2013 European Control Conference (ECC)
July 17-19, 2013, Zürich, Switzerland.

Multi-core parallelisation of integer optimisation model predictive control for power electronic applications

Helfried Peyrl*, Stefan Richter and Alessandro Zanarini

Abstract— **The deployment of integer optimisation based model predictive control (MPC) for the control of power electronic applications has been limited by the computational burden of the scenario tree exploration and the required fast cycle times. The recent technology trend towards multi-core control platforms offers new possibilities for enabling MPC for power converters. We propose a static scheduling method to efficiently parallelise the tree exploration for converter control using the example of Generalised Model Predictive Direct Torque Control (GMPDTC). We evaluate several scheduling strategies (both suboptimal and optimal) for distributing the work packages over the individual cores with respect to their effectiveness using an eight core platform from Freescale. Moreover, the proposed approach has the advantage that the code has a small memory footprint and every improvement of the sequential code will directly result in an improvement of the parallel version.**

I. INTRODUCTION

Model Predictive Control (MPC) represents an exciting academic research field and at the same time a well established and mature control technology in many industrial applications. Until recently, its appeal has been mainly restricted to processes with rather slow dynamics and with sampling times ranging from a few minutes to many hours, such as the ones encountered in the areas of (petro)chemicals, minerals and metals. The main reason for this restriction can be traced to the computational demand that optimisation-based algorithms such as MPC can pose to the control hardware platforms.

The ever increasing computational capacity that is becoming available in the commonly employed controllers has encouraged the emergence of MPC applications in the automotive, and more recently the power electronics industry, where the time scales are in the milli- or even the micro-second area. A number of publications have reported on its possible application to the control of industrial electronic systems, such as dc-dc converters [1], dc-ac inverters [2], [3], and induction motor drives [4]–[7].

Despite these encouraging advances in enabling MPC for fast cycle times, CPU-based implementations have been limited to simple prediction models and short prediction horizons sometimes comprising not more than one step as in the application reported in [8]. Consequently, to achieve faster cycle times and/or longer prediction horizons, researchers have proposed realisations of MPC problem solvers as application specific circuits through Field Programmable Gate Arrays (FPGAs). Examples of these efforts are [9]–[13].

The authors are affiliated with ABB Corporate Research, Segelhofstrasse 1K, 5405 Baden-Dättwil, Switzerland

* Corresponding author. Email: helfried.peyrl@ch.abb.com

Although the performance of the aforementioned FPGA solutions is indeed impressive, they are accompanied by some shortcomings from a practical viewpoint: firstly, the high engineering effort required for development and maintenance is rather high in comparison to traditional software-based implementations. Secondly, the rigidness of an FPGA design makes structural changes at a later stage time consuming and demanding.

The recent trend to multi-core CPUs observed in both conventional desktop PCs and supercomputers did not walk past the embedded world opening new possibilities for advanced control applications. In this paper we propose an effective parallelisation for multi-core platforms of integer optimisation MPC for power electronic applications where the sampling times are in the µs range.

The control problem is treated as a discrete-time control problem, where the complete converter switch positions are determined by one central control algorithm. More specifically, at each sampling time, all possible converter switch positions are considered, and predictions of the system's behaviour are made over a finite control horizon of a few steps, using a discrete-time nonlinear model of the system. The possible time sequences of converter switch positions are then evaluated by means of a cost function. Out of the converter switching sequence that minimises the cost function, the first element is applied to the converter, and in the next sampling instant the procedure is repeated in accordance with the receding horizon policy.

The paper is organised as follows. Section II reviews *Model Predictive Direct Torque Control* as an example for integer optimisation based MPC for power electronic applications. The parallelisation of the algorithm for multi-core CPUs and implementation details are described in Section III. The experimental results obtained with our approach using an eight core test board from Freescale are provided in Section IV. Conclusions and outlook for future work are summarised in Section V.

II. PROBLEM DESCRIPTION

In this section we will review integer optimisation based MPC for power electronic applications using the example of *(Generalised) Model Predictive Direct Torque Control* (GMPDTC). For a detailed description of the method, we refer the reader to [14] and the references therein.

