

High-Performance Implementation of Solvers for MPC and MHE

Gianluca Frison, John Bagterp Jørgensen

Technical University of Denmark

CITIES WP-5 Workshop,
Grundfos, 9 December 2014

Model Predictive Control

- + easy incorporation of forecasts
- + predictive adaptation to setpoint changes
- + natural handling of constraints
- + generalization to non-linear systems
- optimization problem solved on-line

Moving Horizon Estimation

- + explicit use of multiple past measurements
- + natural handling of constraints
- + generalization to non-linear systems
- optimization problem solved on-line

Linear MPC Problem:

$$\min_{x,u} \sum_{n=0}^{N-1} \frac{1}{2} [u'_n \quad x'_n \quad 1] \begin{bmatrix} R_n & S_n & s_n \\ S'_n & Q_n & q_n \\ s'_n & q'_n & \rho_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix} + \frac{1}{2} [x'_N \quad 1] \begin{bmatrix} P & p \\ p' & \pi \end{bmatrix} \begin{bmatrix} x_N \\ 1 \end{bmatrix}$$

$$\text{s.t. } x_{n+1} = A_n x_n + B_n u_n + b_n$$

$$x_0 = \hat{x}_0$$

$$\underline{u}_n \leq u_n \leq \bar{u}_n, \quad n = 0, \dots, N-1$$

$$\underline{x}_n \leq x_n \leq \bar{x}_n, \quad n = 1, \dots, N$$

Problem size

- ▶ n_x : state vector dimension
- ▶ n_u : control vector dimension
- ▶ N : control horizon length

Solution techniques:

- ▶ explicit MPC
 - ▶ offline explicit solution (multiparametric QP)
 - ▶ online look-up table
 - ▶ exponential cost in N , n_x and n_u
- ▶ 1st order methods (gradient method)
 - ▶ online
 - ▶ many and data-dependent cheap iterations
 - ▶ matrix-vector operations (level 2 BLAS)
- ▶ 2nd order methods (Newton method)
 - ▶ online
 - ▶ few and data-independent expensive iterations
 - ▶ matrix-matrix operations (level 3 BLAS)

- ▶ Basic Linear Algebra Subprograms
- ▶ 'BLAS subroutines are a **de facto standard** API for linear algebra libraries and routines.' Wikipedia
- ▶ Usually optimized for **large scale** problems
- ▶ Divided into 3 levels
 - ▶ level 1: vector-vector operations: $\mathcal{O}(n)$ storage, $\mathcal{O}(n)$ flops
 - ▶ level 2: matrix-vector operations: $\mathcal{O}(n^2)$ storage, $\mathcal{O}(n^2)$ flops
 - ▶ level 3: matrix-matrix operations: $\mathcal{O}(n^2)$ storage, $\mathcal{O}(n^3)$ flops
- ▶ On modern processors, a **memop** is much more expensive than a **flop** \Rightarrow in level 3 BLAS there is a lot of space for optimization

- ▶ General QP program & KKT system

$$\begin{array}{ll}
 \min_{x,u} & \frac{1}{2}x'Hx + g'x \\
 \text{s.t.} & Ax = b \\
 & Cx \geq d
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{l}
 Hx + g - A'\pi - C'\lambda = 0 \\
 Ax - b = 0 \\
 Cx - d - t = 0 \\
 \lambda't = 0 \\
 (\lambda, t) \geq 0
 \end{array}$$

- ▶ Newton method (2nd order method) for the KKT system

$$\begin{bmatrix} H & -A' & -C' & 0 \\ A & 0 & 0 & 0 \\ C & 0 & 0 & -I \\ 0 & 0 & T_k & \Lambda_k \end{bmatrix}
 \begin{bmatrix} \Delta x \\ \Delta \pi \\ \Delta \lambda \\ \Delta t \end{bmatrix}
 = -
 \begin{bmatrix} Hx_k - A'\pi_k - C'\lambda_k + g \\ A\pi_k - b \\ Cx_k - t_k - d \\ \Lambda_k T_k \end{bmatrix}$$

- ▶ structured system, can be condensed as

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

- ▶ KKT system of an equality constrained QP

Linear-Quadratic Control Problem

Linear MPC Problem:

$$\min_{x,u} \sum_{n=0}^{N-1} \frac{1}{2} [u'_n \quad x'_n \quad 1] \begin{bmatrix} R_n & S_n & s_n \\ S'_n & Q_n & q_n \\ s'_n & q'_n & \rho_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix} + \frac{1}{2} [x'_N \quad 1] \begin{bmatrix} P & p \\ p' & \pi \end{bmatrix} \begin{bmatrix} x_N \\ 1 \end{bmatrix}$$

$$\text{s.t. } x_{n+1} = A_n x_n + B_n u_n + b_n$$

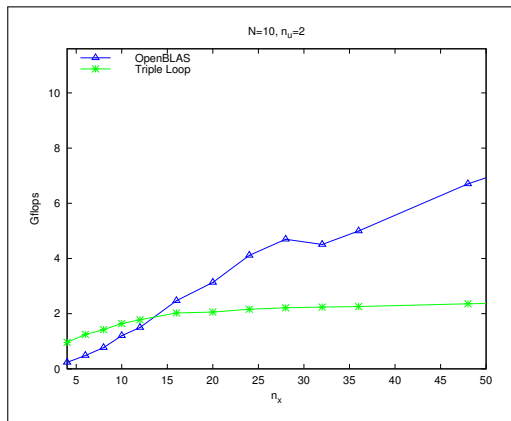
$$x_0 = \bar{x}_0$$

- ▶ used as routine to compute the search direction in Interior-Point methods
- ▶ most computationally expensive part of IP methods
- ▶ backward Riccati recursion used to factorize the KKT matrix

