Dominant Speed Factors of Active Set Methods for Fast MPC

M. Herceg, C. N. Jones, and M. Morari

preprint submitted in July 2014
to Optimal Control Applications and Methods

Abstract

The paper presents a review of active set (AS) algorithms that have been deployed for implementation of fast model predictive control (MPC). The main purpose of the survey is to identify the dominant features of the algorithms that contribute to fast execution of online MPC and to study their influence on the speed. The simulation study is conducted on two benchmark examples where the algorithms are analyzed in the number of iterations and in the workload per iteration. The obtained results suggest directions for potential improvement in the speed of existing AS algorithms.

1 Introduction

The move toward fast MPC algorithms has been a challenge in the control community over the past ten years [29]. In this period several interesting methods have been presented which we believe have defined significant trends in optimization computing. The developed algorithms can be classified in two parts:

- explicit MPC,
- fast online MPC.

In the literature, one can find papers that review the progress of optimization techniques used in fast MPC. Examples are [2] for a survey on explicit MPC and [13] which summarizes the progress in the field of fast nonlinear MPC. A detailed study on the complexity and convergence properties of gradient methods used in fast MPC can be found in [40]. For further results on gradient methods the reader is referred to [28, 35]. We will not repeat the survey on gradient methods, rather, the interest here is in AS methods where convergence is obtained in a relatively small number of iterations. The aim of this paper is to review some of the recent contributions to active set optimization algorithms in the field of linear online MPC where the speedup factor is reported in orders of magnitude.

In the category of AS algorithms we review the following approaches: the dual gradient projection of [5], the online active set strategy of [16], and the quasi-Newton approach of [38]. Furthermore, we study ideas of interior-point methods that have been applied for fast MPC in [46] and [14].

The emphasis is given on identifying the common factors that are crucial to the speed of the algorithm. The reviewed methods have been applied to solve MPC problems on two benchmark examples and the obtained results are compared in the number of iterations and estimated workload per iteration. The conducted analysis shows that the gradient projection algorithm combines most of the identified speedups and that there still exists space for further improvements.

In the first part of the paper we study the structure of the MPC problem that is common to all reviewed algorithms. Subsequently, the selected methods are briefly introduced and the dominant features are highlighted. The third part provides a comparative study that shows how much benefit in terms of speedup can be obtained. The last part of the paper studies code optimization and suggests directions to achieve effective implementation in the encoding process.

2 Structure of Finite Horizon MPC Problem

2.1 Problem Formulation

The principle of the MPC strategy is to determine a control action based on the actual measurement and the prediction of a plant evolution to a near future. The control input is obtained by formulating and solving an optimal control problem. The feedback is achieved by repetitive solution of the optimization problem for varying feedback data. The formulation of the optimal control problem remains identical between sampling instances and the feedback variables enter the problem formulation as parameters θ. The feedback variables are typically the plant measurements or estimates. Without going into details on various formulations of MPC problems, which can be found in the literature, e.g. [8, 32], the focus is given to the final form as it is passed to the appropriate numerical solver. In the following we adopt the notation commonly used in the optimization community [3, 37]. The vector $x \in \mathbb{R}^n$ denotes the decision variables and $\theta \in \mathbb{R}^d$ is the vector of time-varying feedback variables. In the linear MPC case the underlying optimization problem often boils down to the following quadratic program (QP):

$$\min_{x} \quad \frac{1}{2} x^T H x + c^T(\theta) x \tag{1a}$$

$$Ax \leq b(\theta) \tag{1b}$$

where $H \in \mathbb{R}^{n \times n}$, $H \succ 0$, $c(\theta) \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b(\theta) \in \mathbb{R}^m$ are the matrices of the problem data that depend on the plant model and the particular MPC setup. The data depending on the feedback variables is evident in terms $c(\theta)$, and $b(\theta)$. The problem formulation (1) takes into account the plant model,

open-loop prediction, and constraints. This so called dense representation can be obtained using techniques described in [37, Chapter 15] or in [26, Chapter 7.2] for general optimization problems containing equality constraints. In the sequel we will refer to the formulation (1) because the reviewed methods work with the dense form.

2.2 Optimality Conditions

The goal of any fast MPC method is to achieve a solution to the problem (1) as quickly as possible such that the obtained result can be certified with respect to feasibility and optimality. An optimal solution to the problem (1) is characterized with a pair of decision variables x^* and Lagrange multipliers w^*, which must satisfy the optimality conditions

$$Hx^* + c(\theta) - A^T w^* = 0 \tag{2a}$$
$$w^{*T}(Ax^* - b(\theta)) = 0 \tag{2b}$$
$$Ax^* - b(\theta) \leq 0 \tag{2c}$$
$$w^* \geq 0 \tag{2d}$$

The set of rows that are active at the optimum for inequality constraints (2b) is denoted as an active set $\mathcal{A} := \{i \in 1, \ldots, m \mid A_i x^* - b_i(\theta) = 0\}$, and its complement is denoted as the inactive set $\mathcal{I} := \{i \in 1, \ldots, m \mid A_i x^* - b_i(\theta) < 0\}$. The optimal multipliers corresponding to inactive constraints $w^*_{\mathcal{I}}$ are equal to zero, otherwise non-negative. In case the set of inequalities in $\mathcal{A}$ is linearly dependent at the optimum, only the multipliers in the subset $W \subseteq \mathcal{A}$ are positive. The set $\mathcal{W}$ that corresponds to linear independent inequalities at the optimum is referred to as working set. The remaining variables can be obtained by solving the following system of equations

$$\underbrace{\begin{pmatrix} H & A_{\mathcal{W}}^T \\ A_{\mathcal{W}} & 0 \end{pmatrix}}_{K_{\mathcal{W}}} \begin{pmatrix} x^* \\ -w^*_{\mathcal{W}} \end{pmatrix} = \begin{pmatrix} -c(\theta) \\ b_{\mathcal{W}}(\theta) \end{pmatrix} \tag{3}$$

The equation (3) is referred to as the KKT system and the matrix $K_{\mathcal{W}}$ is denoted as the KKT matrix that correspond to the working set $\mathcal{W}$. Note that all the terms that depend on the feedback variables are located on the right hand side of the system. This special block structure of the KKT system is very important in the reviewed fast MPC algorithms. Since H is positive definite and $A_{\mathcal{W}}$ is chosen to be linearly independent, the solution can be computed. In the literature there are several approaches to solve the KKT system that have proved to be effective in practice, see [7] for a detailed overview. In the next section we will focus on some of these approaches that exploit specific properties arising in MPC and can be applied to solve the KKT system.

3 Active Set Methods Employed in Fast MPC

The methods that will be reviewed in this section rely on the basic techniques to solve the KKT system plus they add specific information arising from the MPC setup and experience with closed loop control to improve convergence. The added ingredients are typically the special block-wise structure of the KKT matrix, the changing right hand side of (3), and the estimate of the optimal active set $\mathcal{A}$ or optimal solution from the previous iteration.

In this study we consider the finite horizon formulation of MPC, which can contain polyhedral constraints on the primal variables. This assumption is important because it excludes some of the fast optimization methods (e.g. fast gradient methods) that tackle MPC formulations with only lower/upper bounds on the decision variables. Furthermore, we assume that the prediction horizon is greater than one, which ensures the block-wise structure of the KKT matrix that makes it different from a standard QP.

3.1 Primal Active Set Method

The crucial element in the primal AS method [3, 37] is that the working set $\mathcal{W}$ in (3) changes with one index over iterations, which allows one to deploy the recursive factorization technique that uses rank one modifications of the factors obtained in the previous iteration [9, App. C]. The rank modification relates to an expression $K_{\overline{\mathcal{W}}} = K_{\mathcal{W}} \pm yy^T$ where $K_{\mathcal{W}}$ is the KKT matrix corresponding to the working set $\mathcal{W}$, $K_{\overline{\mathcal{W}}}$ is the matrix with the new working set $\overline{\mathcal{W}}$, and y consists of columns of K in the set $\overline{\mathcal{W}} \setminus \mathcal{W}$. The computational effort of the recursive approach thus depends on the index changes in the working set.

As rank one modifications are cheap to compute, the consequence is that the primal AS method is very effective in practice. Usually, a primal AS method needs many iterations to converge to the optimum, but at low cost.

In the literature there exist approaches that can improve the speed of the AS methods dramatically. For instance, the *warm start* technique that uses the primal/dual solution of the previous iterate to initialize the algorithm is a popular approach. It has been shown in [50] that if the initial estimates of the feasible solution and active set are determined using an off-line analysis, then the AS method can be terminated in few iterates.

The performance of warm starting can be improved if additional information is carried from the solution of the previous optimization problem. In particular, *hot starting* is a technique that also provides internal matrix factors, as well as primal/dual solution. We refer to a recent study of [24] that investigates the effects of hot starts in the MPC concept. Furthermore, [23] reviews specific AS solvers that can hot-start the algorithm providing internal LU, Hessian, and scale factors from the previous solution, e.g. SQOPT [18] solver, BPQD [1, 17], and SQIC solver [19].