GMPDTC is a specialised MPC-based algorithm for the torque control of induction machines even though its principle can be applied to other kinds of power electronic

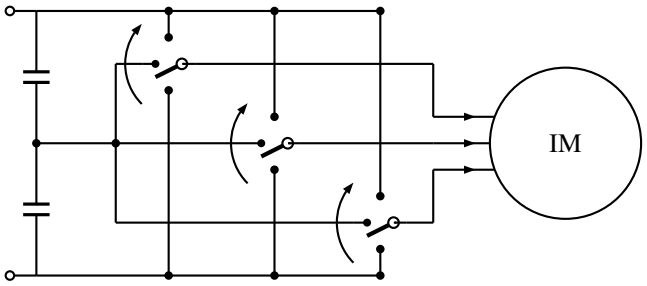

Fig. 1. Equivalent representation of a three-level voltage source inverter driving an induction machine

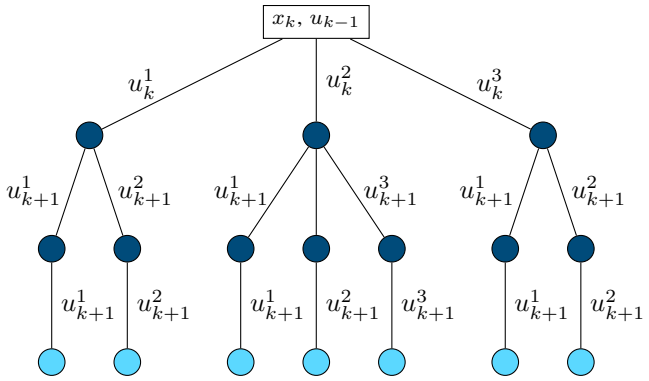

Fig. 2. Example scenario tree for control strategy (S, S, E). Switching nodes are dark blue while extension nodes are shown in light blue.

applications as well. Fig. 1 shows the topology of a three-level neutral point clamped (NPC) voltage source inverter driving an induction motor which represents the example application used throughout this paper.

The control aim is to keep machine torque and flux and the inverter's neutral point potential within specified bands while at the same time minimising the inverter switching frequency or power losses. Instead of using modulation-based techniques such as pulse-width modulation (PWM), the method directly manipulates the switches of the inverter and thus requires the solution of an integer optimisation problem in which the decision variables represent the inverter switch positions.

GMPDTC exploits the fact that the drive outputs (torque, stator flux and neutral point potential) do not need to be kept as close as possible to their references but just within a certain band around their reference values. In every sampling instant the method verifies by means of the prediction model (see below) whether the previous control input can be applied again without driving one (or more) of the outputs outside their bands. If applying the previous control input would result in a band violation, GMPDTC determines the next control move by solving an MPC problem that uses a discrete-time model of the drive to predict the evolution of the states depending on the converter switches:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= g(x_k)\,, \end{aligned} \tag{1}$$

where the state vector $x_k \in \mathbb{R}^5$ comprises the stator and rotor fluxes in the stationary $\alpha\beta$ reference frame and the neutral point potential. The outputs y_k are a nonlinear function of the states and correspond to the electromagnetic torque, the magnitude of the stator flux and the neutral point potential. The control actions $u_k \in \{-1, 0, 1\}^3$ are discrete-valued and describe the different switch configurations of the inverter shown in Fig. 1. For a detailed description of the model we refer the reader to [15].

As a general idea, GMPDTC applies a problem-tailored *move-blocking* strategy to achieve long prediction horizons at a reduced number of switching scenarios: by freezing the control inputs until one of the outputs leaves its band, the degree of freedom in the problem can be reduced significantly yet offering a relatively large prediction horizon. This policy is an efficient heuristic for the application at hand mostly because of two reasons: firstly, since every switching transition is associated with losses in the inverter, a good control algorithm switches as little as necessary and hence intrinsically applies move-blocking. And secondly, move-blocking allows to increase the prediction horizon as in a standard MPC formulation the number of switching scenarios would grow exponentially with the prediction horizon N and therefore would allow only very short prediction horizons.

In GMPDTC, the prediction horizon is partitioned into periods in which switching is allowed and periods in which the switches are frozen until one of the predicted outputs violates its band. In every switching stage, all allowed switch transitions of the inverter are considered and the prediction scenario tree branches accordingly. The sequence of *switching steps* and move-blocking steps (also referred to as *extension steps*) is a parameter of the algorithm and referred to as *control strategy*. It is used instead of a fixed prediction horizon—the length of the prediction horizon varies due to the variable length of the extension steps.

