51st IEEE Conference on Decision and Control
December 10-13, 2012. Maui, Hawaii, USA

Towards a Fixed Point QP Solver for Predictive Control

Juan L. Jerez, George A. Constantinides and Eric C. Kerrigan

Abstract— **There is a need for high speed, low cost and low energy solutions for convex quadratic programming to enable model predictive control (MPC) to be implemented in a wider set of applications than is currently possible. For most quadratic programming (QP) solvers the computational bottleneck is the solution of systems of linear equations, which we propose to solve using a fixed-point implementation of an iterative linear solver to allow for fast and efficient computation in parallel hardware. However, fixed point arithmetic presents additional challenges, such as having to bound peak values of variables and constrain their dynamic ranges. For these types of algorithms the problems cannot be automated by current tools. We employ a preconditioner in a novel manner to allow us to establish tight analytical bounds on all the variables of the Lanczos process, the heart of modern iterative linear solving algorithms. The proposed approach is evaluated through the implementation of a mixed precision interior-point controller for a Boeing 747 aircraft. The numerical results show that there does not have to be a loss of control quality by moving from floating-point to fixed-point.**

I. INTRODUCTION

Implementing a model predictive controller requires being able to solve a quadratic optimization problem in real-time. This is preventing the use of MPC in applications where the sampling requirements are too high for current hardware platforms to compute the solution fast enough and in applications where the cost and power consumption of the computing hardware needed to meet these requirements is beyond budget. For small control problems, say with fewer than four states, it is feasible to compute a solution offline in the form of a piecewise affine control law [2] or through functional approximation [6]. However, for both these approaches, the complexity of the resulting control law grows very fast, hence for medium to large control problems solving the optimization problem online is the only option. Even for relatively small problems, solving the optimization problem online can be the faster alternative [4], [22]. The problem of accelerating the online solution of constrained quadratic programs coming from optimal control problems has been approached from several different directions: by exploiting the structure in the underlying linear algebra computations [24]; through parallel hardware implementations [13], [16], [21], [23]; by exploiting the context of the optimization solver through algorithmic choices and modifications [22]; and by formulating the optimization problem in a way to reduce the computational and memory requirements [12], [17], [20].

In popular algorithms for solving quadratic programs, such as interior-point and active-set methods, the main computational bottleneck is the solution of systems of linear equations arising when solving for the search direction. Because this is a recurring problem in all areas of engineering and science it has received considerable attention. In terms of parallel hardware acceleration, there have been several floating-point implementations of iterative linear solvers in field-programmable gate arrays (FPGAs) demonstrating performance improvements of more than one order of magnitude over high-end microprocessors [3], [4]. However, when possible, porting an iterative algorithm from floating-point to fixed-point arithmetic can reduce both the resource requirements and arithmetic delays by more than one order of magnitude in a custom hardware implementation, leaving potential for very significant performance improvements up to two orders of magnitude [11]. Furthermore, being able to implement optimization algorithms in fixed-point could enable controller implementations in low cost and low power off-the-shelf hardware such as microcontrollers and fixed-point digital signal processors (DSPs). Furthermore, there is still a lack of floating-point support in high-level FPGA design flows, such as LabVIEW FPGA, due to the instantiation of floating-point units quickly exhausting the capacity of modest size devices.

Unfortunately, porting an application to fixed-point is not straightforward. First, one needs to establish bounds on the worst-case peak values of all variables in order to avoid overflow errors. Second, the dynamic range of all variables should be constrained enough such that numbers can be represented accurately using a reasonable number of bits. Current analysis tools in the design automation community cannot handle algorithms that are both nonlinear and recursive [5], a property shared by most algorithms for solving systems of linear equations, direct and iterative.

We concentrate on the Lanczos iteration, which is the building block in modern iterative algorithms for solving linear equations, hence the heart of any optimization solver based on an iterative linear solver. We propose to use a novel form of preconditioning with the objective of establishing tight analytical bounds on all variables of the process regardless of the properties of the original Karush-Kuhn-Tucker (KKT) matrix. This allows us to create a mixed precision interior-point solver where the computationally intensive Lanczos algorithm is implemented in fixed-point and the remaining operations are implemented in floating-point. The numerical behaviour of the proposed solver is verified in the

Juan L. Jerez and George A. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, United Kingdom jlj05@imperial.ac.uk, gac1@imperial.ac.uk

Eric C. Kerrigan is with the Department of Electrical and Electronic Engineering and the Department of Aeronautics, Imperial College London, SW7 2AZ, United Kingdom e.kerrigan@imperial.ac.uk

978-1-4673-2064-1/12/$31.00 ©2012 IEEE

context of a model predictive controller for a Boeing 747 model [15] and the behaviour is compared against full single and double precision floating-point implementations of the same algorithm, showing that it is possible to preserve the accuracy when moving to a fixed-point representation.

