Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

A Family of High-Performance Solvers for Linear Model Predictive Control

Gianluca Frison * **Leo Emil Sokoler** *
John Bagterp Jørgensen *

* *Technical University of Denmark, DTU Compute - Department of Applied Mathematics and Computer Science, DK-2800 Kgs Lyngby, Denmark. (corresponding author: giaf at imm.dtu.dk).*

Abstract: In Model Predictive Control (MPC), an optimization problem has to be solved at each sampling time, and this has traditionally limited the use of MPC to systems with slow dynamic. In this paper, we propose an efficient solution strategy for the unconstrained sub-problems that give the search-direction in Interior-Point (IP) methods for MPC, and that usually are the computational bottle-neck. This strategy combines a Riccati-like solver with the use of high-performance computing techniques: in particular, in this paper we explore the performance boost given by the use of single precision computation, and techniques such as inexact search direction and mixed precision computation. Finally, we test our HPMPC toolbox, a family of high-performance solvers tailored for MPC and implemented using these techniques, that is shown to be several times faster than current state-of-the-art solvers for linear MPC.

1. INTRODUCTION

Model Predictive Control (MPC) has bee traditionally limited to systems with slow dynamic, with sampling times of seconds or minutes. This is due to the fact that an optimization problem needs to be solved at each sampling time. Nowadays, thanks to algorithmic as well as hardware improvements, this is no more the case, and recent works show that, in case of small systems, even control frequency of milliseconds are possible. The two main approaches for fast MPC are explicit (see Bemporad et al. [2002]) and structure-exploiting on-line MPC (see e.g. Rao et al. [1998],Wang et al. [2010]).

In recent years, several approaches have been proposed to the fast on-line solution of small-scale linear MPC problems, as flat code generation (CVXGEN, Mattingley et al. [2012]) and customized triple-loop based BLAS (FORCES, Domahidi et al. [2012]). However, these solvers do not fully exploit the hardware capabilities of modern architectures, and rely on compilers for the code optimization. As a result, typically they can attain only a small fraction of processor peak performance.

In this paper, we propose an efficient solver for the Linear-Quadratic Control Problem (LQCP), that is a common sub-problem in optimal control and estimation, and in particular it gives the search direction in Interior-Point (IP) methods for linear MPC. Our solver for LQCP only requires 3 calls to linear-algebra routines for the factorization of the KKT system: this decreases the data movement, and allows us to hand optimize these few routines. In particular, we make use of high-performance techniques such as blocking for registers, SIMD instructions, customized BLAS and mixed precision computation. The latter exploits the fact that on the target architecture (in this paper, an Intel's processor) the peak performance of single precision computation is twice as much as in double precision. The resulting solver for LQCP is shown to attain a large fraction of the peak performance for a wide range of problem sizes.

This high-performance solver is used as a routine in primal-dual and Mehrotra's predictor-corrector IP methods for linear MPC. Furthermore, we propose the use of inexact IP methods, where the search directions are found by solving the LQCP sub-problems in single precision in early iterations. These IP methods can produce a solution in double precision in a time that is only slightly larger than in single precision. The resulting solver is several times faster than state-of-the-art solvers for linear MPC (see Domahidi et al. [2012] as a reference), and the high-performance is attained for a wider range of problem sizes.

The paper is organized as follows: section 2 describes the LQCP and linear MPC problems. Section 3 presents high-performance solvers for the LQCP, in single, double and mixed precision. Section 4 briefly introduces primal-dual and Mehrotra's predictor-corrector IP methods, and proposes the inexact IP. Section 5 presents the results of some numerical test, and Section 6 contains the conclusion.

2. PROBLEMS DESCRIPTION

In this paper, we focus our attention on efficient solvers for the LQCP. This is a rather general formulation that can represent several problems in optimal control and estimation, and in particular it gives the search direction in Interior-Point (IP) methods for MPC. Thus an high-performance implementation of a solver for the LQCP can boost the performance of solvers for a wide class of problems.