The control strategy is defined by a sequence of the characters S and E corresponding to switching and extension actions. Fig. 2 shows a hypothetical scenario tree for the control strategy encoded by the sequence (S, S, E). In every switching node (S) the plant model (1) is used to compute predictions of states and outputs for the next time step while in an extension node (E), the evolution of the outputs is extrapolated (typically linearly) for as many time steps as all outputs remain within their bands under fixed switch positions. Every leave node of the tree is associated with a certain cost depending on the number of switch transitions and the band violations occurred during the scenario path. The algorithm chooses then the control law that is associated with the leave node that has minimum costs and applies only the first control move to the inverter in accordance with the moving horizon policy of MPC.

III. Parallelisation of integer-based MPC

From a computer science point of view, integer optimisation MPC is a classical *tree traversal* problem in which in the worst-case each node of the tree must be visited exactly once to identify the scenario with minimum costs.

A. Parallel scenario tree traversal

Already sequential implementations can have execution times that depend significantly on data layout and code organisation. For instance, the order in which the nodes are visited (e.g. depth-first or bread-first) has an important impact on the memory footprint of the implementation and its ability to benefit from fast caches.

In general, an efficient parallelisation strategy depends strongly on the time needed to compute a single node, the time spent on inter-core communication, the employed data structures and architecture specific characteristics such as cache sizes.

Integer optimisation problems are typically parallelised based on either *centralised* or *distributed node pool management* [16]. Nonetheless these strategies require some substantial intercommunication for distributing nodes or balancing the workload.

For MPC algorithms such as GMPDTC, however, processing a tree node involves only a modest number of mathematical operations [15]. Moreover, synchronisation between the threads is performed via shared memory which has typically a latency of several dozens of cycles and thus is expensive in comparison with the time needed to compute a node.

In the light of this, we propose a static scheduling method to efficiently parallelise the tree exploration. More specifically, the global tree is split into sub-trees (or paths) which are then assigned to the cores for parallel computation: At the begin of each cycle, the sub-tree exploration tasks are distributed to the cores. Once the worker threads have finished processing, their locally optimal solutions are compared to identify the global optimal solution. This approach does not only exhibit a minimum communication overhead but also has the advantage that every improvement of the sequential code will directly result in an improvement of the parallel version. The number of sub-trees into which the global scenario tree is split is a design parameter and should be chosen depending on the number of available CPU cores to ensure that all cores can be kept busy.

The three-level NPC inverter shown in Fig. 1 admits 27 different switch configurations. Due to the fact that ABB's ACS 6000 has only one snubber circuit in the upper and lower half, not all possible switch transitions are allowed (see Fig. 3). Consequently, there exist 27 different scenario trees, each of them determined by the previously applied inverter switch configuration u_{k-1}.

Mathematically, the feasible switch transitions are characterised by the following inequalities

$$|u_{i,k+1} - u_{i,k}| \leq 1, \quad \forall i \in \{a, b, c\} \tag{2a}$$

$$|u_{a,k+1} - u_{a,k} + u_{b,k+1} - u_{b,k} + u_{c,k+1} - u_{c,k}| \leq 1 \tag{2b}$$

$$||u_{k+1} - u_k||_1 \leq 2. \tag{2c}$$

In a more generic three-phase NPC inverter constraints (2b)-(2c) are not present. We will refer to the different constraints of the ACS 6000 and a standard inverter as *ABB switching restrictions* and *standard switching restrictions*.

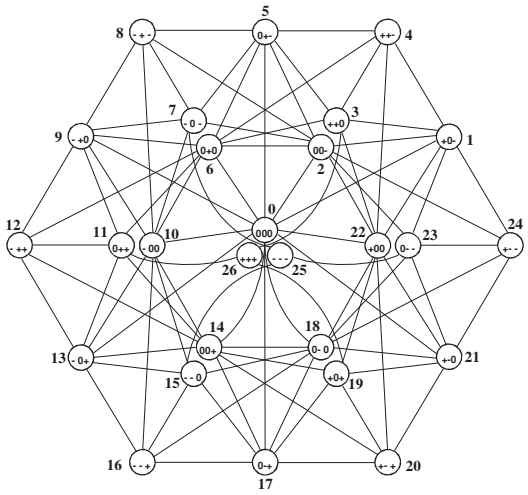

Fig. 3. Possible voltage vectors and allowed switch transitions of the ABB ACS 6000 drive (Courtesy of ABB ATDD Switzerland)

TABLE I

SCENARIO TREE PROPERTIES UNDER ABB SWITCHING RESTRICTIONS.