This paper is organized as follows: Section II describes the Lanczos algorithm; Section III describes the preconditioning procedure and includes the main theoretical results in the paper; the results for the Boeing 747 case study are presented in Section IV to demonstrate the feasibility of the proposed approach; and Section V concludes the paper.

II. THE LANCZOS ALGORITHM

The Lanczos algorithm [14], described in Algorithm 1, transforms a symmetric matrix $A \in \mathbf{R}^{N\times N}$ 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

$$Q_i^T A Q_i = T_i := \begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{i-1} \\ 0 & & \beta_{i-1} & \alpha_i \end{bmatrix}, \quad (1)$$

where $Q_i \in \mathbf{R}^{N\times i}$ and $T_i \in \mathbf{R}^{i\times i}$. The tridiagonal matrix T_i is easier to operate on than the original matrix. It can be used to extract the eigenvalues and singular values of A [7], or to solve systems of linear equations of the form $Ax = b$ using the conjugate gradient (CG) [10] method when A is positive definite or the minimum residual (MINRES) [19] method when A is indefinite. The Arnoldi iteration, a generalisation of Lanczos for non-symmetric matrices, is used in the generalized minimum residual (GMRES) method for general matrices. The Lanczos (and Arnoldi) algorithms account for the majority of the computation in these methods, hence are the key building blocks in modern iterative algorithms for solving all formulations of linear systems appearing in optimization solvers for optimal control problems.

Methods involving the Lanczos iteration are typically used for large sparse problems arising in scientific computing where direct methods, such as LU and Cholesky factorization, cannot be used due to prohibitive memory requirements [8]. However, iterative methods have additional properties that also make them preferable for small problems arising in real-time applications, since they allow one to trade-off accuracy for computation time [4].

Algorithm 1 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. $z_i \leftarrow Aq_i$
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

III. FIXED-POINT IMPLEMENTATION

A floating-point number consists of a sign bit, an exponent and a mantissa. The exponent value moves the binary point with respect to the mantissa. The dynamic range – the ratio of the smallest to largest representable number – grows doubly exponentially with the number of exponent bits, therefore it is possible to represent a wide range of numbers with a relatively small number of bits. However, because two operands can have different exponents it is necessary to perform denormalisation and normalisation operations before and after every addition or subtraction, leading to greater resource usage and longer delays. In contrast, fixed-point numbers have a fixed number of bits for the integer and fraction fields, i.e. the exponent does not change with time and it does not have to be stored. Fixed-point computations are the same as with integer arithmetic, hence the circuitry is simple and fast, but the dynamic range is limited.

In order to represent a variable in fixed-point one has to determine the worst case peak value to decide how many bits to allocate for the integer part. The dynamic range has to be small such that small numbers can also be represented with a good level of accuracy. In interior-point solvers for model predictive control, some elements and eigenvalues of the KKT matrix have a wide dynamic range during a single solve, due to some elements becoming large and others small as the current iteration approaches the constraints. This affects the dynamic range of all variables in the Lanczos method. If one were to directly implement the algorithm in fixed-point, one would have to allocate a very large number of bits for the integer part to capture large numbers and an equally large number of bits for the fractional part to capture small numbers. Furthermore, there will be no guarantees of the avoidance of overflow errors, since most of the expressions cannot be analytically bounded in the general case.

As a result, it is essential to precondition the problem to influence the bounds such that the same precision circuitry can handle linear systems with a wide range of matrices in an efficient way. Note that in this case the primary objective of the preconditioning procedure is not in accelerating the convergence of the iterative algorithm, although empirical evidence suggests that acceleration is also achieved in many cases. For example, if we want to solve the system of linear equations $Ax = b$, we propose instead to solve the problem

$$M^{-\frac{1}{2}} A M^{-\frac{1}{2}} y = M^{-\frac{1}{2}} b$$
$$\widehat{A} y = \widehat{b},$$

where M is chosen to ensure the absence of overflow in a fixed-point implementation. The solution to the original problem can be recovered easily through the transformation $x = M^{-\frac{1}{2}} y$.