Dual AS methods have been applied in the MPC framework, for instance the QPC solver of [48]. For further results, including a comparison with other optimization methods we refer the reader to the study of [38].

3.2 QPspline Method

QPspline is an abbreviation of regularized Newton method for finding a stationary point of a convex piecewise quadratic spline function. Introduced by [31] this method combines ideas from AS and gradient based methods to find the minimizer of a convex quadratic problem over a polyhedral set. The favorable properties of this approach are finite termination, fast convergence, and suitability for large-scale MPC as shown in [38]. Furthermore, the method is able to deal with degenerate cases using a mechanism for dropping redundant constraints.

The dominant speed element in the QPspline algorithm is that the active set changes over multiple indices and the authors suggested to use rank one updates for solution of the KKT system (3). As the rank one updates are cheap to compute, this approach combines the efficiency of AS method with smaller number of steps to reach the optimum. However, one disadvantage is that the approach is restricted to formulations that are feasible by construction.

3.3 Dual Gradient Projection

The dual gradient projection algorithm of [5] relies on two main steps. The first main step is to solve the projected line search exploiting the simple bounds on dual variables and the second main step is to improve the solution by solving the KKT system (3). The second step is determined by repetitively solving the KKT system that varies over iterations depending on the dimension of the working set $\mathcal{W}$. The important fact here is that the active set is allowed to change over multiple indices and the method thus achieves fast convergence to the optimum at in relatively small number of iterations. In addition, the method is capable of solving degenerate cases as well.

Multiple index changes motivated the authors to solve the resulting KKT system (3) using Riccati recursions. The approach is related to the name of Riccati because the recurrence relations take a form of Riccati equation that arises in the unconstrained optimal regulation problem. This technique was introduced by [39] and it applies to a sparse MPC formulation (1) with equality constraints. The principle is to solve the KKT system block-wise in the dimension of individual blocks. Using this type of special recursive factorization technique the authors achieved essential improvements in the speed of the dual gradient projection algorithm.

3.4 Online Active Set Strategy

The online active set strategy is an approach by [16] that is applicable when numerous related QPs are solved sequentially, which is the case for MPC. The method exploits the fact that the MPC problem (1) is solved repetitively where the objective function (1a) and the right hand side of the constraints (1b) do not change much with the parameters θ. The appealing speedup arises in cases when the optimal active set does not change significantly with the change in the

parameters of the subsequent problem. The method has a finite termination property under primal non-degeneracy.

The dominant property here is that the method needs few iterations to achieve the optimal solution because the algorithm is warm started from the solution of the preceding MPC problem. In particular, the method proceeds by working set changes in the direction given by difference in the parameters until the new optimal working set is found. Since the working set is modified over iterations with one index, a recursive factorization based on rank-one updates is applied that gives the method fast convergence with a low cost per iteration.

4 Dominant Speed Factors

In the previous section various methods for solving optimization problems involved in fast MPC have been briefly reviewed. The crucial elements of the methods have been identified that make the algorithms run of the markedly faster in the MPC setting than in a standard QP setting. We can now group the main ingredients of fast MPC algorithms as follows:

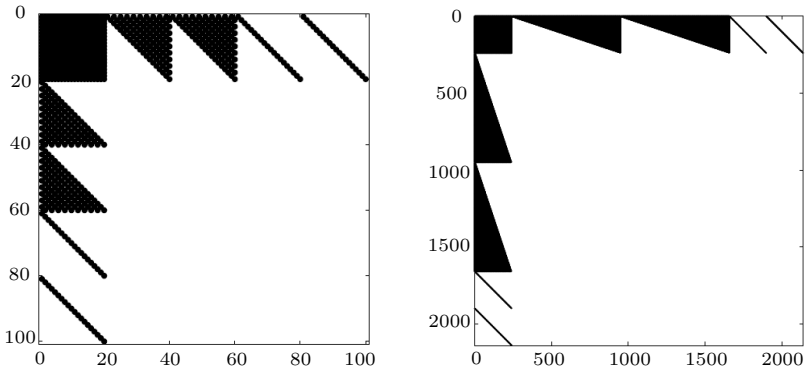
- warm start (estimates of the active set and primal/dual variables) and hot start (estimates of the internal factors)
- linear algebra tailored to MPC (specific KKT solvers, parametrization)

In the next section we focus on the evaluation of these ingredients on particular MPC examples. The goal is to find a method that exploits most of these ingredients and provides an optimality certificate with a low cost and in the least number of steps. Furthermore, we study additional possible improvements and suggest efficient implementation using existing software tools.

4.1 Warm Start

To compare the methods, the QPspline and the dual gradient projection algorithms have been implemented in Matlab. The online active set strategy has been downloaded as qpOASES solver [16] and the primal AS method is available in Optimization Toolbox of Matlab under `quadprog` function. The numerical performance of the methods is compared on two benchmark examples that have been obtained from a collection of benchmark data used as a test set for the development of qpOASES.

The focus of the numerical experiments is to compare the reviewed AS methods for solving MPC problems from a convergence point of view and estimate the workload per iteration. In particular, the performance of these methods is compared in the number of iterations and in the maximum rank change in the recursive factorization scheme. The workload that relates with the recursive factorization is given by rank updates mechanism and observing the maximum change in the active set provides an estimate of the workload per iteration. This information is important in QPspline and dual gradient projection method because these two algorithms feature multiple index sets over iterations. On the



(a) Diesel engine example (b) Chain of masses example

Figure 1: The sparsity pattern of KKT matrices for two examples.

contrary, both the AS method and the online active set strategy allow only one index change per iteration so there is no need to observe the active set changes. Total estimated workload can be then computed as a product of the maximum change in the active set and the number of iteration.

The control aspects are not important in this study because all methods give the same optimal control law and closed loop profiles.

4.1.1 Diesel Engine Example

Consider the diesel engine benchmark [15, Sec. 6] that corresponds to a tracking control problem subject to input constraints. The optimization problem formulated using finite horizon MPC comprises of $n = 20$ variables and $m = 80$ inequality constraints. The MPC feedback is computed within 0.05s sampling time with the control horizon of length 5. The optimization problem has been converted to a dense form and all equality constraints have been eliminated. The dimension of the KKT matrix is $n + m = 100$ and it has a dense structure, which is shown in Fig. 1(a).

The control objective is to follow the desired reference signal while satisfying the constraints on the control inputs. The simulation of the closed loop control is conducted on the time scale of 30 seconds where two reference changes occur. The first change is done at 10s and the second at 20s of the experiment. Only these changes are interesting from the optimization point of view because the constraints on inputs become active. Otherwise, the remaining part of the scenario the MPC results in an unconstrained control law.

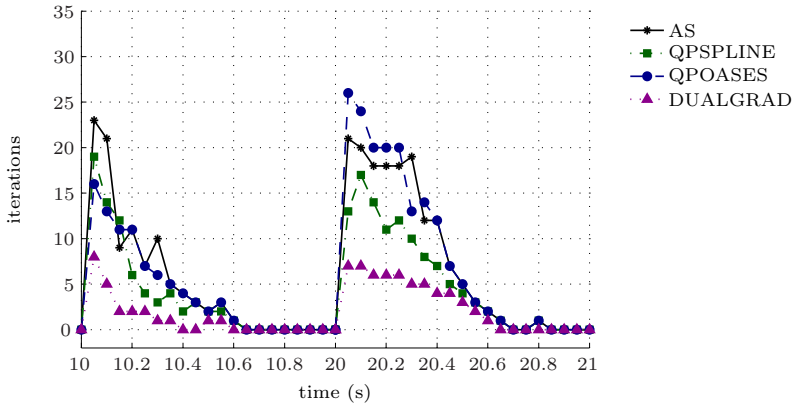
Fig. 2 shows the number of iterations needed for four optimization methods to converge. The cold start refers to a case where no initial guess is provided to start up the algorithm and in the warm start the primal/dual solution of the previous optimization problem is used. In order for the results to be visible,

note that the time scale spans two important changes around 10s and 20s in one chart.

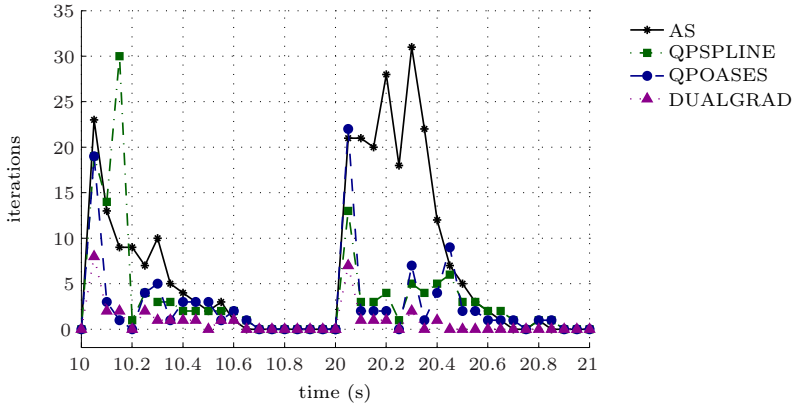
Despite the fact that the qpOASES solver has not been designed with cold starting in mind, here we want to point out how important is the warm starting technique in the MPC setting. Practically, in all algorithms that have been applied to MPC, the primal/dual solution of the previous optimization problem is always supplied as initial guess in the subsequent optimization problem, which reduces the iterations, as it can be seen in Fig. 2. In the warm start results one can observe dramatic reduction in the number of iterations immediately after the step response, which is a consequence of supplying primal/dual estimates. However, this effect is not so visible in the primal AS method of QUADPROG solver where the iterations did not improve much in the warm start because only estimate in the primal solution has been provided.