No. trees in group	1	6	6	6	6	2
No. first level nodes	13	10	8	7	6	4
No. scenarios for (S, E)	13	10	8	7	6	4
No. scenarios for (S, S, E)	121	85	69	51	49	25

The trees can be grouped with respect to the number of scenarios they comprise. Tables I and II summarise the properties of the groups resulting from ABB and standard switching constraints, respectively.

Independent of the control strategy (and the type of restrictions applied), there always exists one tree comprising more scenarios—and hence computations—than all the others (the tree associated with the zero voltage vector characterised by $u_{k-1} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$). Processing this tree will determine the worst case execution time of the algorithm.

We propose to split the scenario trees at the first level, i.e. the children of the root node form the roots of the resulting sub-trees. Then the number of sub-trees to be explored within a cycle will depend only on the current switch positions and the imposed hardware switching constraints. For example, with ABB switching constraints the root node has at most 13 children (see Table I). Since our target platform offers up to eight cores, there are always more sub-trees than CPU cores if the biggest scenario tree must be explored.

TABLE II

SCENARIO TREE PROPERTIES UNDER STANDARD SWITCHING RESTRICTIONS.

No. trees in group	1	6	12	8
No. first level nodes	27	18	12	8
No. scenarios for (S, E)	27	18	12	8
No. scenarios for (S, S, E)	343	245	175	125

B. Scheduling strategies for tree exploration

Due to the hardware switching constraints imposed by the inverter hardware, the sub-trees have different size. Therefore, the time it takes to explore a specific sub-tree will be different as well.

Since the structure of the global scenario tree resulting from the previously applied control action u_{k-1} and the switching restrictions is known, the processing time p_i of a sub-tree i can be estimated for a worst-case scenario. This allows to apply standard techniques from scheduling theory to efficiently parallelise GMPDTC.

The problem of optimally distributing the sub-problems to the available cores can be posed as a *makespan minimisation problem*, a well-studied problem in scheduling theory: given m machines and n jobs with known processing times $p_1, \ldots, p_n$, the makespan minimisation problem is to find a scheduling strategy that finishes the batch of jobs in minimum time.

Under the assumption that processing of a job must not be interrupted and each job can be processed on any machine, the minimum makespan is known to be NP-hard already for $m = 2$ machines [17]. An optimal schedule is given by the solution of the following integer linear problem (ILP):

$$
\begin{aligned}
C_{max}(OPT) := \min\ & C_{max} && (3)\\
\text{subject to } & \sum_{j=1}^{n} x_{ij} p_j \leq C_{max}, && i = 1, \ldots, m\\
& \sum_{i=1}^{m} x_{ij} = 1, && j = 1, \ldots, n\\
& x_{ij} = \{0, 1\},
\end{aligned}
$$

where the binary decision variable x_{ij} is equal to 1 if job j is assigned to machine i and zero otherwise.

A well-known heuristic achieving fairly good approximate solutions for $C_{max}(OPT)$ is the *Longest Processing Time* (LPT) first rule. The jobs are sorted by decreasing processing time and then assigned greedily one by one to the least loaded machine.

Theorem 1 (See, e.g. [17]): The makespan $C_{max}(LPT)$ obtained by applying the LPT-rule satisfies the tight bound

$$\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}.$$

Furthermore, if an optimal schedule results in at most two jobs on any machine, the LPT-rule is optimal.

To achieve the fast execution times required by the control application, the scheduling strategies—no matter how they were derived—can be stored as look-up tables on the control platform. The scheduling strategy selected in a particular cycle is then depending only on the previously applied control action.

We estimated the times needed for actions E and S and derived scheduling strategies using the following three different methods:

Simple — The 'Simple' strategy aims to equally distribute the number of sub-trees to the cores.