Note that the preconditioning procedure and the recovery of the solution still have to be computed using floating-point arithmetic due to the potentially large dynamic range, hence it is required that the preconditioner is as simple as possible. We employ the simplest invertible preconditioner, a positive

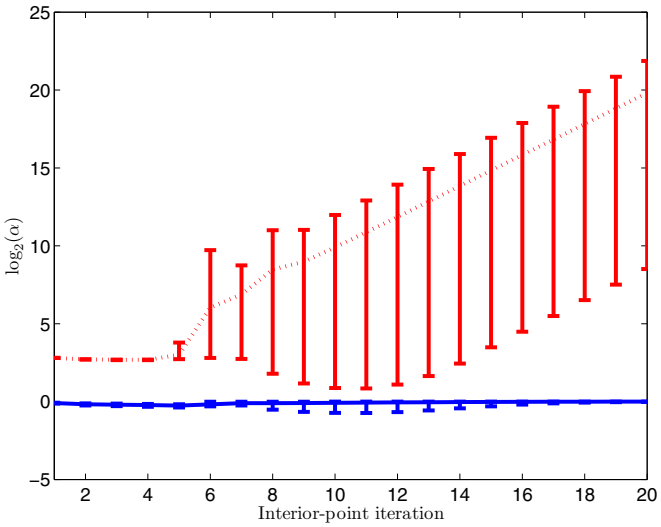

Fig. 1. Evolution of the mean, maximum and minimum peak value of α_i during an interior-point solve for a benchmark of 56 initial conditions for an optimal controller for a Boeing 747 model [15]. The solid and dotted curves represent the unpreconditioned and preconditioned algorithms, respectively.

diagonal matrix M whose diagonal elements are given by the following expression:

$$M_{kk} := \sum_{j=1}^{N} |A_{kj}|, \tag{2}$$

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

Note that in the general case the computational overhead of the preconditioning procedure is comparable to the cost of one iteration of the Lanczos algorithm. Since many iterations are typically required, most of the computation is still carried out in fixed-point arithmetic. In the context of predictive control, only a small number of elements in the matrix A are changing from iteration to iteration making the cost of implementing this preconditioner even smaller. For instance, if there are only bound constrainst on the inputs, the floating-point preconditioning cost is almost negligible compared to the fixed-point Lanczos computation.

In order to illustrate the need for a preconditioner, Figure 1 shows the evolution of the peak value of α (Line 3 in Algorithm 1) throughout an interior-point solve for a benchmark set of problems described in Section IV. Without using preconditioner (2) one would have to allocate 22 bits for the integer part to be able to represent the largest value of α occurring in this benchmark set. In addition, using that number of bits does not guarantee that overflow will not occur on a different set of problems.

When using preconditioner (2) we have the following results:

Lemma 1: The preconditioned matrix $\widehat{A} := M^{-\frac{1}{2}} A M^{-\frac{1}{2}}$ has, for any non-singular symmetric matrix A, spectral radius $\rho(\widehat{A}) \leq 1$.

Proof: Let $R_k := \sum_{j \neq k}^{N} |A_{kj}|$ be the absolute sum of the off-diagonal elements in a row, and let $D(A_{kk}, R_k)$ be a Geshgorin disc with centre A_{kk} and radius R_k. Consider an alternative non-symmetric preconditioned matrix $\widetilde{A} := M^{-1}A$. The absolute row sum is equal to 1 for every row of $\widetilde{A}$, hence the Gershgorin discs associated with this matrix are given by $D(\widetilde{A}_{kk}, 1-|\widetilde{A}_{kk}|)$. It is straightforward to show that these discs always lie inside the interval between 1 and -1 when $|\widetilde{A}_{kk}| \leq 1$, which is the case here. Hence, $\rho(\widetilde{A}) \leq 1$ according to Geshgorin's circle theorem [8, Theorem 7.2.1]. Now,

$$M^{-1}Ax = \lambda x \tag{3}$$

$$\Leftrightarrow M^{-\frac{1}{2}}Ax = M^{\frac{1}{2}}\lambda x \tag{4}$$

$$\Leftrightarrow M^{-\frac{1}{2}}AM^{-\frac{1}{2}}y = \lambda y \tag{5}$$

where (5) is obtained by substituting x for $M^{-\frac{1}{2}}y$. This shows that the eigenvalues of the non-symmetric preconditioned matrix $\widetilde{A}$ and the symmetric preconditioned matrix $\widehat{A}$ are the same. The eigenvectors are different but this does not affect the bounds, which we derive next. ■

