{bmatrix} I & 0 \end{bmatrix} \exp\left(\begin{bmatrix} A_c & I \\ 0 & 0 \end{bmatrix} h\right) \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (4d)$$

Clearly, (4a) is mathematically equivalent to (2). However, this observation can be used to reduce the numerical errors due to finite precision effects. What is nice about the delta form is that the discrete-time plant matrices converge to their continuous-time counterparts, i.e.

$$\lim_{h\to 0} A_\delta = A_c \quad \text{and} \quad \lim_{h\to 0} B_\delta = B_c.$$

This is a very useful property to have when implementing predictive controllers in finite precision arithmetic.

2.2 A New Formulation of the Constrained LQR Problem

Suppose that the problem is to find an input and state sequence that minimizes a positive semidefinite quadratic stage and terminal cost on the states and inputs, subject to satisfying the model constraints and given linear inequality constraints on the state and input over a finite horizon, as is standard in predictive control formulations based on the constrained LQR problem [Scokaert and Rawlings, 1998]. The question is whether it is possible to formulate and solve an optimization problem that utilizes the delta form, rather than the shift form, in a smart way in order to minimize effects due to finite precision arithmetic.

One possible solution is to start by introducing an additional variable $\delta_i \in \mathbb{R}^{n_x}$ and rewrite (4a) equivalently as

$$\delta_i := A_\delta x_i + B_\delta u_i, \quad (5a)$$

$$x_{i+1} = x_i + h\delta_i. \quad (5b)$$

In order to preserve the information contained in A_δ and B_δ, the important point to note is that the decision variables in the optimization problem should not only include the input and state sequences as decision variables, as in existing predictive control formulations [Rao et al., 1998], but also the sequence of additional variables $\{\delta_0, \delta_1, \dots\}$. The plots in Figure 2 for the 5-bit shift representation are a clear illustration of the unintended consequences of not including the additional variables $\{\delta_0, \delta_1, \dots\}$ in the formulation of the QP.

One could therefore proceed, as proposed in Longo et al. [2012], by defining the decision variables for the optimization problem as

$$\theta := \begin{bmatrix} u_0' & \delta_0' & x_1' & u_1' & \delta_1' & x_2' & \cdots & u_{N-1}' & \delta_{N-1}' & x_N' \end{bmatrix}', \quad (6a)$$

and formulate the optimal control problem in the form

$$\min_\theta \frac{1}{2} x_N' Q_f x_N + \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}' \begin{bmatrix} Q & M \\ M' & R \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \quad (6b)$$

$$\text{s.t. } x_0 = \hat{x}, \quad (6c)$$

$$\delta_k = A_\delta x_k + B_\delta u_k, \quad \forall k \in \{0, 1, \dots, N-1\} \quad (6d)$$

$$x_{k+1} = x_k + h\delta_k, \quad \forall k \in \{0, 1, \dots, N-1\} \quad (6e)$$

$$Jx_k + Eu_k \le d, \quad \forall k \in \{0, 1, \dots, N-1\} \quad (6f)$$

where N is the number of samples in the horizon, $\hat{x}$ is a given measurement or estimate of the current state, all matrices and vectors have compatible dimensions and the cost is convex.

2.3 Solving the New Constrained LQR Problem (6)

It is straightforward to show that (6) can be written as a convex QP of the form

$$\min_\theta \frac{1}{2}\theta' H \theta \quad (7a)$$

$$\text{s.t. } F\theta = f,\ G\theta \le g, \quad (7b)$$

where the matrices F, G and H are sparse and banded.

A naive approach to solving the QP (7) would be to arbitrarily choose an existing off-the-shelf QP solver code, which has support for sparse matrices, and compile it for a target embedded system with low precision arithmetic. This approach may fail early on in the implementation phase. For example, the preprocessor of the QP solver might want to first eliminate the additional variables $\{\delta_i\}_{i=0}^{N-1}$ to produce a smaller QP, thus destroying some of the important information contained in A_δ and B_δ. Of course, the information contained in A_δ and B_δ could also

be destroyed at any stage in the QP solver, depending on the details of the algorithm. On the other hand, an existing solver may appear to work in exhaustive simulations, but won't come with any rigorous theoretical guarantees on the accuracy of the solution, hence making it difficult to certify for safety-critical applications.