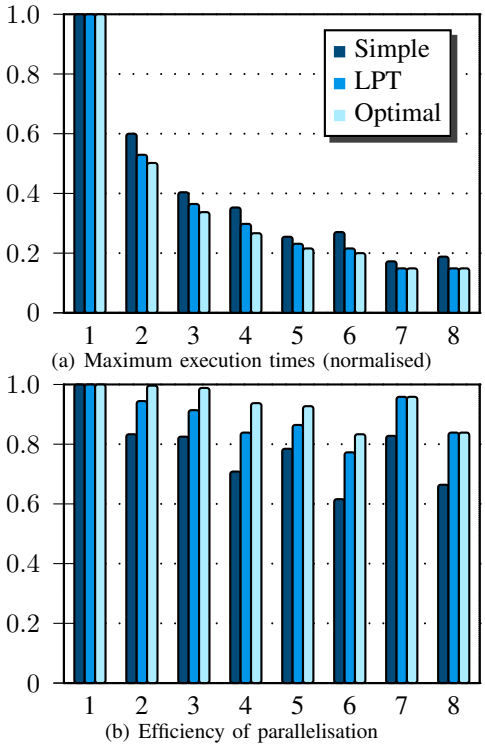

Fig. 4. Results for strategy (S, S, E) under ABB switching restrictions (at most 121 scenarios) using the schedules 'Simple', 'LPT' and 'Optimal'.

LPT — The longest processing time first rule as described above.

Optimal — The strategy obtained by solving ILP (3).

The theoretical parallelisation performance achievable with these strategies is illustrated in Fig. 4 using the example of the control strategy (S, S, E) under ABB switching restrictions. In this example, the performance of the 'LPT' schedule is close to optimal whereas the 'Simple' schedule performs significantly worse. Note that in the case of the 'Simple' schedule, the execution times obtained with six cores are worse than with only five cores. This phenomenon which can also occur with heuristics such as 'LPT' is well-known in the literature and usually referred to as *speedup anomaly* [16]. Furthermore, the best results are achieved already with seven cores. The reason for this is that the number of sub-trees, i.e. jobs, is too low and hence the partitioning of the work too coarse-grained to benefit from the eighth core.

The analysis of other switching restrictions / control strategy combinations yielded similar results although we were not able to compute the optimal schedules in all cases because of the computational complexity of problem (3).

Remark 1: Note that the schedules 'LPT' and 'Optimal' were derived using idealised assumptions such as negligible communication times between the master and the worker threads. Consequently, the estimated efficiencies are an upper bound of what is achievable in a real system. In principle it is possible to improve the quality of the schedules if the communication time between the threads is taken into

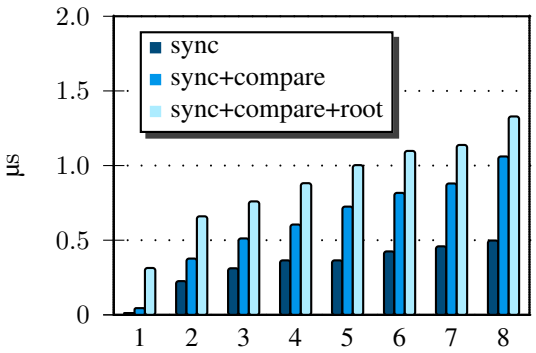

Fig. 5. Time spent on synchronisation (busy wait), comparison of local solutions and processing of the root node.

account accordingly.

IV. EXPERIMENTAL RESULTS

We used Freescale's eight core QorIQ P4080 platform (1.2 GHz) for the experimental evaluation of our algorithms. The results reported in this paper are obtained using 32 bit floating point arithmetic; experiments with other data types exhibited similar parallelisation efficiencies.

In a first experiment we assessed the timing of different synchronisation primitives (mutexes, semaphores, spinlocks, software interrupts, atomic integers and ad-hoc busy wait) some of which implied kernel-space system calls and some other only user-space calls. The most effective synchronisation is achievable by the ad-hoc busy wait that, in practice, is a tailored code that resides completely in user-space. We report the results for the ad-hoc busy wait in Fig. 5. The leftmost bars show just the time required by the busy wait synchronisation, i.e. the threads do not perform any work. The bars in the middle contain in addition the time needed by the master thread to compare the solutions reported by the worker threads. The rightmost bars additionally comprise the time spent in processing the root node of the tree which is processed by the master thread.

In the following we report the experimental results obtained for the strategy (S, S, E) subject to ABB or standard switching restrictions using the scheduling strategies 'Simple', 'LPT' and 'Optimal' described in Section III-B.