Theorem 1: Given preconditioner (2), the symmetric Lanczos algorithm applied to $\widehat{A}$, for any non-singular symmetric matrix A, has intermediate variables with the following bounds for all i, k and j:

- $[q_i]_k \in (-1, 1)$
- $[\widehat{A}]_{kj} \in (-1, 1)$
- $[\widehat{A}q_i]_k \in (-1, 1)$
- $\alpha_i \in (-1, 1)$
- $[\beta_i q_{i-1}]_k \in (-1, 1)$
- $[\alpha_i q_i]_k \in (-1, 1)$
- $[\widehat{A}q_i - \beta_{i-1}q_{i-1}]_k \in (-2, 2)$
- $[q_{i+1}]_k \in (-1, 1)$
- $q_{i+1}^T q_{i+1} \in (-1, 1)$
- $\beta_i \in (0, 1)$,

where i denotes the iteration number and $[]_k$ and $[]_{kj}$ denote the k^{th} component of a vector and kj^{th} component of a matrix, respectively.

Corollary 1: For the integer part of a fixed-point 2's complement representation we require, including the sign bit, one bit for q_i, $\widehat{A}$, $\widehat{A}q_i$, α_i, $\beta_i q_{i-1}$, $\alpha_i q_i$, q_{i+1}, β_i and $q_{i+1}^T q_{i+1}$, and two bits for $\widehat{A}q_i - \beta_{i-1}q_{i-1}$.

Proof: The individual elements of the Lanczos vectors q_i are trivially between 1 and -1 because they form an orthonormal basis, hence they have unit norm. We follow by bounding the elements of the coefficient matrix:

$$|\widehat{A}_{kj}| = \frac{1}{\sqrt{M_{kk}}} \frac{1}{\sqrt{M_{jj}}} |A_{kj}| \tag{6}$$

$$\leq \frac{1}{\sqrt{|A_{kj}|}} \frac{1}{\sqrt{|A_{kj}|}} |A_{kj}| \tag{7}$$

$$= 1, \tag{8}$$

where (7) comes from the minimum value of M_{jj} and M_{kk} being $|A_{kj}|$.

Using Lemma 1 we can put bounds on the rest of the intermediate computations in the Lanczos iteration. We start

with $\widehat{A}q_i$, which is used in lines 2 and 3 in Algorithm 1:

$$\begin{aligned}\|\widehat{A}q_i\|_\infty \leq & \|\widehat{A}q_i\|_2 && (9)\\ = & \|\widehat{A}\|_2, && (10)\\ = & \max_k |\lambda_k(\widehat{A})| && (11)\\ \leq & 1, && (12)\end{aligned}$$

where (9) comes from the properties of vector norms and (10) comes from the properties of the matrix norm (see [8]) and the fact that $\|q_i\|_2 = 1$. Equality (11) follows from the 2-norm of a real symmetric matrix being equal to its largest absolute eigenvalue [8, Theorem 2.3.1].

We continue by bounding α_i, which is computed in line 2 of Algorithm 1 and is part of the tridiagonal matrix described in (1). From the Courant-Fischer minimax theorem [8, p. 394] we know that the maximum and minimum values of the Rayleigh quotient defined as

$$r(x) := \frac{x^T A x}{x^T x} \tag{13}$$

correspond to the maximum and minimum eigenvalues of A. Hence,

$$\begin{aligned}-1 \leq & \frac{q_i^T \widehat{A} q_i}{q_i^T q_i} \leq 1 && (14)\\ \Leftrightarrow -1 \leq & \, q_i^T \widehat{A} q_i \leq 1 && (15)\\ \Leftrightarrow -1 \leq & \, \alpha_i \leq 1, && (16)\end{aligned}$$

where (15) comes from the fact that q_i has unit norm and (16) comes from the definition of α_i. It follows that the elements of $\alpha_i q_i$ are also between 1 and -1.