To estimate the workload per iteration needed by each compared method, we have observed the maximum changes in the working set for QPspline and dual gradient projection methods. This was not needed in the active set method and the online active set strategy because here the maximum change in the active set is always by one index. The results obtained in Fig. 3 show that in the cold start the maximum number of changes is similar in both methods whereas in the warm start the dual gradient projection does not need that many changes. The result in Fig. 3 again demonstrates the importance of warm starts in fast AS methods employed in MPC because less workload per iteration can be achieved. It is worth mentioning here, that both cold and warm start have the same number of changes in the working set as well as the same number of iterations in the first moment after the reference change, i.e. at 10.05s and 20.05s. This is because the initial guess from the solution of the previous optimization problem is the same as the default guess and the closed loop system with MPC controller is excited the most at these times.

Fig. 4(a) and Fig. 4(b) show the comparison of the active set methods with respect to the estimated workload per iteration at the times when the closed loop system was excited the most. The y-axis indicates the maximum change in the working set that corresponds to the maximum effort involved in the recursive factorization while the number of iterations is shown on the x-axis. From the figures in can be read that the active set method and online active set strategy need more iterations to converge but at low cost due to rank one modifications. In contrast, QPspline and dual gradient projection method need less iterations but at the expense of increased cost per iteration. The comparison in Fig. 4(a) and Fig. 4(b) is a trade-off curve, which shows that for the considered scenario the dual gradient projection algorithm achieves the best performance in terms of iterations and estimated cost per iteration. The plots in Figs. 4(a)-4(b) could be interpreted as a criterion to select the suitable optimization method for fast MPC that achieves the least iteration steps with the least workload per iteration.



(a) Cold start



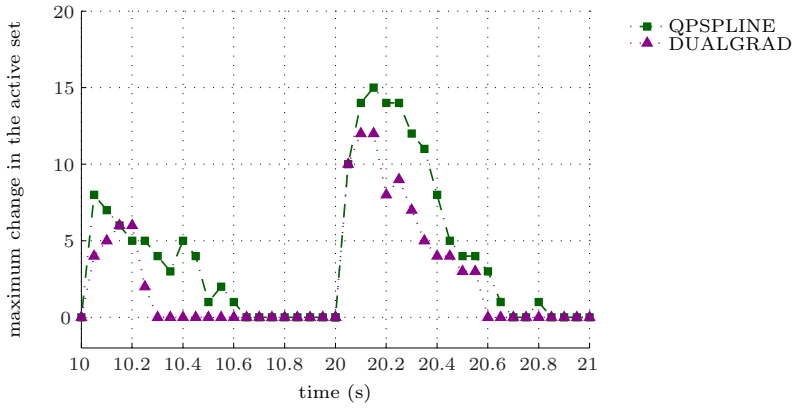
(b) Warm start

Figure 2: The number of iterations needed to find the MPC control law in a tracking setup for a diesel engine example.

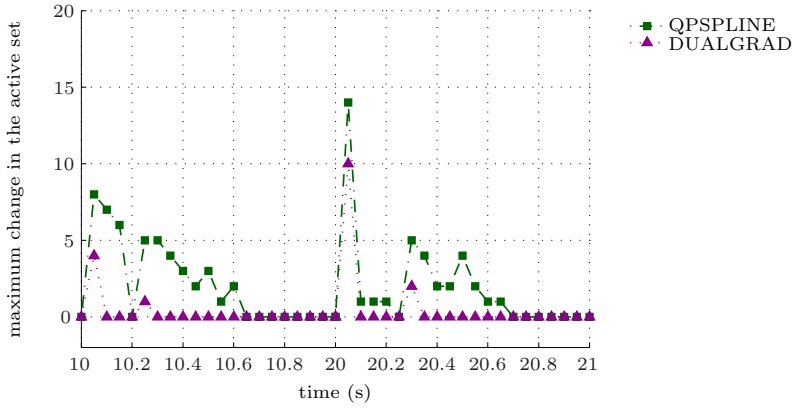
4.1.2 Chain of Masses Example

The control problem in this benchmark example is to stabilize a chain of masses that operate under state and input constraints using the MPC approach [49]. The optimization problem is formulated and converted to a dense form that consists of $n = 240$ variables and involves $m = 1898$ linear inequality constraints. The structure of the KKT matrix is shown in Fig. 1(a) and it has a dimension of $n + m = 2138$.

The plant is controlled with a sample time of 0.2s and the prediction horizon



(a) Cold start

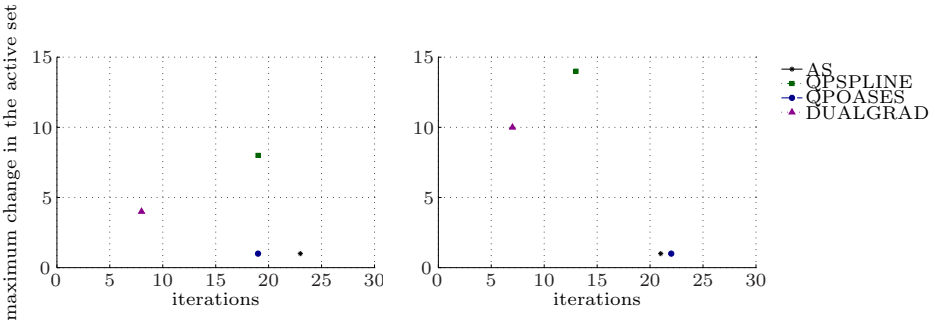


(b) Warm start

Figure 3: The maximum change in the working set for a diesel engine example.

comprises of 80 steps. The objective of the control strategy is to reject a strong perturbation that is applied to the system such that the closed loop system satisfies state and input constraints. The test scenario is conducted over 30s but in the following we depict only first 8s because in the remainder of the simulation no constraints are active.

The number of iterations needed for the active set methods to terminate is depicted in Fig. 5. Similarly as in the diesel engine example, from the plot it is evident that the warm starting technique helps to reduce the number of iterations rapidly. However, this is not so evident in the primal active set method where the performance is similar in cold and warm start, because only the estimates in the primal solution are reused. From the comparison in Fig. 5, the



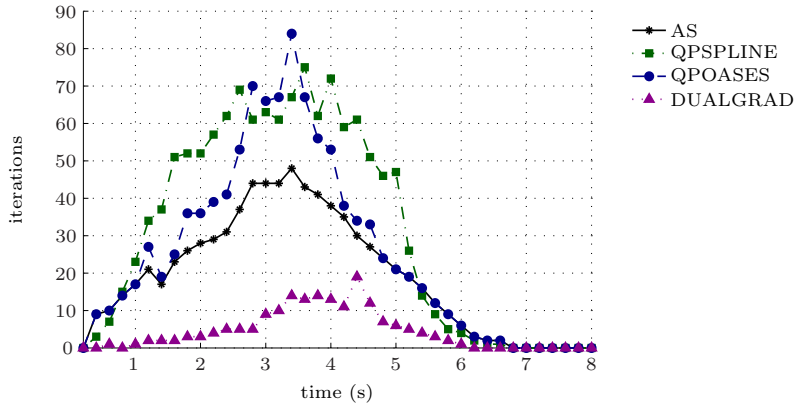
(a) Data at the time 10.05s. (b) Data at the time 20.05s.

Figure 4: The estimated workload per iteration evaluated at different times for a diesel engine example.

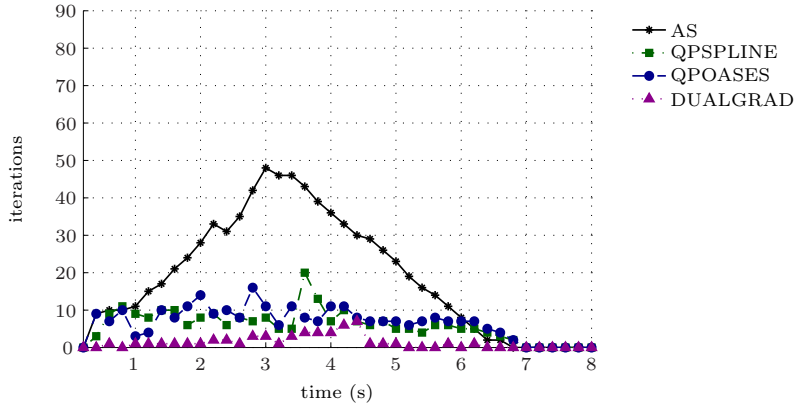
dual gradient projection method achieves the minimum number of iterations.