It could therefore be potentially disastrous to proceed with a low precision implementation of a QP solver and not care about the details of the algorithm or the hardware. An important research goal is to develop optimization algorithms with a guarantee that the loss of accuracy due to finite precision effects is minimized or bounded a priori, thereby enabling implementations using very low precision arithmetic with guarantees on the accuracy of the solution.

Another research question is whether the introduction of the additional variables $\{\delta_i\}_{i=0}^{N-1}$ in the QP significantly increases the computational resources required such that any potential gains due to working in low precision are lost, compared to solving a QP with only the inputs and states as the decision variables using the shift form with a higher precision. It is possible to show that, provided care is taken in the development of the QP solver and its implementation, this is not the case. We briefly outline the arguments, which can be found in more detail in Longo et al. [2012], in the remainder of this section.

Consider proceeding along similar lines to the algorithm in Rao et al. [1998], which is based on the interior point algorithm of Mehrotra [1992]. The calculation of the search direction is the most computationally expensive and numerically sensitive part of the algorithm, hence we present only the ideas crucial to this. Suppose the variables in the reduced KKT system $\mathsf{A}\xi = \mathsf{b}$, where ξ is the search direction, are interleaved in a fashion similar to Rao et al. [1998], i.e.

$$\xi := \begin{bmatrix}\Delta u_0' & \Delta\gamma_0' & \Delta\delta_0' & \Delta\lambda_1' & \Delta x_1' & \Delta u_1' & \Delta\gamma_1' & \Delta\delta_1' & \cdots & \Delta x_N'\end{bmatrix}',$$

where $\Delta\gamma_i$ and $\Delta\lambda_i$ are the search directions for the Lagrange multipliers. By performing a sequence of block eliminations on the KKT system, it is possible to show the search direction can be computed via a Riccati recursion similar to Rao et al. [1998]. Figure 2 shows some numerical results for a 5-bit delta implementation, which nearly overlaps with the results from a 52-bit shift implementation.

Although more multiplications and additions are required for the delta formulation, compared to the shift formulation of Rao et al. [1998], the critical path (the longest non-parallelizable sequence of operations) remains unchanged. As an example, let us compare the discrete-time Riccati difference equation (DRDE) of the delta formulation

$$\begin{aligned}P_{k-1} &:= Q_{k-1} + P_k + h^2A_\delta' P_k A_\delta + hA_\delta' P_k + hP_kA_\delta \\ &- (M_{k-1} + h^2A_\delta' P_k B_\delta + hP_kB_\delta)(R_{k-1} + h^2B_\delta' P_k B_\delta)^{-1} \\ &\qquad (M_{k-1}' + h^2B_\delta' P_k A_\delta + hB_\delta' P_k),\end{aligned}$$

with the equivalent DRDE of the shift formulation

$$\begin{aligned}P_{k-1} &:= Q_{k-1} + A_s' P_k A_s + (M_{k-1} + A_s' P_k B_s) \\ &\qquad (R_{k-1} + B_s P_k A_s)^{-1}(M_{k-1}' + B_s' P_k A_s).\end{aligned}$$

The data dependencies of the calculations to compute P_{k-1} are shown as a graph in Figure 3. For the delta case, there are four extra matrix additions to be performed, shown in Figure 3 by the boxes with dotted

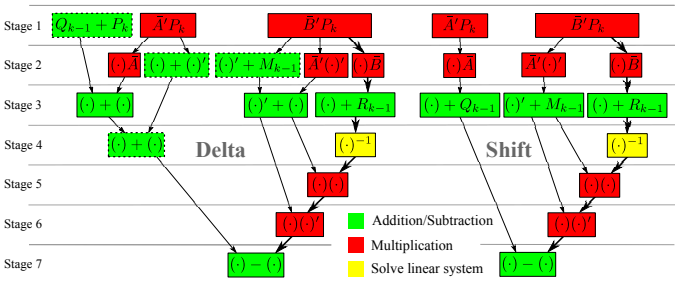

Fig. 3. Data dependencies of the DRDE for the delta and shift formulations. For delta, $\bar{A} := hA_\delta$ and $\bar{B} := hB_\delta$. For shift, $\bar{A} := A_s$ and $\bar{B} := B_s$. Both algorithms have the same critical path (thicker arrows).