The last variable that needs to be bounded is β_i, which is used in lines 1, 3 and 4 in Algorithm 1 and is also part of the tridiagonal matrix described in (1). The eigenvalues of the tridiagonal approximation matrix (1) are contained within the eigenvalues of $\widehat{A}$ [18]. Hence, we can write

$$-1 \leq x^T T_i x \leq 1, \qquad \forall \|x\| = 1. \tag{17}$$

If we choose two unit norm vectors $x_1 := \begin{bmatrix}\cdots & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \cdots\end{bmatrix}^T$ and $x_2 := \begin{bmatrix}\cdots & 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \cdots\end{bmatrix}^T$, then we have

$$\begin{aligned}-1 \leq & \frac{\alpha_i + \alpha_{i+1} + 2\beta_i}{2} \leq 1, && (18)\end{aligned}$$

and

$$\begin{aligned}-1 \leq & \frac{\alpha_i + \alpha_{i+1} - 2\beta_i}{2} \leq 1. && (19)\end{aligned}$$

If we denote $s = \alpha_1 + \alpha_2 \in (-2, 2)$, then β_i has to simultaneously satisfy

$$\begin{aligned}-2 - s \leq & \, 2\beta_i, && (20)\\ 2\beta_i \leq & \, 2 - s, && (21)\\ -2 - s \leq & -2\beta_i, && (22)\\ -2\beta_i \leq & \, 2 - s. && (23)\end{aligned}$$

These bounds are maximized when $s = 0$, giving $-1 \leq \beta_i \leq 1$. However, we know that β_i is non-negative, because it is calculated as the norm of q_{i+1} in line 4 of Algorithm 1. It follows that the scalar $q_{i+1}^T q_{i+1}$ and the individual elements of q_{i+1} and $\beta_i q_{i-1}$ also take values in the set $(-1, 1)$. For the intermediate result $\widehat{A}q_i - \beta_i q_{i-1}$ we can employ interval arithmetic to establish a bound between -2 and 2. ■

Remark 1: Division and square root operations are implemented as iterative procedures in digital architectures. The data types for the intermediate variables can be designed to prevent overflow errors. In this case, the fact that $[q_{i+1}]_k \leq \beta_i$ and $q_{i+1}^T q_{i+1} \leq 1$ can be used to establish tight bounds on the intermediate results for any implementation.

Remark 2: A possible source of problems both for fixed-point and floating-point implementations is encountering $\beta_i = 0$. However, we know that encountering $\beta_i = 0$ means that we have already computed a perfect tridiagonal approximation to $\widehat{A}$, i.e. the roots of the characteristic polynomial of T_{i-1} are the same as those of the charcateristic polynomial of T_i. Since in predictive control we do not require exact solutions for the search direction, we would have terminated the algorithm before reaching this point.

IV. NUMERICAL RESULTS

In this section we show that it is indeed possible to implement the numerically intensive part of the QP solver in fixed-point without compromising on the numerical accuracy of the solution. The effect on computing performance and resource requirements is quantified in [11] in the context of an FPGA.

In order to investigate the numerical behaviour of the proposed approach we will evaluate the effect of number representation on the solution of linear equations and the solution of convex quadratic programs coming from a predictive controller for a Boeing 747 model [15]. For the simplified model of the aircraft the objective is to control the roll and pitch angles as well as the airspeed. There are 17 control variables including ailerons, spoiler panels, elevators, rudders and four engines. There are 12 states including angles and rates, airspeed and engine states. The prediction and control horizons are set to five and the optimal control problem is formulated as as banded QP

$$\min_\theta \frac{1}{2}\theta^T H \theta + h^T \theta \quad \text{subject to} \quad F\theta = f, \quad G\theta \leq g, \tag{24}$$

in non-condensed form [24], where θ includes the states and inputs for the whole horizon. A scaling transformation is applied in order to improve the conditioning of the problem [9].