The maximum change in the active set can be observed in Fig. 6 for the QPspline method and the dual gradient projection. The plot in Fig. 6 represents the maximum rank change involved in the recursive factorization of the KKT matrix and corresponds to a worst case workload per iteration. In the cold start both methods perform roughly the same number of changes, but in the warm start the dual gradient projection algorithm operates with smaller changes. This implies that the maximum rank change is at most 11 for the dual gradient projection method and 32 for the QPspline method while the size of the full KKT matrix is 2138. The results depicted in Fig. 6 emphasize that the workload per iteration can be strongly influenced by warm starting.

In Fig. 5 one can determine that at the time of 3s and 4s the closed loop system is excited the most because all methods exhibit a maximum number of iterations at this point. In particular, Fig. 7(a) and Fig. 7(b) depict the maximum change in the working set versus the iteration count in the warm start. Active set method performs at most one index change and therefore the method needs the most iterations to converge. Similarly, the online active set strategy does at most one index change but since it exploits effectively the solution of previous optimization problem the the number of iterations is reduced. The dual gradient projection method needs the least number of iterations with slightly increased cost due multiple index changes. The QPspline algorithm needs also few iterations to converge, but due to more index changes the cost is higher than in the dual gradient projection method. The results shown in Fig. 7(a) and Fig. 7(b) indicate the dual gradient projection algorithm as the approach that benefits the most from the warm starting technique. Furthermore, comparing the plots of the two examples in warm start, i.e. Figs. 4(a)–4(b) and Figs. 7(a)–7(b) speak for dual gradient projection and qpOASES method for implementation because both approaches were able to find the optimal solution in the least number of iterations with the minimum computational effort when the control systems were excited the most.



(a) Cold start

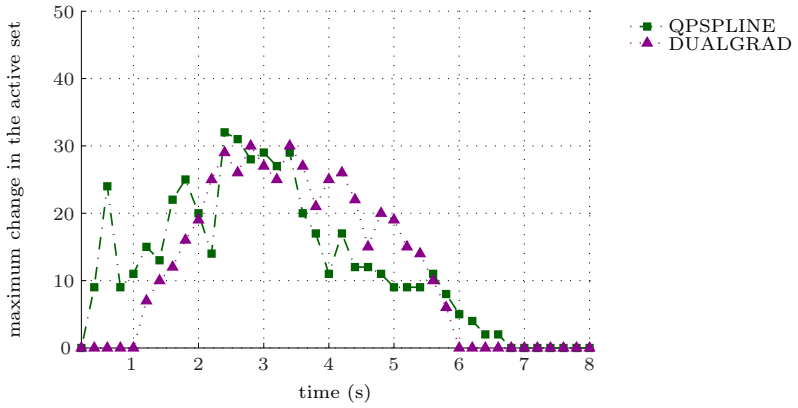


(b) Warm start

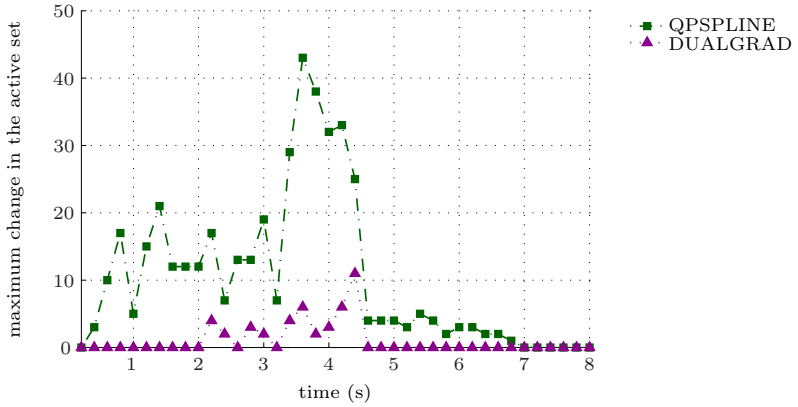
Figure 5: The number of iterations needed to find the MPC control law in a regulation setup for a chain of masses example.

4.1.3 Summary

In this section we have compared AS methods to observe the convergence and estimate the computational complexity per iteration. The results obtained on two MPC benchmark examples demonstrate that the effect of warm start plays a major role in fast optimization algorithms. However, it depends on the particular method how efficient is the initial guess implemented to warm start the algorithm. In particular, it has been observed that an initial estimate based on both primal and dual variables shows better performance than just an estimate



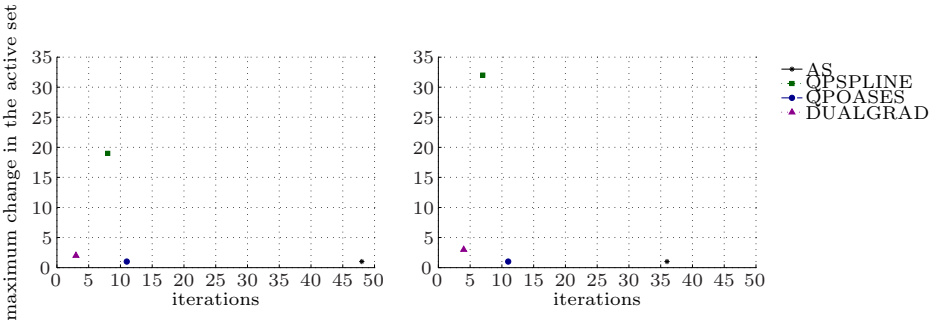
(a) Cold start



(b) Warm start

Figure 6: The maximum change in the working set for a chain of masses example.

in the primal variables. Furthermore it has been shown on two MPC examples that the active set methods, which allow multiple changes in the active set, require less iterations and converge faster than primal active set solver but with slightly increased cost per iteration. In particular, for the considered examples the dual gradient approach shows to benefit the most from the warm starting technique and achieves fast convergences with a relatively small workload per iteration.



(a) Data at the time of 3s.

(b) Data at the time of 4s.

Figure 7: The workload per iteration for a chain of masses.

4.2 Tailored Linear Algebra

4.2.1 Solution Methods

In the literature there are several approaches to solve the KKT system that have proved to be effective in practice, see [7] for a detailed overview and [23, 33, 37]. In the following, we will briefly summarize the main methods that are generally applied to solve KKT systems.

The methods for solving KKT systems can be classified in two categories:

- coupled (direct)
  - full-space methods
  - iterative methods
- segregated (indirect)
  - range-space methods
  - null-space methods

The coupled methods solve the KKT system (3) for all variables in the dimension of the KKT system, whereas the segregated methods solve the equations separately for primal variables x and dual variables w in the reduced dimension.

Full-space methods solve the system (3) by factoring the KKT matrix while exploiting sparsity and symmetry patterns. Typically, the matrix K is factorized using techniques that return a triangular matrix at the output. The triangular structure is then utilized to solve the system of equations with forward and backward substitutions. The LDL, LBL, and LU types of factorization are the most common techniques in this case.

Iterative methods deploy recurrence relations in variables x and w that quickly converge to the solution. Given the initial guess x_0, w_0 and the iterator

$k = 0, 1, \ldots$, the update equations are usually defined as

$$\begin{aligned} x_{k+1} &= x_k + \alpha Q_H^{-1}(-c - Hx_k + A_{\mathcal{W}}^T w_{\mathcal{W},k}) && \text{(4a)} \\ w_{\mathcal{W},k+1} &= w_{\mathcal{W},k} + \omega Q_{\mathcal{W}}^{-1}(-A_{\mathcal{W}} x_{k+1} + b_{\mathcal{W}}) && \text{(4b)} \end{aligned}$$

where Q_H^{-1}, $Q_{\mathcal{W}}^{-1}$ are the preconditioning matrices that influence the convergence rate, and $\alpha > 0$, $\omega > 0$ are the relaxation parameters. There are very efficient algorithms in this class that can be applied, such as general minimal residual, quasi-minimal residual, and preconditioned bi-conjugate gradient stabilized method. For special symmetric cases there exist conjugate gradient algorithms, minimal residual method, symmetric LQ method, and simplified quasi-residual method.

Range-space method require $H \succ 0$ and solve (3) in two steps:

$$\begin{aligned} A_{\mathcal{W}} H^{-1} A_{\mathcal{W}}^T w_{\mathcal{W}} &= A_{\mathcal{W}} H^{-1} c + b_{\mathcal{W}} && \text{(5a)} \\ Hx &= A_{\mathcal{W}}^T w_{\mathcal{W}} - c && \text{(5b)} \end{aligned}$$

The dual variables w are determined solving the reduced system (5a) and their values are substituted in (5b) to get the primal variables x. This, also called the Schur complement approach, is suitable when H is positive definite, well conditioned, and the number of active constraints $|\mathcal{W}|$ is small.