2.1 Linear-quadratic control problem

The LQCP is the equality constrained quadratic program

$$\begin{aligned} \min_{u_n, x_{n+1}} \quad & \phi = \sum_{n=0}^{N-1} \varphi_n(x_n, u_n) + \varphi_N(x_N) \\ s.t. \quad & x_{n+1} = A_n x_n + B_n u_n + b_n \end{aligned} \tag{1}$$

where

$$\begin{aligned} \varphi_n(x_n, u_n) &= \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix}' \begin{bmatrix} R_n & S_n & s_n \\ S_n' & Q_n & q_n \\ s_n' & q_n' & \rho_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix} = \mathcal{X}_n' \mathcal{Q}_n \mathcal{X}_n \\ \varphi_N(x_N) &= \begin{bmatrix} u_N \\ x_N \\ 1 \end{bmatrix}' \begin{bmatrix} 0 & 0 & 0 \\ 0 & P & p \\ 0 & p' & \pi \end{bmatrix} \begin{bmatrix} u_N \\ x_N \\ 1 \end{bmatrix} = \mathcal{X}_N' \mathcal{P} \mathcal{X}_N \end{aligned} \tag{2}$$

All matrices can in general be dense and time variant. In this paper, we assume that the matrices $\mathcal{Q}_n$ are symmetric positive definite.

2.2 Linear MPC problem

The linear MPC problem with linear constraints is the quadratic program

$$\begin{aligned} \min_{u_n, x_{n+1}} \quad & \phi = \sum_{n=0}^{N-1} \varphi_n(x_n, u_n) + \varphi_N(x_N) \\ s.t. \quad & x_{n+1} = A_n x_n + B_n u_n + b_n \\ & C_n x_n + D_n u_n \geq d_n \\ & C_N x_N \geq d_N \end{aligned} \tag{3}$$

where $\varphi_N(x_N)$ are defined as in (2). Again, all matrices can in general be dense and time variant.

3. SOLVERS FOR THE LQ CONTROL PROBLEM

In this section we present algorithms (and relative implementation) to efficiently solve LQCP. These algorithms can be used as building blocks in different IP methods.

3.1 Solution strategies

There exists several solution strategies for the LQCP (1). In the following of the paper we will consider two of them.

The first solution strategy is based on the fact that the LQCP (1) is an instance of the equality constrained quadratic program

$$\begin{aligned} \min_{x} \quad & \phi = \tfrac{1}{2} z' H z + g' z \\ s.t. \quad & Az = b \end{aligned} \tag{4}$$

The (in general only necessary) optimality conditions for (4) are the well-known KKT conditions, that can be written in matrix notation as

$$\begin{bmatrix} H & -A' \\ -A & 0 \end{bmatrix} \begin{bmatrix} z^* \\ \pi^* \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{5}$$

that is called the KKT system associated with (4). In the case of the LQCP, it can be proved that, if the matrices $\begin{bmatrix} R_n & S_n \\ S_n' & Q_n \end{bmatrix}$ and P are positive definite, then the KKT conditions are also sufficient and (5) has an unique solution. The KKT system of the LQCP is large and sparse, and has a special structure that can be exploited to obtain efficient solvers, see Rao et al. [1998]. The fact that the solution of the LQCP can be obtained by means of the solution of a system of linear equations (i.e. through factorization of the matrix and backward and forward substitutions) implies that we can use mixed precision to perform the computations, as shown later.

Another solution strategy is based on dynamic programming. We do not want to present the theory again (that can be found for example in Jørgensen [2005]), but only show that this leads to an efficient solver in practice, where the factorization and backward substitution in the solution of (5) are fused, as shown in section 3.2.

To implement the IP methods, we need routines to factorize and solve the KKT system, to solve an already factorized KKT system, and to compute the residuals. These routines can be seen as building blocks to implement a number of different IP methods.

3.2 Factorization and solution of the KKT system

The dynamic programming approach can be used to derive an efficient solver, analogue to the classical Riccati recursion but more efficient in practice, where the factorization and the backward substitution are fused together: see Frison et al. [2014] for the details of the derivation. The algorithm (together with the calls to BLAS functions) is presented in Algorithm 1 (where $\mathcal{M}_n^{1/2}$ is the lower triangular Cholesky factor of matrix $\mathcal{M}_n$, partitioned as $\mathcal{Q}_n$ in (2)), and only requires 3 function calls per backward iteration and 3 per forward iteration: this reduces the overhead associated with the function calls, as well as the data movement. The cost of the algorithm is $N\big((\frac{7}{3}n_x^3 + 4n_x^2 n_u + 2n_x n_u^2 + \frac{1}{3}n_u^3) + (\frac{13}{2}n_x^2 + 9n_x n_u + \frac{5}{2}n_u^2)\big)$ flops, plus eventually $N(2n_x^2)$ if π is needed (as e.g. in mixed precision).