We propose a mixed precision interior-point solver (refer to Algorithm 2) where the Lanczos iterations – the most computationally intensive part in an interior-point solver based on an iterative linear solver (MINRES in this case) – is computed in fixed-point, whereas the rest of the algorithm is computed in double precision floating-point. The fixed-point behaviour is simulated using Matlab's fixed-point toolbox [1], which allows specifying rounding and overflow modes. When using floor rounding and no saturation, the results were verified to match simulation results on a Xilinx

Algorithm 2 Primal-Dual Interior-Point Algorithm

Choose any initial point $(\theta_0, \nu_0, \lambda_0, s_0)$ with $[\lambda_0^T \; s_0^T]^T > 0$
for $k = 0, 1, 2, \ldots$ **do**

$$\mathcal{A}_k := \begin{bmatrix} H + G^T W_k G & F^T \\ F & 0 \end{bmatrix}$$

$$b_k := \begin{bmatrix} -h - F^T\nu - G^T(\lambda_k - W_k g + \sigma\mu_k s_k^{-1}) \\ -F\theta_k + f \end{bmatrix}$$

Solve $\mathcal{A}_k z_k = b_k$ for $z_k =: \left[(\theta_k + \Delta\theta_k)^T \; \Delta\nu_k^T\right]^T$

$\Delta\lambda_k := W_k(G(\theta_k + \Delta\theta_k) - g) + \sigma\mu_k s_k^{-1}$

$\Delta s_k := -s_k - (G(\theta_k + \Delta\theta_k) - g)$

$\alpha_k := \max\{\alpha \in (0, 1] : (\lambda_k, s_k) + \alpha(\Delta\lambda_k, \Delta s_k) > 0\}$

$(\theta_{k+1}, \nu_{k+1}, \lambda_{k+1}, s_{k+1}) :=$
$(\theta_k, \nu_k, \lambda_k, s_k) + \alpha_k(\Delta\theta_k, \Delta\nu_k, \Delta\lambda_k, \Delta s_k)$

end for

FPGA with the same options for the arithmetic units. We created a benchmark set of linear systems by generating 56 QP problems with different initial states and running the solver for 20 iterations.

Figure 2 shows the relative error at convergence for a fixed-point implementation with 32 bits for the fraction field and several floating-point implementations. The error is smallest for the double precision floating-point implementations but the 32-bit fixed-point implementations, in general, outperform single precision floating-point, which only has 24 mantissa bits. This is a consequence of the preconditioner constraining the dynamic range of all variables, allowing a fixed-point scheme with a small number of bits to efficiently represent numbers. Figure 2 also shows that the MINRES algorithm fails to converge for unpreconditioned problems even when implementing the algorithm in floating-point. This suggests that preconditioner (2) can also be suitable for other purposes in addition to allowing efficient fixed-point implementations. Figure 3 shows that the convergence speed is similar for all implementations. The slight differences are due to larger final errors being reached in a smaller number of iterations, but the rate of convergence is the same for all implementations.

In order to evaluate the closed-loop behaviour of the different controller implementations we ran a simulation where the aircraft is at steady-state and the reference changes at $t = 5$s. This change in reference guarantees that the input constraints become active. The sampling period is $T_s = 0.2$s and the states and inputs are given equal weight in the cost function. 150 MINRES iterations and 20 interior-point iterations are used in all cases in order to have a fair comparison. Figure 4 shows the accumulated cost for different controller implementations. When using the preconditioner, the quality of the control is practically the same with the 32-bit fixed-point controller as with the double precision floating-point controller. For the unpreconditioned controller we used information about the maximum value that all signals in the Lanczos process took for the benchmark set to decide how many bits to allocate for the integer part of each signal. The total number of bits for each signal was kept constant with