Null-space method introduces a matrix $Z \in \mathbb{R}^{n \times |\mathcal{W}|}$ that satisfies the following equation

$$A_{\mathcal{W}} Z = 0 \tag{6}$$

Finding the matrix Z allows to compute the primal solution using the relation

$$x = Zv + \hat{x} \tag{7}$$

where $v \in \mathbb{R}^{|\mathcal{W}|}$ and $\hat{x} \in \mathbb{R}^n$ is a particular solution of

$$A_{\mathcal{W}} \hat{x} = b_{\mathcal{W}} \tag{8}$$

After $\hat{x}$ has been determined via (8) it is possible to solve

$$Z^T H Z v = Z^T(-c - H\hat{x}) \tag{9}$$

in variables v and determine x per (7). The second step is to compute w via

$$A_{\mathcal{W}} A_{\mathcal{W}}^T w_{\mathcal{W}} = A_{\mathcal{W}}(c + Hx) \tag{10}$$

which can be solved effectively because the left hand side matrix is symmetric and positive definite. The approach is suitable when the number of active constraints $|\mathcal{W}|$ is small as in range-space method. Both range-space and null-space methods operate with dense positive definite matrices, thus an effective way to solve such a subsystems is via Cholesky factorization or iterative conjugate gradient method.

4.2.2 Recursive Factorization

It has been observed by many authors that the tailored factorization of the KKT system arising in optimal control can yield significant benefits in the speed of MPC algorithms. Examples are [5, 27, 39, 46] and more detailed studies can be found in the following doctoral theses [12, 23, 33, 42]. Specifically, if the KKT system to be solved comes from a formulation of an optimal control problem, the matrix to be factorized has block-wise structure. The computational aspects involving both sparse and dense formulations have been studied in [45] on a chain of masses example that shows that a sparse block-wise factorization approach scales better in solving MPC problems over large horizons. This result again indicates that specially tailored block-wise factorization techniques suit fast MPC implementations. At this point we can summarize that the block-wise factorization dominates in sparse MPC formulations whereas dense formulations are driven by rank one updates.

A very detailed analysis of KKT solvers that focuses on a comparison of recursive Riccati type- versus coupled solvers is is given in [42, Chap. 3, 5]. The obtained results revealed that a structured Riccati-solver is very effective in solving large scale optimization problems arising in optimal control and the particular speed-up factor depends on the formulation of the MPC problem.

The focus of this section is to review methods to solve the KKT system (3) that deal with recursive factorization techniques and identify possible speedup factors. It has been shown in Sec. 4.1, that the dominant element in fast MPC methods is that the active set is allowed to vary over multiple indices, which significantly reduces the number of iterations to reach the optimum. Therefore, a reasonable implementation for these methods appears to be a combination between the recursive factorization based on rank one updates and direct full factorization, depending on how the active set changes between iterations. This has been observed by [4] in the dual gradient projection method because for larger changes in the active set the rank one updates may become expensive.

In the literature there are several methods on recursive factorization methods available and the following list summarizes the methods for which software tools exist:

- Block LDL updates[1], as suggested by [30, 36].
- Cholesky updates CHOLLRUP[2], CHOLMOD[3], and LINPACK[6].
- Block LU updates LUSOL[4], LUMOD[5]
- Recursive QR factorization[7].

The methods have been designed to solve effectively the KKT system for a varying working set. Most of the methods operate with rank one updates because the maximum change in the working set is one index, but CHOLLRUP and CHOLMOD assume multiple changes in the working set and work with multiple rank updates.

It has been studied in [12] that multiple rank updates provide computationally efficient solutions to the KKT system with multiple changes in the AS. Also, as shown in [11], multiple rank Cholesky factorization is twice as fast as the rank one update scheme. This is the motivation for the next section where we study the effect of multiple rank update versus rank one update on the speed of MPC examples.

4.2.3 Multiple Rank Updates

This section studies the problem of multiple rank updates for the use in recursive factorization of fast MPC algorithms. The rank modification scheme can be applied to coupled full-space methods as well as to segregated methods. The block LU approach of [23] appears as a suitable candidate for the full-space methods because the matrix $K_{\mathcal{W}}$ may not be positive definite. The application of the null-space method is described in [26]. In this study we employ the range-space approach where the system (5a) is factored using LDL factorization.

The LDL factorization routine searches for a lower triangular matrix $L_{\mathcal{W}\mathcal{W}}$ and a diagonal matrix $D_{\mathcal{W}\mathcal{W}}$ such that the product $L_{\mathcal{W}\mathcal{W}}D_{\mathcal{W}\mathcal{W}}L_{\mathcal{W}\mathcal{W}}^T$ is equal to a positive definite symmetric matrix $K_{\mathcal{W}\mathcal{W}} = A_{\mathcal{W}}H^{-1}A_{\mathcal{W}}^T$. The crucial step in the factorization is to find factors $L_{\overline{\mathcal{W}}}$, $D_{\overline{\mathcal{W}}}$ of matrix $K_{\overline{\mathcal{W}}}$ formed by rows/columns of the active set when it changes from the current set $\mathcal{W}$ to a new set $\overline{\mathcal{W}}$. Two types of active set changes are possible: index addition and deletion.

Consider the case where an index set $\mathcal{N}$ is to be appended to the current working set $\mathcal{W}$ to get new set $\overline{\mathcal{W}} = \mathcal{W} \cup \mathcal{N}$. The initial factors have been computed using direct factorization and are available as $L_{\mathcal{W}\mathcal{W}}D_{\mathcal{W}\mathcal{W}}L_{\mathcal{W}\mathcal{W}}^T = K_{\mathcal{W}\mathcal{W}}$. The LDL factorization of the new matrix $K_{\overline{\mathcal{W}}}$ can be put as follows

$$\underbrace{\begin{pmatrix} L_{\mathcal{W}\mathcal{W}} & 0 \\ F_{\mathcal{N}\mathcal{W}} & L_{\mathcal{N}\mathcal{N}} \end{pmatrix}}_{L_{\overline{\mathcal{W}}}} \underbrace{\begin{pmatrix} D_{\mathcal{W}\mathcal{W}} & 0 \\ 0 & D_{\mathcal{N}\mathcal{N}} \end{pmatrix}}_{D_{\overline{\mathcal{W}}}} \underbrace{\begin{pmatrix} L_{\mathcal{W}\mathcal{W}}^T & F_{\mathcal{N}\mathcal{W}}^T \\ 0 & L_{\mathcal{N}\mathcal{N}}^T \end{pmatrix}}_{L_{\overline{\mathcal{W}}}^T} = \underbrace{\begin{pmatrix} K_{\mathcal{W}\mathcal{W}} & K_{\mathcal{W}\mathcal{N}} \\ K_{\mathcal{N}\mathcal{W}} & K_{\mathcal{N}\mathcal{N}} \end{pmatrix}}_{K_{\overline{\mathcal{W}}}} \quad (11)$$

where the subscript denotes the particular rows/columns in the augmented matrix $K_{\overline{\mathcal{W}}}$. The submatrices that form the matrix $K_{\overline{\mathcal{W}}}$ are known while the new

[1] Parts of the software available at http://www2.imm.dtu.dk/~hbni/Software/.
[2] Available at http://lapmal.epfl.ch/software/index.shtml.
[3] Available at http://www.cise.ufl.edu/research/sparse/cholmod.
[4] Available at http://www.stanford.edu/group/SOL/software/lusol.html.
[5] Available at http://www.stanford.edu/group/SOL/software/lumod.html.
[6] Available at http://www.netlib.org/linpack.
[7] Available as a basic package in Matlab, http://www.mathworks.com.

factors $F_{\mathcal{NW}}$, $L_{\mathcal{NN}}$, and $D_{\mathcal{NN}}$ are to be determined. Applying the matrix multiplications of the terms in (11) one obtains formulas to compute the new parts, i.e.

$$(L_{\mathcal{WW}} D_{\mathcal{WW}}) F_{\mathcal{NW}}^T = K_{\mathcal{WN}} \tag{12a}$$

$$L_{\mathcal{NN}} D_{\mathcal{NN}} L_{\mathcal{NN}}^T = K_{\mathcal{NN}} - F_{\mathcal{NW}} D_{\mathcal{WW}} F_{\mathcal{NW}}^T \tag{12b}$$

where (12b) is referred to as a rank-$|\mathcal{N}|$ downdate. Given $L_{\mathcal{WW}}$, $D_{\mathcal{WW}}$ factors and matrix $K_{\overline{\mathcal{W}}}$, a lower triangular system (12a) is solved to obtain $F_{\mathcal{NW}}$. Then, LDL rank downdate is performed on the system (12b) with dimension of $|\mathcal{N}|$.