edges. Assuming a fully parallel implementation, a maximum of three matrix adders and three matrix multipliers are used simultaneously at each stage of the algorithm; this is true for both the delta and shift case, implying that the same number of hardware blocks are needed for both implementations (of course, in the delta case, the adders will be used more often). Hence, although four extra additions are needed for the delta case, the number of computational resources needed for minimal latency is the same in both cases. Similar arguments hold for the other computations involved in determining the search direction.

Of course, without knowing the details of the target processor, compiler or programming language, it may be difficult to tell whether it is possible to do a parallel implementation of an algorithm for the delta formulation as above, which is guaranteed to have the same latency or use as many resources as an equivalent algorithm for the shift formulation. However, with advances in computing technologies, this might be possible in the near future. Fortunately, the future is already here in the form of FPGAs, where it is possible to have exact control over the sequence in which computations are performed and the resources that they require. Hence, it is possible to use an FPGA to solve a QP with the delta formulation using the same amount of computational resources as would be required to solve a QP with the shift formulation.

When designing new algorithms for predictive controllers implemented in low precision arithmetic, the first point to take home from this section is that it is important to be careful with the formulation of the QP so that important information is not lost prior to solving the QP. The second point is that the sequence in which computations are done might be critical to minimizing numerical errors while solving the QP, e.g. it might be important to add numbers of a similar magnitude together before adding the result to numbers with a larger magnitude. The third point is that, for a computationally efficient and numerically reliable implementation, the details of the embedded computing platform and development tools should inform the details of the algorithm and vice versa.

3. FIXED-POINT ARITHMETIC

As mentioned in the introduction, porting part or all of an algorithm from floating-point to fixed-point can reduce both the hardware resource requirements and arithmetic delays. As can be seen in Table 3, it is possible to

Table 1. Resources required and latency (in clock cycles) for a single floating- or fixed-point adder on a Xilinx Virtex-7 XT 1140 FPGA.

Number Representation	Registers	LUTs	Latency
double float (52-bit mantissa)	1046	911	14
single float (23-bit mantissa)	557	477	11
53-bit fixed-point	53	53	1
24-bit fixed-point	24	24	1

decrease the latency of an addition by roughly one order of magnitude, with an even bigger relative reduction in the resources required, by switching from floating-point to fixed-point in an FPGA. Of course, the relative reductions in resources and latency are different for other arithmetic units and for other architectures, but the general argument will hold that it is computationally more efficient to use fixed-point than floating-point arithmetic.

The reason for the large differences in resource usage and latency is as follows. Recall that a floating-point number consists of a sign bit, a mantissa and an exponent. During every addition or subtraction the exponent part is used by the circuitry to align the radix points (decimal or binary point) of the two numbers. For example, consider adding the unsigned decimal numbers 1.200×10^2 and 3.400×10^0, each with four digits for the mantissa and one digit for the exponent. The radix point in the second number first needs to be shifted to give 0.034×10^2 before adding it to the first number to give the correct answer 1.234×10^2. In contrast, fixed-point numbers have a fixed number of bits for the integer and fractional parts, which means that all the extra circuitry and processing is unnecessary. For the above example, if three digits are used for the integer part and two digits for the fractional part in a fixed-point representation, then the first number would be represented as 120.00 and the second number as 003.40 so that the addition of the two numbers gives the correct answer 123.40.

The price to pay with fixed-point is a reduction in the dynamic range (ratio of the largest representable number to the smallest non-zero representable number) that can be represented with the same amount of bits. Following on from the example above, in fixed-point one would be able to represent the range of non-zero numbers 000.01 to 999.99 using a total of 5 digits, which has a dynamic range of 9.9999×10^5. In floating point one can represent a much larger range of non-zero numbers from 0.001×10^0 to 9.999×10^9, which has a dynamic range of 9.999×10^{12}. At least 13 digits would be necessary in fixed-point in order to represent the same range.