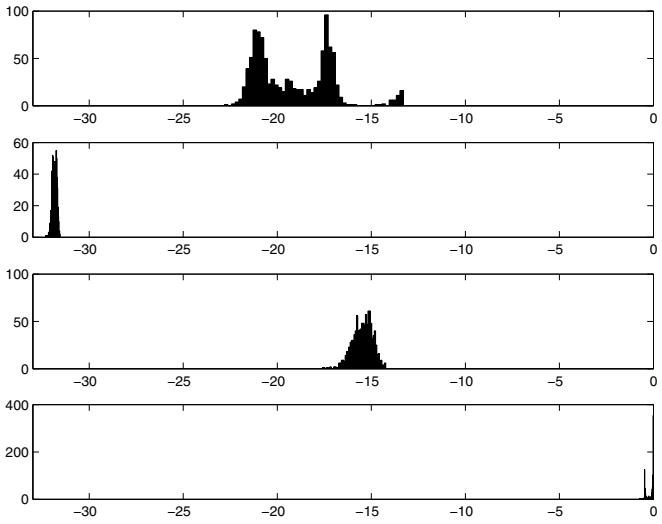

Fig. 2. Histogram showing the final log relative error $\log(\frac{\|Ax-b\|_2}{\|b\|_2})$ at termination for different linear solver implementations. From top to bottom, preconditioned 32-bit fixed-point, double precision floating-point and single precision floating-point implementations, and unpreconditioned single precision floating-point implementation.

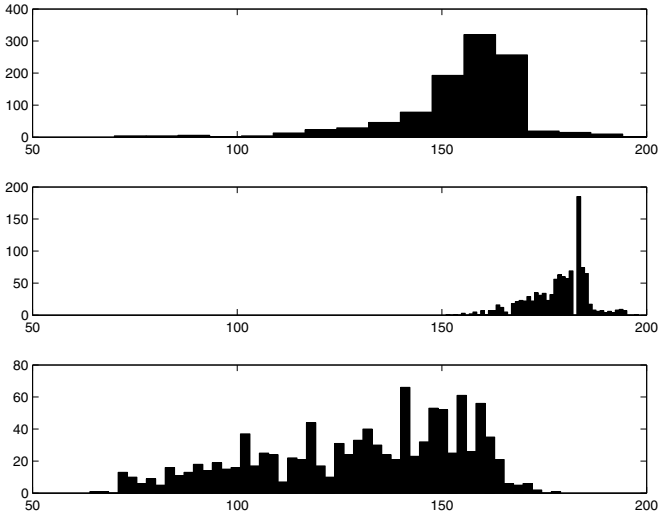

Fig. 3. Histogram showing the number of MINRES iterations to termination for 32-bit fixed-point, double precision floating-point and single precision floating-point implementations.

respect to the preconditioned controller implementation. Of course, this cannot guarantee the absence of overflow so we changed the overflow mode to saturation (this would incur an extra penalty in terms of hardware resources). Figure 4 shows that it is essential to apply preconditioner (2) in order to be able to implement the mixed precision controller and still maintain the control quality.

V. CONCLUSION

We have presented a novel use of preconditioning that allows establishing very tight analytical bounds on all variables of the Lanczos iteration, which is the most computationally intensive block in modern iterative linear solvers. This approach allows a fixed-point implementation to efficiently

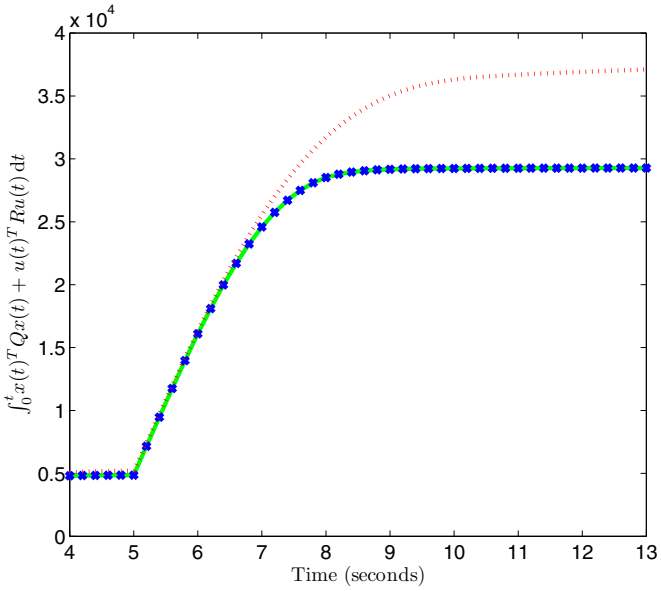

Fig. 4. Accumulated closed-loop cost for different mixed precision interior-point controller implementations. The dotted line represents the unpreconditioned 32-bit fixed-point controller, whereas the crossed and solid lines represent the preconditioned 32-bit fixed-point and double precision floating-point controllers, respectively.

represent numbers using a small number of bits, while at the same time guaranteeing the absence of overflow errors. The move from floating-point to fixed-point can potentially improve computation speed by several orders of magnitude as well as allowing complex MPC controllers to be implemented in low cost hardware. The numerical accuracy and speed of convergence of the proposed implementation has been verified to be as good as double precision floating-point for control purposes.