Consider a deletion of indices given in the set $\mathcal{N}$, i.e. $\mathcal{W} = \overline{\mathcal{W}} \setminus \mathcal{N}$. This case is simpler as addition because it does not need $K_{\overline{\mathcal{W}}}$ but exploits the information from existing factors $L_{\mathcal{NN}}$, $D_{\mathcal{NN}}$ and $F_{\mathcal{NW}}$ that are to be deleted. The new factors are obtained from

$$L_{\mathcal{WW}} D_{\mathcal{WW}} L_{\mathcal{WW}}^T = L_{\mathcal{NN}} D_{\mathcal{NN}} L_{\mathcal{NN}}^T + F_{\mathcal{NW}} D_{\mathcal{WW}} F_{\mathcal{NW}}^T \tag{13}$$

which is referred to as rank-$|\mathcal{N}|$ update.

From equations (11)–(13) it follows that the rank modifications are driven by the changes in the working set. The addition of indices leads to rank downdate and the deletion of indices to rank update. The efficiency of computing the $L_{\overline{\mathcal{W}}}$, $D_{\overline{\mathcal{W}}}$ factors thus depends on the changes in the working set and is measured in rank-$|\mathcal{N}|$ downdates and updates. For a more detailed explanation on the multiple rank update mechanism the reader is referred to [11].

4.2.4 Stability of Rank Updates

As mentioned in [12, p. 2], a positive definite matrix does not need any stability control because its factorization step is stable therefore the row/column permutation can be chosen purely to enhance sparsity. For a matrix that is not positive definite, there are efficient methods that preserve adequate stability such as threshold partial pivoting, threshold rook pivoting, threshold complete pivoting [12]. Furthermore, in [12, p. 3] is given that multi rank updates have the same degree of stability as rank-1 updates, that have the same stability characteristics as Gaussian elimination with partial or complete pivoting [12, p. 28]. For a more detailed discussion about stability see [12, Sec. 3.4].

4.2.5 Performance Comparison

This section addresses the efficiency of the multiple rank updates versus rank one updates for the use in fast MPC algorithms. From the discussion in the previous section it is expected that multiple rank updates perform better than rank-1 updates. This presumption will be studied in this section using CHOLMOD software, which is a part of `SuiteSparse`[1] package of sparse matrix algorithms for Matlab.

[1] http://www.cise.ufl.edu/research/sparse/SuiteSparse/

$\lvert\mathcal{N}\rvert$	5	10	15
rank-1 (ms)	3.408	5.777	8.595
rank-x (ms)	2.056	2.166	2.434

Table 1: Numerical values of CPU times in ms comparing rank update schemes for the diesel engine example.

$\lvert\mathcal{N}\rvert$	10	20	30	40
rank-1 (ms)	6.173	13.443	18.123	27.391
rank-x (ms)	2.223	2.619	2.643	3.145

Table 2: Numerical values of CPU time in ms comparing rank update schemes for the chain of masses example.

The simulation results have been obtained for the two MPC examples presented in Sec. 4.1 when factoring a symmetric positive definite matrix $K_{\mathcal{W}\mathcal{W}} = A_{\mathcal{W}} H^{-1} A_{\mathcal{W}}^T$ based on the transition from current working set $\mathcal{W}$ to a new set $\overline{\mathcal{W}}$. The index sets $\mathcal{W}$, $\overline{\mathcal{W}}$ are constructed in a way to simulate absolute changes in the working set, which is composed of rank updates and downdates in arbitrary order.

The CPU time is measured[2] that is needed to perform the following task of LDL factorization:

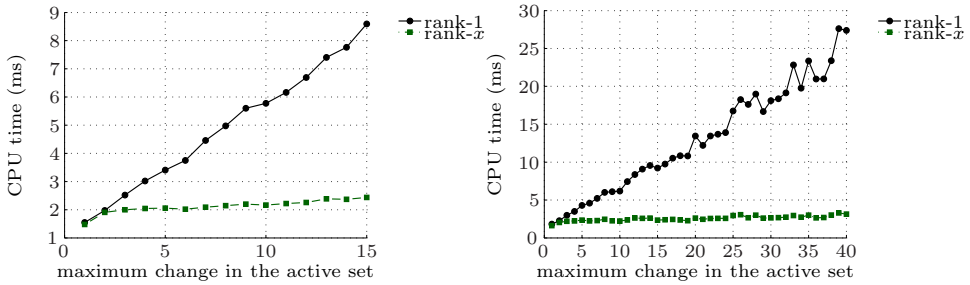
1. repeated rank-1 update in a loop,

2. multiple-rank updates,

The CPU time is measured over 50 calls and a median is taken to each task to give a realistic estimate. It should be noted that CHOLMOD works up to 8-rank updates. If the rank exceeds 8, then the update/downdate is done as a series of rank-k updates for each block of k columns.

Fig. 8 shows the CPU times needed for factorization of the KKT matrix versus the maximum changes in the working set. Important to note is the trend of recursive factorization based on rank updates with the increasing changes in the active set. Rank one update requires more CPU time to compute the L, D factors for large changes in the active set whereas the multiple rank update approach is slightly increasing. Both methods show the same performance only when the AS changes at by at most one index. From all figures is evident that multiple rank updates are faster than rank one if the AS changes by more than one index. The trend of the multiple rank updates depicted in Fig. 8 may seem constant with respect to the rank one updates but is, in fact, marginally growing. The particular values of the CPU time for the diesel engine example is given in Tab. 1 and for the chain of masses example in Tab. 2.

[2]Ubuntu Linux, Processor Intel Core2 Duo T6500, 2.10GHz, 4GB RAM, L2 cache 2048 KB



(a) Diesel engine example.

(b) Chain of masses example.

Figure 8: CPU time needed for factorization of the KKT matrix depending on the maximum change in the active set.

4.2.6 Summary

In this section we have considered the dense formulation of the MPC problem and investigated the speed of factoring the related matrix with respect to varying active set. The results have been conducted by applying the recursive LDL factorization that is common method for solving KKT systems in the range-space methods. The simulation study on two examples shows that the multiple rank update scheme is faster than the rank-one update, except for the case when the active set changes at most by one index where these two approaches are equal. Therefore, the active set methods that operate with multiple index changes, i.e. QPspline and the dual gradient projection could improve the speed by employing multiple rank scheme.

4.3 Code Optimization

One of the dominant factors that can influence the speed of optimization algorithms is code optimization. This factor has not been mentioned in the previous sections but is important when any of the reviewed methods is to be implemented in a low-level programming language for purpose of MPC.

The main motivation behind code optimization is that one can exploit the information available at the design phase to increase the code performance for a specific control application. For example, by taking the hardware constraints into account one can make the algorithm run more effective, which leads to significant implementation speedups. Information about the target hardware is easily accessible in the product manual and one can exploit this knowledge to design the optimized code that fits the particular application.

In this section we discuss the code optimization aspects for implementing the reviewed algorithms where time and size matters. The motivation for analysis

of the code optimization aspects is due to recent contributions to fast MPC algorithms that feature code generation, e.g. [14, 25, 34] and others that will be reviewed in the sequel.

4.3.1 Code and Hardware Factors

In the previous parts we have learned that specially tailored solvers, which exploit the structure of MPC problem are able to run faster than general purpose QP solvers. The conservativeness of general purpose solvers comes from a fact that these solvers have been designed and tuned to deal with general classes of practical problems where the structure can vary from sparse to dense, dimensions of variables/constraints are not known a-priori, and the problem data may be unscaled. The consequence is, that the general-purpose solvers have been extensively tested for numerous problems and can provide certificates of optimality for almost any problem data. This makes them very robust but at the expense of speed. Importantly, the sizes of input data are not known a priori in the general purpose QP solvers and they may change size with every new call to solver. However, for a specific control application the sizes of input data are known beforehand and one can exploit this information to get additional speedups.

Knowing the problem dimension is crucial when tackling memory management. Static memory allocation has major advantages over dynamic allocation. Namely, the amount of allocated memory can be estimated in advance, the memory blocks are automatically initialized to zero, the compiler can optimize much better over fixed sizes than with varying dimensions, and there is no overhead with allocating/deallocating functions. Furthermore, dynamic allocation in embedded systems can be risky because of low memory constraints, the worst case execution time is difficult to bound and potential segmentation faults.

Exact dimensions are favorable when writing code constructs that allows performance specific programming. Specifically, as shown in [10] one can adjust the code to the size of available memory given by the hardware. For instance, by applying blocking, loop merging, scheduling, and buffering techniques, the execution time of the compiled code can be much faster. Furthermore, with the recent progress in CPU architectures there is a space for extra improvement when taking into account vector instructions[1] and multi-threading due to multiple processor cores.

According to [20], there are options for optimization in the data storage scheme as well. For instance, in the C programming language the matrices are stored column-wise, which does not sacrifice performance if the size of the data fits into L2 cache [20, 44]. For larger amounts of data it has been shown in [21] that a block-wise storage can reduce cache misses and increase the performance markedly.

Therefore, hardware specifications determine some of the parameters needed

[1] See http://software.intel.com/en-us/avx/ for details.

for efficient code generation. The most important hardware factors are:

- total memory available,
- size of L1 and L2 caches,
- support of vector instructions,
- number of cores and parallelism.