Algorithm 1 Factorization and solution of LQCP

```
 1: $\begin{bmatrix} L_{N+1,22} & \\ L_{N+1,32} & L_{N+1,33} \end{bmatrix} \leftarrow \mathcal{P}^{1/2}$                ▷ potrf
 2: for $n \leftarrow N \rightarrow 0$ do
 3:     $\mathcal{L}_{n+1}' \mathcal{A}_n \leftarrow L_{n+1,22}' \begin{bmatrix} B_n & A_n & b_n \end{bmatrix} + \begin{bmatrix} 0 & 0 & L_{n+1,32}' \end{bmatrix}$   ▷ trmm
 4:     $\mathcal{M}_n \leftarrow \mathcal{Q}_n + (\mathcal{L}_{n+1}' \mathcal{A}_n)'(\mathcal{L}_{n+1}' \mathcal{A}_n)$   ▷ syrk
 5:     $\begin{bmatrix} L_{n,11} & & \\ L_{n,21} & L_{n,22} & \\ L_{n,31} & L_{n,32} & L_{n,33} \end{bmatrix} \leftarrow \mathcal{M}_n^{1/2}$   ▷ potrf
 6: end for
 7: if $PI = 1$ then
 8:     $\pi_0 \leftarrow L_{0,22}(L_{0,22}' x_0 + L_{0,32}')$   ▷ trmv
 9: end if
10: for $n \leftarrow 0 \rightarrow N$ do
11:     $u_n \leftarrow -(L_{n,11}')^{-1}(L_{n,21}' x_n + L_{n,31}')$   ▷ gemv & trsv
12:     $x_{n+1} \leftarrow A_n x_n + B_n u_n + b_n$   ▷ gemv
13:     if $PI = 1$ then
14:         $\pi_{n+1} \leftarrow L_{n+1,22}(L_{n+1,22}' x_0 + L_{n+1,32}')$   ▷ trmv
15:     end if
16: end for
```

3.3 Solution of the (factorized) KKT system

In a predictor-corrector IP, the corrector step is computed by solving a system of linear equations (in the form (5)) that has the same left hand side as the system giving the predictor step, but a different right hand side. And similarly, in mixed precision we need to solve multiple systems with the same left hand side.

An efficient algorithm to solve (5) for the LQCP exploiting the already factorized l.h.s. matrix can be obtained by exploiting the analogy between Algorithm 1 and the classical Riccati recursion, i.e. that $L_{n,22}$ is the lower triangular Cholesky factor of the Riccati recursion matrix P_n. The algorithm is presented in Algorithm 2. The cost of the algorithm is $N(8n_x^2 + 8n_x n_u + 2n_u^2)$ flops, plus eventually $N(2n_x^2)$ if π is needed.

Algorithm 2 Solution of (factorized) LQCP

```
 1: $p_N \leftarrow p$
 2: for $n \leftarrow N \to 0$ do
 3:     $P_{n+1}b_n \leftarrow L_{n+1,22}L'_{n+1,22}b_n + p_{n+1}$                ▷ trmv
 4:     $\begin{bmatrix} l'_n & p'_n \end{bmatrix}' \leftarrow \begin{bmatrix} s'_n & q'_n \end{bmatrix}' + \begin{bmatrix} B'_n & A'_n \end{bmatrix} \cdot (P_{n+1}b_n)$    ▷ gemv
 5:     $l_n \leftarrow L_{n,11}^{-1} l_n$                                      ▷ trsv
 6:     $p_n \leftarrow p_n - L_{n,21} l_n$                                     ▷ gemv
 7: end for
 8: if $PI = 1$ then
 9:     $\pi_0 \leftarrow L_{0,22}L'_{0,22}x_0 + p_0$                          ▷ trmv
10: end if
11: for $n \leftarrow 0 \to N$ do
12:     $u_n \leftarrow -(L'_{n,11})^{-1}(L'_{n,21}x_n + l_n)$                ▷ gemv & trsv
13:     $x_{n+1} \leftarrow A_n x_n + B_n u_n + b_n$                          ▷ gemv
14:     if $PI = 1$ then
15:         $\pi_{n+1} \leftarrow L_{n+1,22}L'_{n+1,22}x_0 + p_{n+1}$          ▷ trmv
16:     end if
17: end for
```

3.4 Residuals computation

To solve a system of linear equations using mixed precision, we need a routine for the computation of the residuals, that in the solution of (5) are defined as

$$\begin{bmatrix} r_g \\ r_b \end{bmatrix} = -\begin{bmatrix} H & -A' \\ -A & 0 \end{bmatrix} \begin{bmatrix} z^* \\ \pi^* \end{bmatrix} - \begin{bmatrix} g \\ b \end{bmatrix}.$$

If system (5) was solved exactly, the residuals would be zero. However, because of the finite precision of the computations, in practice residuals are generally not zero. An algorithm for the computation of the residuals for LQCP is presented in Algorithm 3. The cost of the algorithm is $N(6n_x^2 + 8n_x n_u + 2n_u^2)$ flops.