The fundamental problem in representing a variable in fixed-point is to determine the worst-case peak values in order to decide how many bits to allocate for the integer part, while also having a grasp on the dynamic range to decide on the number of bits for the fractional part. In optimization solvers, particularly interior-point methods, some of the variables have a large dynamic range, due to some elements becoming large and others small as the iterates approach the constraints. An implementation of an interior point algorithm in fixed-point would require a very large number of bits for the integer part, in order to capture large numbers, as well as a very large number of bits for the fractional part, in order to capture small numbers.

Algorithm 1 The Lanczos Algorithm

Given q_1 such that $\|q_1\|_2 = 1$ and an initial value $\beta_0 := 1$
for $i = 1$ to i_{max} **do**
1. $q_i \leftarrow \frac{q_i}{\beta_{i-1}}$
2. $z_i \leftarrow \mathsf{A}q_i$
3. $\alpha_i \leftarrow q_i^T z_i$
4. $q_{i+1} \leftarrow z_i - \alpha q_i - \beta_{i-1} q_{i-1}$
5. $\beta_i \leftarrow \|q_{i+1}\|_2$

end for

As mentioned in Section 2.3, the most computationally expensive and numerically critical part of an optimization algorithm is the computation of the search direction ξ, which involves computing the solution to a set of linear equations of the form

$$\mathsf{A}\xi = \mathsf{b}, \tag{8}$$

where A is a symmetric matrix.

The challenge with implementing linear solvers using fixed-point arithmetic is that it is very difficult to bound or control the dynamic range of the variables. Current methods for automatically computing the range of variables cannot handle algorithms that are both nonlinear and recursive, which includes direct and iterative algorithms for solving systems of linear equations. By solving this problem, it would be possible to achieve significant speedups by efficiently implementing part or all of the linear solver in fixed-point.

Iterative methods for solving linear equations, such as MINRES, have a number of advantages compared to direct methods, such as LU or QR factorization; they are easier to parallelize, require less hardware and memory and allows trading off accuracy of the solution against computation time. At the core of modern iterative solvers, such as MINRES, is the Lanczos algorithm, which accounts for the majority of the computational resources and time. In this section, we present some new results that make it possible to implement the Lanczos algorithm in fixed-point arithmetic.

3.1 The Lanczos Algorithm

The Lanczos algorithm, which is described in Algorithm 1, transforms a symmetric matrix $\mathsf{A} \in \mathbb{R}^{\mathsf{N}\times\mathsf{N}}$ into a tridiagonal matrix T with similar spectral properties as A using an orthogonal transformation matrix Q. At every iteration the approximation is refined such that

$$\mathsf{Q}_i^T \mathsf{A} \mathsf{Q}_i = \mathsf{T}_i := \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{i-1} \\ 0 & & \beta_{i-1} & \alpha_i \end{bmatrix}, \tag{9}$$

where $\mathsf{Q}_i \in \mathbb{R}^{\mathsf{N}\times i}$ and $\mathsf{T}_i \in \mathbb{R}^{i\times i}$. The tridiagonal matrix T_i is easier to operate on than the original matrix. The algorithm can be used to extract the eigenvalues and singular values of A or to solve systems using the Conjugate Gradient method when A is positive definite or MINRES when A is indefinite. The Arnoldi iteration, a generalisation of Lanczos for non-symmetric matrices,