With these pieces of information one can tailor the code to the given hardware and exploit the maximum performance out of it. This is so called *parameter-based performance tuning* and according to [10] there are three levels of hardware dependent optimization:

1. cache optimization
2. CPU and register level optimization
3. performance-conscious programming.

Knowing the hardware parameters exactly, one can directly plug the parameters to preprogrammed templates. If the parameters are not known, one can resort to self-tuning approach that runs a sequence of benchmark tests to estimate the parameters and use them consequently into prepared templates.

4.3.2 Available Software

Several software tools exist that provide optimized code. The basic principle is to exploit the additional information arising from the particular MPC setup and the target hardware to export the optimization algorithm to a lower-level programming language. This is referred to as code generation. In this part we will review some code generation tools that offer optimized linear algebra that is crucial to application of fast MPC algorithms.

- ATLAS[1] is an abbreviation of automatically tuned linear algebra software. It is a self-tuning software that estimates the hardware parameters based on recursive code generation and subsequent recompilation [47]. However, there are limitations regarding vector instructions and parallelism.
- FLAME (formal linear algebra methods environment) offers a library of dense linear algebra purely written in C. The code is parametrized by hardware constraints and uses blocking techniques for cache and CPU optimization. Furthermore, it supports parallelism, which proved to be more efficient comparing to traditional linear algebra libraries [43] and offers blockwise data storage management.

[1] Available at http://math-atlas.sourceforge.net/.

- ACADO toolkit is a tool for dynamic optimization and control features a code generation module [22]. Generated code uses own libraries and classes. The optimization is done on memory management level using static allocation and loop unrolling. Using this level of code optimization, the authors achieved solutions to sequential QPs in order of microseconds when applied to continuous stirred tank reactor case study.

- CVXGEN abbreviates code generation for convex optimization problems. The tool generates library-free code readily for deployment. CVXGEN does not depend on hardware parameters and allocates memory statically. Code optimization is implemented in LDL factorization routine that is a core step for obtaining a search direction when solving the sparse KKT system. Due to deployment of this tailored factorization the CVXGEN has become a popular tool in the control community because of its simplicity and applicability of interior point method to embedded systems [34]. However, the generated code can become large due to loop unrolling and thus the software is limited to a small sized applications.

- FORCES project aims at linking hardware parameters with code generation tailored to MPC problems [14]. It features various tricks for code optimization and the generated code depends on hardware parameters. Implemented are vectorized instructions, but at the moment the software lacks parallelism. FORCES has proved to be faster than CVXGEN when considering MPC application studies.

- FIORDOS is a software that generates C code for solving convex problems using the fast gradient method [25]. The generated code relies on static allocation and the core of the algorithm is a cheap implementation of the gradient projection step. The software has been verified on a power electronics application [41].

- Matlab Coder and Simulink Coder are the toolboxes of Matlab that generates C/C++ code from scripts and simulation blocks. The generated code takes care of the target hardware and optimizes the performance based on given hardware templates. With this approach the generated routines can be compiled either independently, or integrated with Matlab using matlab-executable interfaces. Matlab coder can be deployed for embedded systems, but the primal purpose is acceleration of Matlab scripts, simulations, and real-time control.

From the above discussion the dominant speed factors in the existing software packages are mostly due to the static allocation, loop-unrolling, and tailored linear algebra. Other optimization factors are typically not taken into account or the remainder is left on the compiler. With the FORCES project the effect of further code optimization is brought closer to hardware specification, but it still lacks some improvements, such as parallelism and blocked data storage. From the listed software tools, FLAME seems to offer the most techniques for code optimization. However, as the packages implement a subset of

available code optimization techniques, the complete speed capacity is still not fully exploited in combination with higher level MPC concept.

4.3.3 Effect of Loop Unrolling

In this part the effect of loop unrolling will be investigated. The motivation behind loop unrolling are two software packages (CVXGEN, ACADO) that use this technique to produce high speed code. The analysis is oriented on four code primitives that create hot spots in optimization algorithms, namely:

- matrix-vector product
- matrix-matrix product
- Cholesky factorization
- LDL factorization

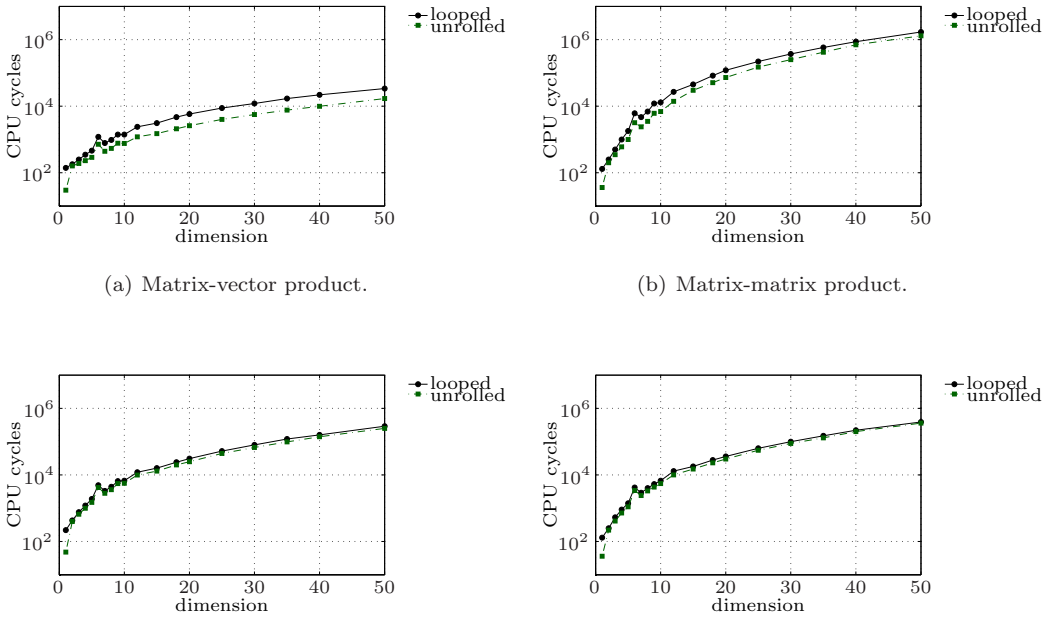
The speed of each routine is compared for two cases: if coded as a standard FOR loop and if coded elementwise (unrolled). Speed measurement is implemented via special RDTSC library that detects CPU cycles and is supported for Intel processors with CPUID instruction[2]. The experiment has been executed on Intel Atom processor Z530 (1 core, 2 threads), 1.6GHz, with cache size 512 KB. As given at the vendor's website, Intel Atom processor is used in small internet devices such as tablets, smartphones, netbooks, hybrids, consumer electronics, thus its architecture is similar to embedded systems.

The main routine for all computation has been written in C with functional calls to particular routines. The data of the matrices have been randomly generated. No specific code performance tuning is implemented. The data have been averaged over 50 calls and GCC compiler is deployed with `-O3` optimization flag. Results obtained from this experiment are depicted in Fig. 9. From the results it can be deduced that the effect of loop unrolling brings minor speedup and is not as significant as one would expect. Furthermore, the benefit of loop unrolling is limited by the code size and for larger dimensions this approach can become intractable. The most benefit in the speed appears due to static memory allocation and optimization performed by the compiler.

4.3.4 Summary

Using code optimization it is possible to provide fast algorithms tailored to specific applications. The key factor that makes the application run faster is the knowledge of the problem dimension that leads to improved memory management and better optimization performed by the compiler. The existing software packages provide the code optimization by exporting the fast MPC algorithm to a lower level programming language. The generation of such an optimized code

[2] http://www.intel.com/Assets/PDF/appnote/241618.pdf



(a) Matrix-vector product.

(b) Matrix-matrix product.

(c) Cholesky factorization.

(d) LDL factorization.

Figure 9: CPU cycles needed for execution of four code primitives in FOR loop and unrolled on Intel Atom machine.

exploits various tricks, including loop unrolling. It has been shown that the loop unrolling technique does not provide significant improvement in the speed of the algorithm for the considered examples and can become restrictive for large-scale MPC. Furthermore, it has been discussed that there exist a place for additional code optimization, e.g. when taking into account vectorized instructions, blocking techniques, and exploiting parallelism. The code optimization thus provides means for effective implementation of the reviewed AS methods and it can be exploited in order to improve the speed for fast MPC applications.

5 Conclusions

In this paper the fundamental parts of selected fast MPC algorithms have been reviewed. It has been identified that the dominant speed factors can be attributed to three ingredients: 1) warm start, 2) tailored linear algebra and 3) code optimization. Each of these ingredients come from the specific structure of the MPC optimization problem and the receding horizon principle that make fast MPC methods different from standard optimization algorithms. It has been

identified that the speed of the reviewed methods can be improved by applying special techniques in linear algebra and by code optimization to particular hardware. While the linear algebra part has been researched in depth in the control community, there exists a significant potential in code optimization utilizing the recent progress on parallel hardware architectures, which has not been thoroughly exploited in the MPC context. As already outlined as [6], this is a promising future research trend and the results in this survey are supporting this direction. The reviewed AS algorithms do not exploit the dominant speed factors to a full extent. According to results obtained with two benchmark examples, the dual gradient projection algorithm benefits from two of the dominant speedup factors. However, the efficiency of the method can be still improved with code optimization and export to a target hardware.