Algorithm 3 Residuals of LQCP

```
1: $r_{s,0} \leftarrow -(S_0 x_0 + R_0 u_0 + B'_0 \pi_1 + s_0)$          ▷ gemv & symv & gemv
2: $r_{b,0} \leftarrow x_1 - \left( \begin{bmatrix} B_0 & A_0 \end{bmatrix} \begin{bmatrix} u_0 \\ x_0 \end{bmatrix} + b_0 \right)$      ▷ gemv
3: for $n \leftarrow 1 \to N-1$ do
4:     $\begin{bmatrix} r_{s,n} \\ r_{q,n} \end{bmatrix} \leftarrow \begin{bmatrix} \pi_n \\ 0 \end{bmatrix} - \left( \begin{bmatrix} R_n & S_n \\ S'_n & Q_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} + \begin{bmatrix} B'_n \\ A'_n \end{bmatrix} \pi_{n+1} + \begin{bmatrix} s_n \\ q_n \end{bmatrix} \right)$
                                                              ▷ symv & gemv
5:     $r_{b,n} \leftarrow x_{n+1} - \left( \begin{bmatrix} B_n & A_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} + b_n \right)$      ▷ gemv
6: end for
7: $r_{q,N} \leftarrow \pi_N - (P x_N + p)$                                  ▷ symv
```

3.5 Implementation details

For each algorithm, we implemented two versions: one calling BLAS, and the other calling tailored linear algebra routines written in C using the following HPC techniques: see Frison et al. [2014] for more details.

Blocking for registers. This is the single most important technique, and can be used on all machines. It has a double aim: reduce the number of memory operations, and hide latency of floating-point operations. On modern architectures, the CPU is much faster than the main memory: as a consequence the cost of a memop is much higher that the cost of a flop. A hierarchy of smaller and faster memories (registers, caches) is used to mitigate this difference in speed, and the programmer should reuse data already present in faster memories. As an idea, blocking is a technique that consist of loading a sub-matrix in a certain memory level ($\mathcal{O}(n^2)$ memops), to perform the required operation on that sub-matrix ($\mathcal{O}(n^3)$ flops for level-3 BLAS). In this way, the ratio flops/memops is increased. In our implementation we only block for registers, since for matrices too large to fit in cache BLAS is high-performing, and thus we can switch to the version calling BLAS. Blocking for registers is also used to hide the latency of operations: for example, on most Intel machines floating-point add and mul are pipelined and can be issued every clock cycle, but their result is available after respectively 3 and 5 clock cycles.

SIMD instructions. SIMD (Single-Instruction Multiple-Data) are instructions that perform the same operation in parallel on all elements of small vectors of data: this reduces the number of operations, and can improve performance up to n_v times for small vectors of size n_v. Nowadays many architectures implement SIMD instructions: as an example, Intel and AMD have the SSE instructions (that operates on 2 doubles or 4 floats at a time) and AVX instructions (that operates on 4 doubles or 8 floats at a time). The size of the small vectors suggests that, using SIMD instructions, the theoretical performance in single precision is twice as much as the theoretical performance in double precision. The drawback is that usually SIMD are more difficult to program, and they have alignment requirements: SSE instructions can efficiently load and store data that is 16 bytes aligned, while for AVX instructions the alignment requirement is 32 bytes. The alignment requirements limit the possibility for a compiler to use SIMD. We explicitly deal with alignment requirements in the IP methods, such that the data passed to LQCP solvers is already aligned.

Customized BLAS. In our LQCP solvers, we need only a small subset of BLAS, and then there is no need to implement it all. The innermost loop of each linear-algebra routine is implemented as a separate micro-kernel, hand optimized using block for registers and SIMD intrinsics. Furthermore, in the code used for this paper the size of all matrices is fixed at compile time: this reduces the number of branches, and allows the compiler to further optimize.