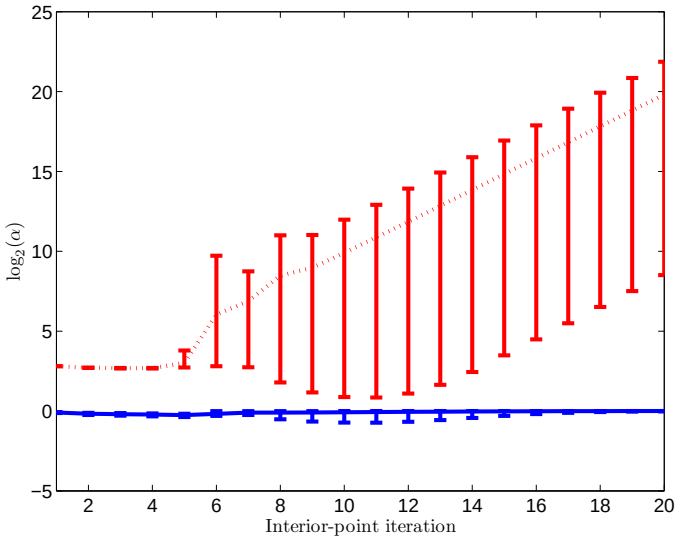

Fig. 4. Evolution of the mean, maximum and minimum peak value of α_i during an interior-point solve for 56 initial states encountered during the implementation of a predictive controller for the same Boeing 747 model as used in Hartley et al. [2012]. The dotted and solid curves represent the unpreconditioned and preconditioned problems, respectively.

is used in the Generalized Minimal Residual method for general matrices.

In order to illustrate the problem of bounding the dynamic values, Figure 4 shows the evolution of the peak value of α (Line 3 in Algorithm 1) generated during an implementation of the interior point method in Wright [1997] for solving a set of constrained LQR problems. For this set of problems, 22 bits are needed for the integer part to be able to represent the largest value of α; moreover, using this number of bits does not guarantee that overflow will not occur for a different set of initial states. Figure 4 also shows the range of the same variable when employing a novel preconditioner, discussed below, to the same set of problems; moreover, this new preconditioner can be used to analytically compute the range of the variables and guarantee that overflow will not occur for a different set of initial states.

3.2 A Novel Preconditioner for (8)

Preconditioning is often used to improve the behavior of iterative methods for solving linear systems, where it is usually employed to accelerate convergence of the algorithm. Our aim is different here, since we are interested in restricting the range of the variables inside the algorithm.

Instead of solving (8) directly, we propose to solve the problem

$$\mathsf{S}^{-\frac{1}{2}}\mathsf{A}\mathsf{S}^{-\frac{1}{2}}\psi = \mathsf{S}^{-\frac{1}{2}}\mathsf{b} \Leftrightarrow \widehat{\mathsf{A}}\psi = \widehat{\mathsf{b}}, \tag{10}$$

where S is chosen to avoid overflow in a fixed-point implementation and the solution to the original problem is recovered through the transformation $\xi = \mathsf{S}^{-\frac{1}{2}}\psi$. We employ a positive diagonal matrix S whose diagonal elements are given by the following expression:

$$\mathsf{S}_{kk} := \sum_{j=1}^{\mathsf{N}} |\mathsf{A}_{kj}|, \tag{11}$$

i.e. S_{kk} is the absolute sum of the elements in row k.

The above preconditioner can be used to derive bounds on the variables within the Lanczos algorithm. The proof of the following result follows from the careful application of standard results from linear algebra and is based on the fact that it is possible to show that the eigenvalues of the preconditioned matrix $\widehat{\mathsf{A}}$ are contained inside the unit disk [Jerez et al., 2012b]:

Theorem 1. Given preconditioner (11), the symmetric Lanczos algorithm applied to $\widehat{\mathsf{A}} := \mathsf{S}^{-\frac{1}{2}}\mathsf{A}\mathsf{S}^{-\frac{1}{2}}$, for any non-singular symmetric matrix A, has intermediate variables with the following bounds for all i: $q_i \in (-1,1)$, $\widehat{\mathsf{A}} \in (-1,1)$, $\widehat{\mathsf{A}}q_i \in (-1,1)$, $\alpha_i \in (-1,1)$, $\beta_i q_{i-1} \in (-1,1)$, $\alpha_i q_i \in (-1,1)$, $\widehat{\mathsf{A}}q_i - \beta_{i-1}q_{i-1} \in (-2,2)$, $q_{i+1} \in (-1,1)$, $q_{i+1}^T q_{i+1} \in (-1,1)$, $\beta_i \in (\epsilon,1)$, and $1/\beta_i \in (1,1/\epsilon)$, where ϵ is a small positive number determined by an appropriately defined termination criterion.