Remark 2: We ordered the derived schedules such that the job with the longest processing time is assigned to the master thread. Synchronisation between threads is performed via shared memory; the worker threads are notified with a delay caused by the latency of writing to memory the *dispatch* command and the latency of fetching it from memory. This memory round trip delay keeps the memory controller busy yet the master thread can actively start carrying out its job. Similarly when the worker threads are finished, there is a latency of writing to memory the *work-done* command. During the write-to-memory, the master thread is unable to see that the worker threads completed their jobs. Therefore, this back notification delay can also be exploited by the master thread for carrying out additional work. Ultimately, giving the master thread a slightly longer job is beneficial to further improve the runtimes.

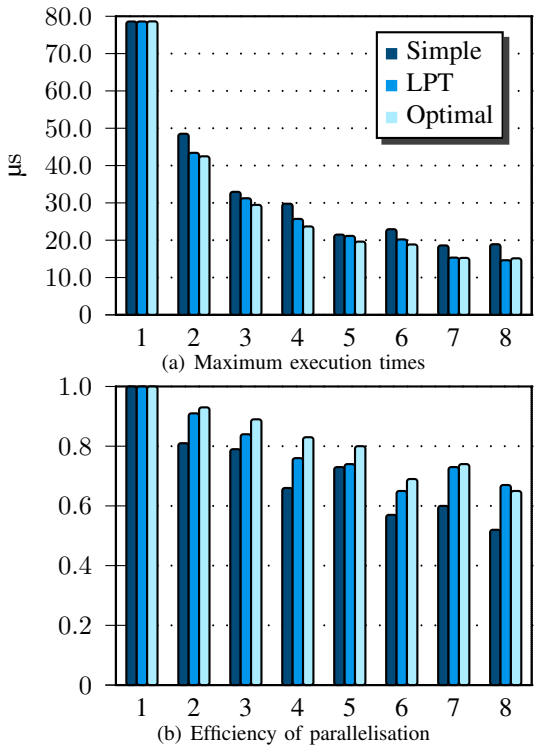

Fig. 6. Results for strategy (S, S, E) under ABB switching restrictions (at most 121 scenarios, P4080, float).

In Fig. 6 the runtimes and efficiencies of the strategy (S, S, E) under ABB restrictions are shown. In this scenario, the 'Optimal' schedule outperforms both the 'LPT' and the 'Simple' schedules although the difference between 'Optimal' and 'LPT' is small. The target sampling time of 25 µs is achievable already with four cores when the optimal schedule is used. Note that for eight cores, 'Optimal' takes slightly more time than 'LPT'. This is due to the fact that the schedules were derived under idealised assumptions. For example, the time needed for comparing the costs of a local solution of a worker thread with the costs of the incumbent solution and updating it if necessary is not taken into account.

The pattern of the experimental runtimes of the three schedules matches the one obtained under idealised assumptions shown in Fig. 4. This indicates that our standing assumption that the traversals of the sub-trees can be regarded as jobs of a makespan minimisation problem is reasonable. Nevertheless, the achievable efficiencies of the parallelisation in the experiments are worse than the ones obtained in the theoretical assessment. This is a consequence of the idealised assumptions under which the schedule policies were derived. In the following we list the main factors which were neglected:

- Synchronisation times between master and worker threads
- Processing time of the root node
- Comparison and updating best incumbent solutions during tree traversal

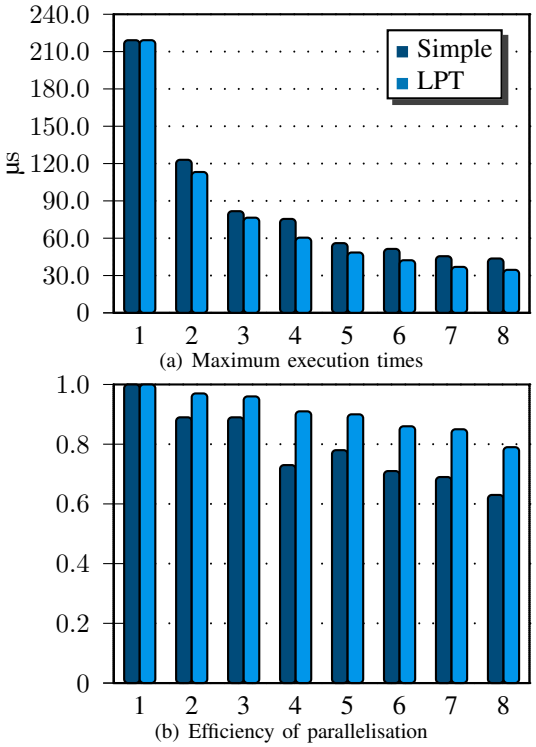

Fig. 7. Results for strategy (S, S, E) under ABB switching restrictions (at most 343 scenarios, P4080, float).