Single/double/mixed precision. On the target architecture one SIMD instruction can operate on twice as many floats as doubles. This, together with the fact that floats occupy half the space in memory (including registers and caches) and use half the memory bandwidth, gives that the performance in single precision is about twice the performance in double precision. Hence the reason for using single precision whenever possible. Mixed precision iterative refinement is a technique that allows to solve a system of linear equations exploiting the higher performance of single precision in the most expensive parts while maintaining the double precision of the final result, see Buttari et al. [2007]. A mixed precision algorithm for the solution of LQCP is presented in Algorithm 4. The algorithm can be seen as an iterative algorithm, where the l.h.s. factorized in single precision is used as a good preconditioner. Our numerical tests show that in most cases 1 iterative refinement step is enough to have almost double precision. For small systems, the mixed precision algorithm is slower than the double precision one, due to the cost of the additional solutions and residuals computations; anyway, for large systems the performance is close to the single precision one.

Algorithm 4 Factorization and solution of LQCP (mixed precision)

```
1: factorize and solve LQCP in single precision using Algorithm 1 with $PI = 1$, obtaining $(x, u, \pi)$
2: for $it\_ref \leftarrow 1 \rightarrow IT\_REF\_MAX$ do
3:     compute the residuals in double precision using Algorithm 3, obtaining $(r_s, r_q, r_b)$
4:     Solve LQCP in single precision using Algorithm 2 with $PI = 1$ and $(s, q, b) = (r_s, r_q, r_b)$ as r.h.s, obtaining $(\Delta x, \Delta u, \Delta\pi)$
5:     update the solution in double precision $(x, u, \pi) \leftarrow (x, u, \pi) + (\Delta x, \Delta u, \Delta\pi)$
6: end for
```

4. IP METHODS FOR THE LINEAR MPC PROBLEM

The linear MPC problem (3) is an instance of the general quadratic program

$$\begin{aligned} \min_{z} \quad & \tfrac{1}{2} z' H z + g' z \\ s.t. \quad & Az = b \\ & Cz \geq d \end{aligned}$$

that can be solved by means of an interior-point (IP) method. In this paper, we consider the primal-dual IP and Mehrotra's predictor-corrector primal dual IP (in the following, predictor-corrector IP), see Nocedal et al. [2006] for details about the algorithms.

4.1 Primal-dual IP method

In the primal-dual IP, at each iteration k of the IP method it has to be solved a system of linear equations in the form

$$\begin{bmatrix} H + C'(T_k^{-1}\Lambda_k)C & -A' \\ -A & 0 \end{bmatrix} \begin{bmatrix} y_k \\ \pi_k \end{bmatrix} = \\ = - \begin{bmatrix} g - C'(\Lambda_k e + T_k^{-1}\Lambda_k d + T_k^{-1}\sigma\mu_k e) \\ b \end{bmatrix} \tag{6}$$

where t_k are the slack variables, π_k and λ_k are the Lagrangian multipliers of the equality and inequality constraints, μ_k is the duality measure, σ is a centering parameter and e is a vector on ones. In case of the linear MPC problem, (6) is the KKT system of an instance of the LQCP (1), see Rao et al. [1998]. This means that (6) can be solved using Algorithm 1.

4.2 Predictor-corrector IP method

In case of the predictor-corrector method, at each iteration of the IP method two systems of linear equations have to be solved, respectively for the computation of the predictor and of the corrector search directions. These systems are similar to (6), and differ only for the right hand side: this means that the factorization has to be performed only once, and that they can be solved respectively using Algorithm 1 and Algorithm 2.

4.3 Inexact IP methods

In Fig. 1 we show the result of a convergence test for the duality measure in case of single, double and mixed precision used in the computation of the search direction, for both primal-dual and predictor-corrector IP methods. The fact that the single precision solution behaves as the higher precision ones till approximately 10^{-6} suggest that we can implement an inexact IP method (proposed for MPC problems by Shahzad et al [2010], with MINRES to compute the search direction), where the inexact search direction is computed by solving the LQCP subproblems using Algorithm 1 in single precision, exploiting the higher performance of single precision computation. Numerical tests shows that a value of the duality measure of 10^{-5} is a good threshold value between single and higher precision.

5. NUMERICAL RESULTS

In this section we test the HPMPC toolbox, that is our implementation of the solvers family presented in this paper. In a first part, we compare different implementations of the proposed solver for the LQCP; in the second part, we assess the performance of the proposed IP methods for the linear MPC problem. The test problem is the mass-spring system, see Domahidi et al. [2012].