References

[1] http://www.mcs.anl.gov/~leyffer/solvers.html.

[2] Alessandro Alessio and Alberto Bemporad. A survey on explicit model predictive control. In L. Magni, D. Raimondo, and F. Allgöwer, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 345–369. Springer Berlin Heidelberg, 2009.

[3] A. Antoniou and W.-S. Lu. *Practical Optimization: Algorithms and Engineering Applications*. Springer, 2007.

[4] D. Axehill. *Integer Quadratic Programming for Control and Communication*. PhD thesis, Linköping University, Department of Electrical Engineering and Automatic Control, The Institute of Technology, 2008.

[5] D. Axehill and A. Hansson. A dual gradient projection quadratic programming algorithm tailored for model predictive control. In *Proc. of the 47th IEEE Conference on Decision and Control*, pages 3057–3064, 2008.

[6] D. Axehill and A. Hansson. Towards parallel implementation of hybrid MPC–a survey and directions for future research. In Rolf Johansson and Anders Rantzer, editors, *Distributed Decision Making and Control*, volume 417 of *Lecture Notes in Control and Information Sciences*, pages 313–338. Springer Berlin / Heidelberg, 2012.

[7] M. Benzi, G.H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[8] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control with for linear and hybrid systems*. preprint, 2012. http://www.mpc.berkeley.edu/mpc-course-material.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, U.K., 2004.

[10] S. Chellappa, F. Franchetti, and M. Püschel. How to write fast numerical code: a small introduction. In R. Lämmel, J. Visser, and J. Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II*, LNCS 5235, pages 196–259. Springer-Verlag, Berlin, Heidelberg, 2008.

[11] T.A. Davis and W.W. Hager. Multiple-rank modifications of a sparse cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 22(4):997–1013, 2001.

[12] L. Deng. *Multiple-Rank Updates to Matrix Factorizations for Nonlinear Analysis and Circuit Design*. PhD thesis, Stanford University, May 2010.

[13] M. Diehl, H.J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear MPC and moving horizon estimation. In Lalo Magni, Davide Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 391–417. Springer Berlin / Heidelberg, 2009.

[14] A. Domahidi, A. Zgraggen, M.N. Zeilinger, M. Morari, and C.N. Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *IEEE Conference on Decision and Control*, pages 668 – 674, Maui, HI, USA, December 2012. forces.ethz.ch.

[15] H.J. Ferreau. An online active set strategy for fast solution of parametric quadratic programs with applications to predictive engine control. Master's thesis, University of Heidelberg, 2006.

[16] H.J. Ferreau, H.G. Bock, and M. Diehl. An online active set strategy to overcome the limitations of explicit MPC. *International Journal of Robust and Nonlinear Control*, 18(8):816–830, 2008.

[17] R. Fletcher. Resolving degeneracy in quadratic programming. *Annals of Operations Research*, 46–47(2):307–334, 1993.

[18] P. E. Gill, W. Murray, and M. A. Saunders. *SQOPT 7 User's Guide*. Stanford University, Dept of Management Science and Engineering, 2006. http://www.stanford.edu/group/SOL/sqopt.htm.

[19] P.E. Gill and E. Wong. *User's Guide for SQIC*, Feb 2014. http://ccom.ucsd.edu/~optimizers/software.html#sqic.

[20] K. Goto and R. van de Geijn. On reducing TLB misses in matrix multiplication. FLAME Working Note #9. Technical Report TR-2002-55, The University of Texas at Austin, Department of Computer Sciences, November 2002.

[21] K. Goto and R.A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):1–25, 2008.

[22] B. Houska, H.J. Ferreau, and M. Diehl. An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, 47(10):2279–2285, 2011. http://www.acadotoolkit.org/.

[23] H. Huynh. *A Large-scale Quadratic Programming Solver Based on Block-LU Updates of the KKT System.* PhD thesis, Stanford University, California, USA, September 2008.

[24] T.C. Johnson, C. Kirches, and A. Wächter. An active-set quadratic programming method based on sequential hot-starts. *Optimization Online*, 2013. http://www.optimization-online.org/DB_HTML/2013/10/4084.html.

[25] C.N. Jones, A. Domahidi, M. Morari, S. Richter, F. Ullmann, and M.N. Zeilinger. Fast Predictive Control: Real-Time Computation and Certification. In *IFAC Conference on Nonlinear Model Predictive Control*, pages 94–98, Noordwijkerhout, the Netherlands, August 2012. fiordos.ethz.ch.

[26] C. Kirches. *Fast numerical methods for mixed-integer nonlinear model-predictive control.* Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden, Wiesbaded, 2011.

[27] C. Kirches, H.G. Bock, J.P. Schlöder, and S. Sager. A factorization with update procedures for a KKT matrix arising in direct optimal control. *Mathematical Programming Computation*, 3(4):319–348, 2011.

[28] M. Kögel and R. Findeisen. Fast predictive control of linear systems combining nesterov's gradient method and the method of multipliers. In *50th IEEE Conference on Decision and Control and European Control Conference*, pages 501–506, Dec 2011.

[29] J. Lee. Model predictive control: Review of the three decades of development. *International Journal of Control, Automation and Systems*, 9:415–424, 2011.

[30] W. Li and J. J. de Nijs. An implementation of the QSPLINE method for solving convex quadratic programming problems with simple bound constraints. *Journal of Mathematical Sciences*, 116(4):3387–3410, 2003.

[31] W. Li and J. Swetits. A new algorithm for solving strictly convex quadratic programs. *SIAM J. Optim.*, 7(3):595–619, August 1997.

[32] J. M. Maciejowski. *Predictive Control with Constraints.* Prentice Hall, 2002.

[33] C. Maes. *A Regularized Active-set Method for Sparse Convex Quadratic Programming.* PhD thesis, Stanford University, California, USA, November 2010.

[34] J. Mattingley and S. Boyd. *Automatic code generation for real-time convex optimization*, chapter Convex optimization in signal processing and communications. Cambridge University Press, 2009. http://cvxgen.com/.

[35] I. Necoara and V. Nedelcu. Rate analysis of inexact dual first order methods. application to dual decomposition. *IEEE Transactions on Automatic Control*, PP(99):1–1, 2013.

[36] H. Nielsen. AAFAC: A package of FORTRAN 77 subprograms for solving $a^T ax = c$. Technical Report Report NI 90-01, Institute for Numerical Analysis, Technical University of Denmark, Lyngby, Denmark, 1990.

[37] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, 1999.

[38] P. Patrinos, P. Sopasakis, and H. Sarimveis. A global piecewise smooth newton method for fast large-scale model predictive control. *Automatica*, 47(9):2016–2022, 2011.

[39] 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, December 1998.

[40] S. Richter. *Computational Complexity Certification of Gradient Methods for Real-Time Model Predictive Control*. PhD thesis, ETH Zurich, Zurich, Switzerland, November 2012.

[41] S. Richter, S. Mariéthoz, and M. Morari. High-speed online MPC based on a fast gradient method applied to power converter control. In *American Control Conference*, pages 4737 – 4743, Baltimore, MD, USA, June 2010.

[42] M.C. Steinbach. *Fast Recursive SQP Methods for Large-Scale Optimal Control Problems*. PhD thesis, University of Heidelberg, 1995.

[43] F.G. van Zee, E. Chan, R. van de Geijn, E. S. Quintana-Orti, and G. Quintana-Orti. Introducing: The LIBFLAME library for dense matrix computations. *IEEE Computing in Science & Engineering*, 11(6):56–62, 2009.

[44] Field G. van Zee. *LIBFLAME - the complete reference*. The University of Texas at Austin, 2011. http://www.cs.utexas.edu/users/flame/Spark/.

[45] M. Vukov, A. Domahidi, H.J. Ferreau, M. Morari, and M. Diehl. Auto-generated algorithms for nonlinear model predictive control on long and on short horizons. In *IEEE 52nd Annual Conference on Decision and Control*, pages 5113–5118, Dec 2013.

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

[47] R. Clint Whaley. *Automated empirical optimization of high performance floating point kernels*. PhD thesis, The Florida State University, 2004.

[48] A. Wills. QPC–quadratic programming in c, version 2.0, 2009. http://sigpromu.org/quadprog/.

[49] L. Wirsching, H.G. Bock, and M. Diehl. Fast nmpc of a chain of masses connected by springs. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 591 –596, oct. 2006.

[50] M.N. Zeilinger, C.N. Jones, and M. Morari. Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization. *IEEE Transactions on Automatic Control*, 56:1524 – 1534, July 2011.