It turns out that the bounds obtained by Theorem 1 are quite tight in practice, as can be seen in Figure 4.

The important practical implication of Theorem 1 is that, provided S is computed with a sufficiently high precision, Theorem 1 can be used to a priori compute the resources required by the Lanczos algorithm to solve the preconditioned problem using fixed-point arithmetic:

Corollary 2. The integer part of a fixed-point two's complement representation requires, including the sign bit, at most one bit for q_i, $\widehat{\mathsf{A}}$, $\widehat{\mathsf{A}}q_i$, α_i, $\beta_i q_{i-1}$, $\alpha_i q_i$, q_{i+1}, β_i and $q_{i+1}^T q_{i+1}$, two bits for $\widehat{\mathsf{A}}q_i - \beta_{i-1}q_{i-1}$ and $\lceil -\log_2(\epsilon) \rceil$ bits for $1/\beta_i$.

Figure 5, taken from Jerez et al. [2012a], shows the trade-off between latency and hardware resources offered by floating-point and fixed-point implementations of the Lanczos algorithm with the new preconditioner (11). The figure shows that fixed-point implementations make more efficient use of the resources and reduces latency, while also being able to guarantee the same accuracy of the solution as with a floating-point implementation.

4. CONCLUSIONS

Existing formulations of predictive control with off-the-shelf optimization solvers are not guaranteed to work when implemented on embedded systems, even if the simulations on a desktop or laptop give acceptable results. This is because the majority of microprocessors in embedded systems do not support IEEE-754 double precision floating-point, due to cost and energy requirements. In order to guarantee that a predictive controller will work in practice, new formulations and solvers need to be developed in order to work on number systems typically implemented in embedded microprocessors.

This paper has shown that the choice of discretization method and details within the optimization solver are critical for guaranteeing good numerical behavior of a predictive controller when using very low precision arithmetic.

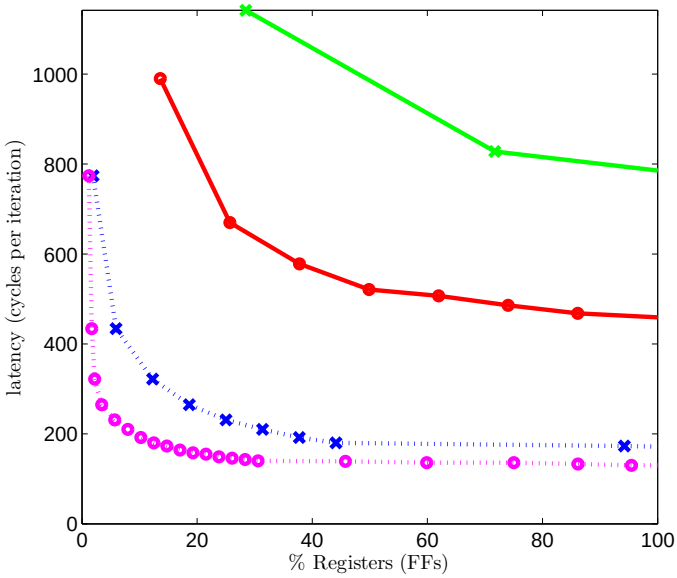

Fig. 5. Latency against FF utilization trade-off on a Xilinx Virtex-7 XT 1140 FPGA for the same benchmark set as in Figure 4 with $\mathsf{N} = 229$; the trade-off is similar for other resources. Double (52-bit mantissa) and single (23-bit mantissa) precision floating-point are represented by solid lines with crosses and circles, respectively. Fixed-point implementations with 53 and 29 bits for the fractional part are represented by the dotted lines with crosses and circles, respectively; these implementations meet the same accuracy specifications as for the floating-point implementations.

In addition, this paper has presented new results that allow for the implementation of the most computationally expensive and numerically critical part of an optimization solver, namely the Lanczos algorithm when computing the search direction using an iterative linear solver, in fixed-point arithmetic.

