On the Use of Parallel Computing for Exact Complexity Certification of Quadratic Programming Algorithms

Joel Wikner
Division of Automatic Control
Linköping University
Linköping, Sweden
joel.wikner@liu.se

Daniel Arnström
Division of Systems and Control
Uppsala University
Uppsala, Sweden
daniel.arnstrom@it.uu.se

Daniel Axehill
Division of Automatic Control
Linköping University
Linköping, Sweden
daniel.axehill@liu.se

***Index Terms*—MPC, mpQP, High Performance Computing, Exact Complexity Certification**

I. Introduction

For implicit linear Model Predictive Control (MPC), the main computational burden is to in real-time solve optimization problems formulated as quadratic programs (QPs), to produce control actions governing a system. These QPs can be expressed as a multi-parametric quadratic program (mpQP) (see e.g.[1]) given by

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^T H x + \left(f^T + \theta^T f_\theta^T\right) x \\ \text{subject to} \quad & Ax \leq b + W\theta \end{aligned} \tag{1}$$

where the decision variable $x \in \mathbb{R}^n$ is related to the control action. Moreover, the parameter $\theta \in \Theta_0 \subseteq \mathbb{R}^p$ is related to the state of the plant, the feasible set of the problem is defined by $A \in \mathbb{R}^{m\times n},\ b \in \mathbb{R}^m$, and $W \in \mathbb{R}^{m\times p}$, and the objective function is defined by $f \in \mathbb{R}^n,\ f_\theta \in \mathbb{R}^{n\times p}$, and $H \in \mathbb{S}_+^n$.

When using implicit MPC in real time to control discrete-time systems that are safety-critical, it is crucial to know the worst-case execution time of QP solvers, as the availability of scheduled stabilizing control actions might be required to maintain stability and reach a certain system performance.

Algorithms that solve QPs with such guarantees have recently been developed in [2] and [3]. Here, the main idea is to perform an offline analysis and partition the parameter space into regions that share the same sequence of solver states q, i.e., the sequence of operations the optimization algorithm used to reach the optimum $x^\star(\theta)$ as a function of θ. In addition, the number of iterations $k(\theta)$ the optimization algorithm will perform will also be given as a function of θ. [3]

Importantly, since the complexity certification investigates a given parameter space, it is also possible for a user to partition the parameter space *before* certification, and then certify the resulting regions in the partition separately. This enables certification of the time-complexity of larger QP problems using high-performance computing (HPC). Specifically, through parallelization, which is the focus of this paper.

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

II. Parallel certification strategies

Parallel computing is commonly used to solve partial differential equations (PDEs) [4], and allows otherwise large computations to be divided into smaller tasks. Such tasks can be shared on multiple cores of a CPU, or similarly on multiple CPUs, that run simultaneously to speed up execution [4]. If these CPUs do *not* share memory, as is common in distributed computing, then communication between CPUs will have to be handled, e.g., by a message-passing architecture MPA [4]. Distributed computing is often used for scalable HPC, commonly called supercomputers [5].

Applying distributed computing to the certification problem, the natural question that follows is how to perform the initial partition of the parameter space. Here, multiple options can be mentioned. Assuming the parameter space is given by a p-dimensional hyperrectangle $R = [l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_d, u_d] \subset \mathbb{R}^p$, where each pair $[l_j, u_j]$ is the interval along dimension j with lower and upper bound $l_j, u_j \in \mathbb{R}$ respectively, one can

1) divide the parameter space into $K \in \mathbb{N}$ equally sized sub(hyper)rectangles $R_1, R_2, .., R_K$ where $R = \bigcup_{i=1}^K R_i$. Here, a natural choice of K is (if possible) the number of CPU cores available in the computation.
2) try to find an estimate of the true parameter-space partition to be computed by the complexity certification and partition the parameter space according to this.
3) use the true parameter partition, which can be useful as a ground truth.

Thus, certification of the original optimization problem, i.e. over the original parameter space Θ_0, can be achieved by distributing regions in the partitioned parameter space across nodes—a user-defined collection of CPU cores—which certify in parallel, and finally collecting the resulting iterations $k(\theta)$.

Interestingly, all of these approaches suffer from a common issue, even for the true "exact" parameter partition men-

tioned. Namely, a load-balancing problem, as certifying certain regions of the parameter space inherently demands longer computational time due to increased complexity within those regions. This inefficiency can sometimes be severe for the computation as a whole, as individual computational resources might remain idle even though particularly complex regions are yet to be certified by other active resources.