The test machine is a laptop equipped with the processor Intel Core i7 3520M @ 2.9 GHz (up to 3.6 GHz in turbo mode), running Linux Xubuntu 13.04. The compiler is gcc 4.7.3. All tests are performed on one core.

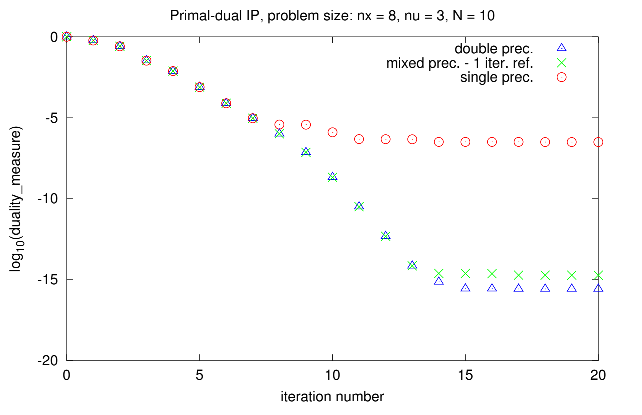
In Frison et al. [2014] we have already shown that the approach based on SSE and AVX micro-kernels gives high-performance on a number of Intel and AMD architectures.

5.1 LQ control problem

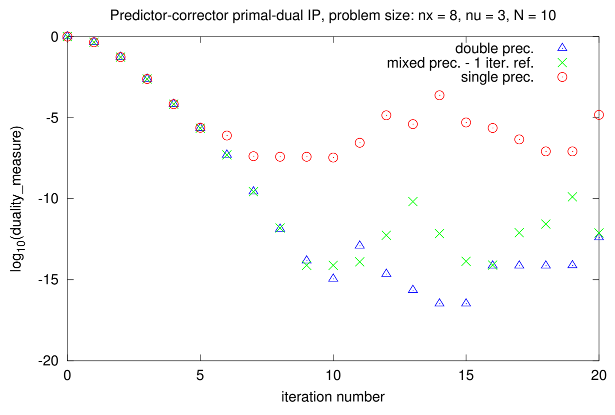
In this part we compare different approaches to implement the solver for LQCP proposed in Algorithm 1: a version making use of an highly-optimized BLAS (OpenBLAS version 0.2.6), a version using customized BLAS-like routines and AVX micro-kernels, and a version using customized BLAS-like routines and triple-loops, all of them in double, single and mixed precision with 1 iterative refinement step.

The results are in Fig. 2, where we assess the performance in Gflops of the different implementations. The processor theoretical peak performance in single precision is computed as 3.6 GHz * 2 floating-point instructions per clock (one add and one mul) * 8 flops per floating-point instruction (single precision, AVX instructions) = 57.6 Gflops. In double precision, AVX instructions can perform 4 flops per floating-point instruction, so the peak performance is the half, 28.8 Gflops.

The use of an highly-optimized BLAS library gives high-performance only for large systems, since it needs to perform a number of operations (e.g. copies of data in contiguous and aligned memory, blocking for different memory levels) that heavily impact performance for small matrices, while are well amortized for large ones. As a result, the performance is really poor for small systems. The code is implemented making explicit use of SIMD instructions, so the performance in single precision is



(a) Primal-dual IP.



(b) Predictor-corrector IP.

Fig. 1. Convergence test for the proposed IP methods, where the search direction is computed in single, double or mixed (with 1 iterative refinement step) precision. The test problem is the mass-spring system with $n_x = 8, n_u = 3, N = 10$. The combination of single precision above $\mu = 10^{-5}$ and higher (double or mixed) precision below is an inexact IP method that gives the same solution as an exact IP method, but requires less computation time.

higher that in double; the cross-over between mixed and double precision is around $n_x = 60$.

The triple-loop based approach can reach only a small fraction of the peak performance (even if the loops size is fixed at compile time), and the obtained performance is almost identical in single and in double precision. As a consequence, it can outperform BLAS only for very small systems. Furthermore, there is no advantage in using mixed precision, that would be always worse that double.