An important decision that a researcher or developer has to make is whether it is worth investing time in studying computer architecture and arithmetic, since processors are continuously changing and arguably developing at a much faster rate than some well-established programming languages, like C/C++. It might be true that, for some applications, it would be better to wait for technology to catch up such that the details of the processor are unimportant and that the control specifications can be met by porting code that uses existing methods. For some current and future applications this arguably may not happen for a few more decades, if at all. In some applications an understanding of the hardware architecture and associated numerical issues could result in significant improvements in the development cost, performance and reliability of the embedded control system.

This paper presented some numerical results for implementations on an FPGA. At the moment, development tools for FPGAs are arguably not yet as easy to use as the majority of tools for embedded systems based on high level languages, such as C/C++, but we expect this gap to decrease with time. In the meantime, we would like to encourage researchers, hesitant to take the leap to FPGAs but willing to make contributions to the implementation of predictive controllers, not to constrain their ideas to what is currently possible with non-FPGA technologies. By taking an active interest in new ideas and trends in computing they will be able to make fundamental contributions and develop the algorithms of the future.

Our view of which control formulations, algorithms, hardware and software design processes are (or will be) useful has changed since trying to implement predictive controllers on embedded systems. We hope that this paper will have given a glimpse of some of the exciting opportunities for fundamental research in this area.

REFERENCES

G. C. Goodwin, J. I. Yuz, J. C. Agüero, and M. Cea. Sampling and sampled-data models. In *Proc. American Control Conference 2010*, pages 1–20, 2010.

E. Hartley, J. L. Jerez, A. Suardi, E. C. Kerrigan, G. A. Constantinides, and J. M. Maciejowski. Predictive control of a Boeing 747 aircraft using an FPGA. In *Proc. 4th IFAC Nonlinear Model Predictive Control Conference*, Noordwijkerhout, The Netherlands, August 2012.

J. J. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 5th edition, 2011.

J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. Fixed-point Lanczos: Sustaining TFLOP-equivalent performance in FPGAs for scientific computing. In *In Proc. 20th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 53–60, Toronto, Canada, Apr 2012a.

J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan. Towards a fixed point QP solver for predictive control. In *Proc. 51st IEEE Conference on Decision and Control*, Maui HI, USA, December 2012b. Submitted.

I. Koren. *Computer Arithmetic Algorithms*. A. K. Peters Ltd., 2nd edition, 2002.

S. Longo, E. C. Kerrigan, and G. A. Constantinides. A predictive control solver for low-precision data representation. In *Proc. 51st IEEE Conference on Decision and Control*, Maui HI, USA, December 2012. Submitted.

S. Mehrotra. On the implementation of a primal-dual interior-point method. *SIAM J. Optimization*, 2(4):575–601, 1992.

R. H. Middleton and G. C. Goodwin. Improved finite word length characteristics in digital control using delta operators. *IEEE Trans. Automatic Control*, AC-31(11): 1015–1021, 1986.

C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *J. Optimization Theory and Applications*, 99(3):723–757, 1998.

P. O. M. Scokaert and J. B. Rawlings. Constrained linear quadratic regulation. *Automatic Control, IEEE Transactions on*, 43(8):1163 –1169, 1998.

S. J. Wright. Applying new optimization algorithms to model predictive control. In J. C. Kantor, C. E. Garcia, and B. Carnahan, editors, *Proc. 5th Int. Conf. Chemical Process Control - CPC V*, pages 147–155, Lake Tahoe, CA, 1997. CACHE Publications.

https://dspace.org/

Number representation in predictive control

Kerrigan, Eric C.; Jerez, Juan L.; Longo, Stefano; Constantinides, George A.

2012-09-30
Elsevier
https://doi.org/10.3182/20120823-5-NL-3013.00017
Eric C. Kerrigan, Juan L. Jerez, Stefano Longo and George A. Constantinides. Number representation in predictive control. IFAC Proceedings Volumes, Volume 45, Issue 17, 2012, pp. 60-67

Downloaded from a DSpace repository. https://dspace.org/