Implementation - current approaches

Riccati solver implementation

- ▶ code-generated **triple-loop** linear algebra works well for small problems
- ▶ optimized **BLAS** works well for large problems



Implementation - our approach

Naive approach

- ▶ use code-generated triple loop for small instances
- ▶ use BLAS for large instances

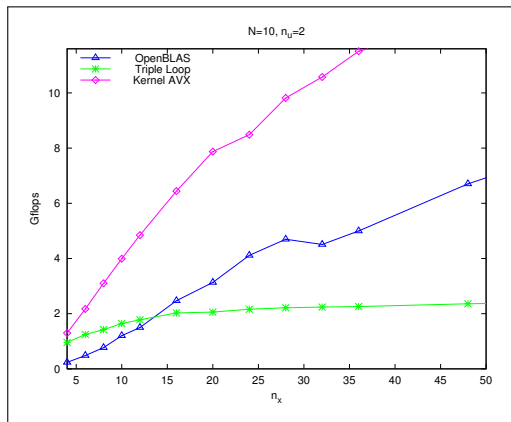
Better approach: **combine** the best of the two

- ▶ high-performing code (key routines (`gemm`) in optimized assembly: SIMD vectorization, hide operation latency)
- ▶ portability (most code is architecture-independent)
- ▶ small overhead (focus on small-scale problems)
- ▶ partial code generation (avoid branches & index computation)

Implementation - our approach

We got the best of the two approaches

- ▶ close-to-peak performance on all architectures
- ▶ performance increases quickly for small matrices



Test machine # 1

- ▶ Intel Core i7
3520M
- ▶ 2 cores, 4 threads
- ▶ AVX SIMD
- ▶ $\approx 346\$$ (processor
only)
- ▶ ≈ 18 W per core



Comparison with state-of-the-art solver for linear MPC

| n_x | n_u | N | HPMPC | | FORCES | |
|-------|-------|-----|--------|--------|--------|--------|
| | | | double | single | double | single |
| 4 | 1 | 10 | 0.05 | 0.05 | 0.11 | 0.08 |
| 8 | 3 | 10 | 0.14 | 0.14 | 0.33 | 0.29 |
| 12 | 5 | 30 | 0.71 | 0.60 | 2.00 | 1.67 |
| 22 | 10 | 10 | 0.70 | 0.53 | 3.25 | 2.90 |
| 30 | 14 | 10 | 1.34 | 0.99 | 7.23 | 6.70 |
| 60 | 29 | 30 | 20.35 | 11.58 | 145.80 | 153.02 |

Table : Test problem: mass-spring system with box constraints. Run times [ms] for 10 IP iterations. Test processor: Intel Core i7 3520M @ 3.6 GHz.

Test machine # 2

- ▶ ARM Cortex A9
- ▶ 4 cores
- ▶ NEON SIMD
- ▶ $\approx 120\$$ (entire board)
- ▶ < 1 W per core



Comparison with state-of-the-art solver for linear MPC

| n_x | n_u | N | HPMPC | | FORCES | |
|-------|-------|-----|--------|--------|---------|---------|
| | | | double | single | double | single |
| 4 | 1 | 10 | 0.53 | 0.47 | 1.14 | 0.93 |
| 8 | 3 | 10 | 1.57 | 1.06 | 4.23 | 3.40 |
| 12 | 5 | 30 | 10.60 | 6.92 | 28.29 | 22.56 |
| 22 | 10 | 10 | 11.73 | 6.42 | 39.96 | 33.54 |
| 30 | 14 | 10 | 23.86 | 11.75 | 121.71 | 70.88 |
| 60 | 29 | 30 | 569.34 | 220.07 | 1876.28 | 1373.46 |

Table : Test problem: mass-spring system with box constraints. Run times [ms] for 10 IP iterations. Test processor: ARM Cortex A9 @ 1.0 GHz.

Test machine # 3

- ▶ ARM Cortex A7
- ▶ 2 cores
- ▶ NEON SIMD
- ▶ $\approx 50\$$ (entire board)
- ▶ $< 0.5\text{ W}$ per core



Comparison with state-of-the-art solver for linear MPC

| n_x | n_u | N | HPMPC | | FORCES | |
|-------|-------|-----|--------|--------|--------|--------|
| | | | double | single | double | single |
| 4 | 1 | 10 | 0.74 | 0.59 | * | * |
| 8 | 3 | 10 | 2.13 | 1.38 | * | * |
| 12 | 5 | 30 | 18.62 | 9.32 | * | * |
| 22 | 10 | 10 | 21.13 | 9.29 | * | * |
| 30 | 14 | 10 | 48.36 | 18.95 | * | * |
| 60 | 29 | 30 | 906.54 | 304.47 | * | * |

Table : Test problem: mass-spring system with box constraints. Run times [ms] for 10 IP iterations. Test processor: ARM Cortex A7 @ 1.0 GHz.

- ▶ Application of HPC techniques to MPC
- ▶ HPMPC: state-of-the-art solver for embedded MPC
 - ▶ on the algorithmic side: structure-exploiting solver
 - ▶ on the implementation side: hardware-exploiting code
- ▶ MPC and MHE problems, box and soft constraints
- ▶ Target architectures: x86, x86_64, ARM, PowerPC