The proposed AVX micro-kernel based approach can attain a large fraction of the peak performance in both single and double precisions for a wide range of problem sizes. For small problems, this approach outperforms both optimized BLAS and triple-loop bases approach, and the performance increases quickly with the problem size. The maximum performance is attained at $n_x = 160$ in double precision (19.89 Gflops, 69% of peak) and $n_x = 128$ in single (40.19 Gflops, 70% of peak) and mixed (34.37 Gflops, 60% of peak) precisions. For larger problems, there is a certain degradation in performance, since memory footprint exceeds cache size, and our code does not perform blocking for cache. Anyway, for this size BLAS is high-performing, and the algorithm calling BLAS can be used instead.

5.2 Linear MPC problem

Here we assess the performance of the different IP methods for the linear MPC problem (3). We tested exact IP methods in single, double and mixed, and inexact ones in single+double and single+mixed precisions, where the threshold between single and higher precisions is $\tau = 10^{-5}$, for both primal-dual and predictor-corrector IP methods.

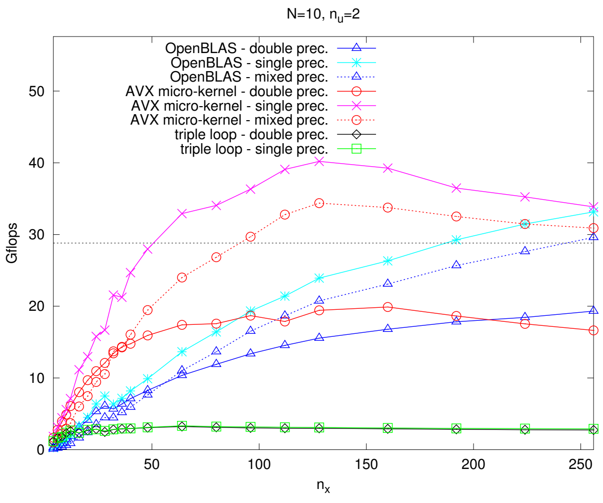
The results are in the table in Fig. 3. Since the factorization of the KKT matrix is the most expensive part in IP algorithms, the behavior of the IP methods closely resembles the behavior of Algorithm 1 in the different precisions. Single precision is always the fastest. Among higher precisions, the best results are usually obtained for inexact IP methods with the combination single+double for small problems, and single+mixed for large problems. For the largest problem, the use of inexact IP method and mixed precision requires a computational time slightly larger than the single precision, with the same accuracy as the double precision.

Whether primal-dual IP or predictor-corrector IP is the most efficient choice is problem dependent: the one has a lower cost per iteration, the other requires less iterations. Anyhow, in general primal-dual IP may be the best choice for small problems, and it takes more advantage of mixed precision computation.

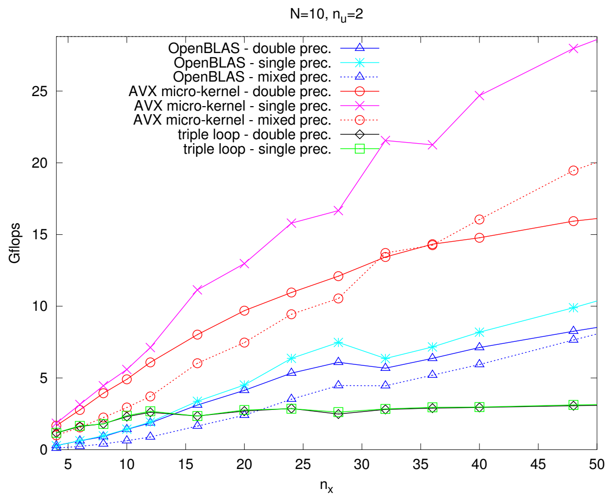
Comparing the results in the table in Fig. 3 with the ones in Domahidi et al. [2012], we can see that the solvers of our HPMPC solvers family are several times faster than state-of-the-art solvers such as FORCES, CVXGEN and CPLEX, and that the performance gap increases with the problem size.

6. CONCLUSION

In this paper, we presented an efficient algorithm for the solution of the linear-quadratic control problem (LQCP). The fact that this algorithm performs only few function calls to linear-algebra routines was exploited to implement them using high-performance computing techniques, such as blocking for registers, SIMD instructions and customized BLAS. These high-performance routines were used as building blocks in solvers for the LQCP in single, double and mixed precision. In turn, the LQCP solvers were used as routines in IP methods, and in particular we proposed the use of inexact IPs where the inexact search direction is obtained solving the LQCP in single precision. This approach gives a solution in double precision, while exploiting the higher performance of single precision computation on modern architectures. An implementation of these solvers, HPMPC, is several times faster than state-of-the-art solvers for MPC. As future work, we plan to add multi-thread support, and optimize the code for embedded architectures (e.g. Intel Atom, ARM, PowerPC).