Future work could focus on extending this methodology to enable fixed-point implementations of entire linear solvers, iterative and factorization-based, and to the rest of the interior-point algorithm through several layers of preconditioning or different linear system formulations.

VI. ACKNOWLEDGEMENTS

This work was supported by the EPSRC (Grants EP/G031576/1 and EP/I012036/1) and the EU FP7 Project EMBOCON, as well as industrial support from Xilinx, the Mathworks, and the European Space Agency. The authors would also like to thank Dr Ed Hartley for providing the benchmark set of initial conditions for the Boeing 747 optimal controller.

REFERENCES

[1] MATLAB fixed-point toolbox. `http:/www.mathworks.com/products/fixed/`, 2012.

[2] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, Jan 2002.

[3] D. Boland and G. A. Constantinides. An FPGA-based implementation of the MINRES algorithm. In *Proc. Int. Conf. on Field Programmable Logic and Applications*, pages 379–384, Heidelberg, Germany, Sep 2008.

[4] G. A. Constantinides. Tutorial paper: Parallel architectures for model predictive control. In *Proc. European Control Conference*, pages 138–143, Budapest, Hungary, Aug 2009.

[5] G. A. Constantinides, N. Nicolici, and A. B. Kinsman. Numerical data representations for FPGA-based scientific computing. *IEEE Design & Test of Computers*, 28(4):8–17, Aug 2011.

[6] A. Domahidi, M. Zeilinger, M. Morari, and C. Jones. Learning a Feasible and Stabilizing Explicit Model Predictive Control Law by Robust Optimization. In *Proc. 50th IEEE Conf. on Decision and Control*, pages 513–519, Orlando, FL, USA, Dec 2011.

[7] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal on Numerical Analysis*, 2(2):205–224, 1965.

[8] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, USA, 3rd edition, 1996.

[9] E. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides. Predictive control of a Boeing 747 aircraft using an FPGA. In *Proc. 4th IFAC Nonlinear Model Predictive Control Conference*, pages 80–85, 2012.

[10] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, Dec 1952.

[11] 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 2012.

[12] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica*, 5(48):999–1002, May 2012.

[13] J. L. Jerez, K.-V. Ling, G. A. Constantinides, and E. C. Kerrigan. Model predictive control for deeply pipelined field-programmable gate array implementation: Algorithms and circuitry. *IET Control Theory and Applications*, 8(6):1029–1041, Jul 2012.

[14] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, Oct 1950.

[15] C. V. D. Linden, H. Smaili, A. Marcos, G. Balas, D. Breeds, S. Runhan, C. Edwards, H. Alwi, T. Lombaerts, J. Groeneweg, R. Verhoeven, and J. Breeman. GARTEUR RECOVER benchmark. `http://www.faulttolerantcontrol.nl/`, 2011.

[16] K.-V. Ling, B. F. Wu, and J. M. Maciejowski. Embedded model predictive control (MPC) using a FPGA. In *Proc. 17th IFAC World Congress*, pages 15250–15255, Seoul, Korea, Jul 2008.

[17] G. M. Mancuso and E. C. Kerrigan. Solving constrained LQR problems by eliminating the inputs from the QP. In *Proc. 50th IEEE Conf. on Decision and Control*, pages 507–512, Orlando, FL, USA, Dec 2011.

[18] C. C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra and its Applications*, 34:235–258, Dec 1980.

[19] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, Sep 1975.

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

[21] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare. A system-on-a-chip implementation for embedded real-time model predictive control. *IEEE Transactions on Control Systems Technology*, 17(5):1006–1017, Sep 2009.

[22] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, Mar 2010.

[23] A. G. Wills, G. Knagge, and B. Ninness. Fast linear model predictive control via custom integrated circuit architecture. *IEEE Transactions on Control Systems Technology*, 20(1):59–71, 2012.

[24] S. J. Wright. Interior-point method for optimal control of discrete-time systems. *Journal on Optimization Theory and Applications*, 77:161–187, 1993.