III. Load balancing

Two approaches with the potential to mitigate load-balancing issues are presented in this section. One, which can be viewed as a feedback approach, and a second, which can be viewed as a feedforward approach.

A. Feedback using communication between nodes

In the case of an unbalanced initial parameter space partition, the complexity certification algorithm mentioned in Section I can be terminated early given that a user-defined condition is true. An example of such a condition is if the amount of certified regions N^{max} exceeds a certain threshold (see [6] Section V.A for details). If it holds true, the certification process terminates giving the stacks F and S containing tuples corresponding to regions in Θ_0 that have been certified and are yet to be certified, respectively.

If nodes available to the user are found to be idle during certification on a specific node, here called the *active node*, part of the stack S for this *active node* can be allocated to idle nodes via a *feedback law*.

For example, a basic law could be to split the tuple S into two tuples S_1 and S_2, where S_1 is retained at the *active node* and S_2 is given to the idling nodes for further certification. (The split, in the case of an uneven length of the tuple S, is carried out so that the number of elements in S_1 is greater than S_2.) The stack F retained by each node is concatenated when the tasks of all nodes have terminated, giving the final partition of the parameter space and the number of iterations $k(\theta)$.

We note here that an MPA, as described in Section II, is needed since distributed computations are of interest, and nodes require knowledge of other idle nodes. In addition, a means of receiving and sending unfinished regions S is also needed. This communication is undesired, as it in general adds computation time.

B. Feedforward by estimating a balanced partition of the parameter space

An observation that can be made is that *if* the initial partition mentioned in Section II is well balanced with respect to the computation time for each node, then the need for feedback and communication would be reduced, since the problem of idle nodes would be less of an issue. Much akin to how feed-forward is used in control.

As such, attempting to a priori estimate the true parameter partition given by the complexity certification and using it as an initial partition seems to be an attractive alternative to the feedback approach. It is also of interest to accurately estimate a complexity measure related to the effort of certification, i.e., information about how computationally challenging a given region from the initial partition would be to certify.

An example of a complexity measure could be an estimate of the number of iterations $\hat{k}(\theta)$ of each region in the initial partition. Another alternative is to estimate a complexity density

$$\hat{d}(\theta) = \frac{\hat{N}^{reg}(\theta)\hat{k}(\theta)}{V} \tag{2}$$

where, for a region in the initial partition, $\hat{N}^{reg}$ is an estimate of the number of "true" regions located therein and V is the given p-dimensional volume.

For example, using Alternative 1 in Section II with K nodes, a partition of the parameter space would then be balanced with respect to the iterations if the number of iterations $\hat{k}(\theta)$ in each region is roughly equal.

As mentioned in Section II, parallel computing is commonly used to solve PDEs and therefore it is of interest to investigate methodologies in that field. One such example is [7], where the load balancing problem is addressed using adaptive finite element methods.

IV. Estimating the complexity measure

A direct method to obtain $\hat{k}(\theta)$ is to evaluate, for each region in the initial partition, the number of iterations executed by Algorithm 4 in [8] when solving the mpQP in (1) for a given θ within each region. A key advantage of this approach is that this procedure can be performed in parallel.

Similarly, a lower bound for N^{reg} can be obtained from $\hat{k}(\theta)$ as the unique number of iterations observed in each region in the initial partition. Note that this requires multiple samples per region in the initial partition.

Future work

Future work will involve a numerical analysis and a comparative evaluation of the methods presented in Section II-IV.

References

[1] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[2] D. Arnström, D. Broman, and D. Axehill, "Exact worst-case execution-time analysis for implicit model predictive control," *IEEE Transactions on Automatic Control*, vol. 69, no. 10, pp. 7190–7196, 2024.

[3] D. Arnström and D. Axehill, "A unifying complexity certification framework for active-set methods for convex quadratic programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2758–2770, 2022.

[4] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

[5] T. Sterling, M. Brodowicz, and M. Anderson, *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.

[6] S. Shoja, D. Arnström, and D. Axehill, "Parallel domain-decomposition algorithms for complexity certification of branch-and-bound algorithms for mixed-integer linear and quadratic programming," *arXiv preprint arXiv:2503.16411*, 2025, submitted to journal.

[7] R. E. Bank and M. Holst, "A new paradigm for parallel adaptive meshing algorithms," *SIAM Journal on Scientific Computing*, vol. 22, no. 4, pp. 1411–1443, 2000.

[8] D. Arnström, *Real-time Certified MPC Reliable Active-Set QP Solvers*. Linköping University, 2023.