(a) Large scale.



(b) Small scale.

Fig. 2. Performance of different implementations of the proposed LQCP solver: triple-loop based, micro-kernel based and optimized BLAS based, in single, double and mixed precision. Figure (a) (respectively (b)) is scaled along the y axis to have theoretical single (respectively double) precision peak performance at turbo frequency at the top of the picture, 57.6 Gflops (respectively 28.8 Gflops).

			HPMPC: primal-dual IP					HPMPC: predictor-corrector IP					FORCES#	
n_x	n_u	N	s	d	m	s+d	s+m	s	d	m	s+d	s+m	s	d
4	1	10	0.04	0.04	0.06	**0.04**	0.04	0.05	**0.05**	0.10	0.05	0.08	0.08	0.11
8	3	10	0.09	0.09	0.16	**0.09**	0.11	0.14	**0.14**	0.26	0.14	0.20	0.29	0.33
12	5	30	0.41	0.51	0.71	**0.44**	0.48	0.60	0.71	1.13	**0.65**	0.84	1.67	2.00
22	10	10	0.41	0.55	0.62	**0.45**	0.47	0.53	0.70	0.91	**0.62**	0.73	2.90	3.25
30	14	10	0.80	1.09	1.16	**0.90**	0.92	0.99	1.34	1.63	**1.18**	1.32	6.70	7.23
60	29	30	10.05	18.23	14.11	12.45	**11.23**	11.58	20.35	18.42	16.16	**15.02**	153.02	143.80
90	44	30	29.80	60.04	42.23	38.53	**33.57**	32.67	65.59	52.67	50.54	**43.05**	*	*

Fig. 3. Proposed primal-dual and predictor-corrector IP methods (run time in [ms] for 10 iterations, averaged over 100 random initial states): exact IP methods in single (s), double (d) and mixed (with 1 iterative refinement step) (m) precision; inexact IP methods in double (s+d) and mixed (with 1 iterative refinement step) (s+m) precision, where the threshold between single and higher precisions is $\mu = 10^{-5}$. Bold represents the fastest high-precision solver for each problem size. Notice that the first 6 problems are taken from Domahidi et al. [2012]: the proposed predictor-corrector IP method is from 2 (1st problem) to about 10 (6th problem) times faster than FORCES, that in turns is faster that CPLEX and CVXGEN. # FORCES code is compiled using gcc 4.6.3, with optimization flags -O2 -mavx -funroll-loops. * The code for the larger problems could not be downloaded, since the connection to the server drops due to the long code generation time.

REFERENCES

Bemporad, A., Morari, M., Dua. V., Pistikopoulos, E.N. (2002). The Explicit Linear Quadratic Regulator for Constrained Systems. *Automatica*.

Buttari, A., Dongarra, J., Langou, J., Langou, J., Luszczek, P., Kurzak J. (2007). Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems. *The International Journal of High Performance Computing Applications*, 21(4), 457-466.

Domahidi, A., Zgraggen, A., Zeilinger, M.N., Morari, M., Jones, C.N. (2012). Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control. *proc. IEEE CDC 2012*

Frison, G., Jørgensen, J.B. (2013). Efficient Implementation of the Riccati Recursion for Solving Linear-Quadratic Control Problems. *proc. IEEE MSC 2013.*

Frison, G., Sørensen, H.H.B., Dammann, B, Jørgensen, J.B. (2014). High-Performance Small-Scale Solvers for Linear Model Predictive Control. *proc. IEEE ECC 2014.*

Jørgensen, J.B. (2005). *Moving Horizon Estimation and Control.* Ph.D. thesis, Department of Chemical Engineering, Technical University of Denmark, Denmark.

Mattingley, J, Boyd, S. (2012) CVXGEN: a Code Generator for Embedded Convex Optimization. *Optimization and Engineering*, 12(1):1-27.

Nocedal, J., Wright, S.J. (2006) *Numerical Optimization.* Springer, New York, 2nd edition.

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

Shahzad, A., Kerrigan, E.C., Constantinides, G.A. (2010). A Fast Well-conditioned Interior Point Method for Predictive Control. *proc. IEEE CDC 2010.*

Wang, Y. and Boyd, S. (2008). Fast model predictive control using online optimization. *proc. IFAC W.C.*