- Code is not completely branch free and therefore worst-case performance difficult to assess.

Fig. 7 shows the results obtained for an inverter with industry standard switching restrictions. Because of the computational complexity we were not able to derive the optimal schedules in this example. In general, the efficiencies of the parallel code is higher because the larger number of subtrees allows a more fine-grained distribution of the work over the threads. For strategy (S, S, E), a target sampling time of 50 µs is achievable already with five cores if the 'LPT' schedule is used. When integer arithmetic is used, 25 µs are achievable as well on the P4080.

V. CONCLUSIONS AND OUTLOOK

In this paper we proposed a static scheduling method to efficiently parallelise the tree exploration of integer optimisation based MPC problems for power electronic applications. We verified our method experimentally on an eight core platform which together with the developed algorithms proved to be powerful enough to solve integer MPC problems of practical problem size and interest. Moreover, the platform is powerful enough to execute all computations in floating-point arithmetic, therefore possibly simplifying the development process.

In future work we plan to investigate how the control algorithm could benefit from additional information about the scenario tree arising at runtime. A very scarce online node redistribution could possibly further improve the performance especially if combined with the recently proposed branch-and-bound approach for GMPDTC in [18].

To sum up, multi-core MPC constitutes already by today a viable alternative to development- and maintenance-intensive FPGA-based solutions.

ACKNOWLEDGEMENT

The authors would like to express their sincere thanks to Tobias Geyer, ABB Corporate Research, Switzerland for his support with the MPDTC code and the insightful discussions.

REFERENCES

[1] T. Geyer, G. Papafotiou, R. Frasca, and M. Morari, "Constrained Optimal Control of the Step-Down DC-DC Converter," *IEEE*, vol. 23, no. 5, pp. 2454–2464, Sept. 2008.

[2] P. Cortes, J. Rodriguez, D. Quevedo, and C. Silva, "Predictive Current Control Strategy With Imposed Load Current Spectrum," *IEEE Transactions on Power Electronics*, vol. 23, no. 2, pp. 612–618, 2008.

[3] J. Rodriguez, J. Pontt, P. Correa, P. Lezana, and P. Cortes, "Predictive power control of an AC/DC/AC converter," in *Industry Applications Conference, 2005. Fourtieth IAS Annual Meeting. Conference Record of the 2005*, vol. 2, Oct. 2005, pp. 934–939.

[4] J. Holtz and S. Stadtfeld, "A Predictive Controller for the Stator Current Vector of AC Machines Fed from a Switched Voltage Source," in *International Power Electronics Conference IPEC*, Tokyo, 1983, pp. 1665–1675.

[5] A. Linder and R. Kennel, "Model Predictive Control for Electrical Drives," in *Power Electronics Specialists Conference. PESC 2005. IEEE*, 2005, pp. 1793–1799.

[6] T. Geyer, G. Papafotiou, and M. Manfred Morari, "Model Predictive Direct Torque Control—Part I: Concept, Algorithm, and Analysis," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1894–1905, 2009.

[7] H. Peyrl, G. Papafotiou, and M. Morari, "Model Predictive Torque Control of a Switched Reluctance Motor," in *ICIT '09 Proceedings of the 2009 IEEE International Conference on Industrial Technology*, Gibbsland, Australia, Feb. 2009, pp. 1–6.

[8] G. Papafotiou, J. Kley, K. G. Papadopoulos, P. Bohren, and M. Morari, "Model Predictive Direct Torque Control—Part II: Implementation and Experimental Evaluation," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1906–1915, 2009.

[9] T. Johansen, W. Jackson, R. Schreiber, and P. Tondel, "Hardware architecture design for explicit model predictive control," in *American Control Conference, 2006*, 2006, pp. 1924–1929.

[10] L. G. Bleris, P. D. Vouzis, M. G. Arnold, and M. V. Kothare, "A co-processor FPGA platform for the implementation of real-time model predictive control," in *American Control Conference, 2006*, 2006, pp. 1912–1917.

[11] M. S. K. Lau, S. P. Yue, K. V. Ling, and J. M. Maciejowski, "A comparison of interior point and active set methods for FPGA implementation of model predictive control," in *Proc. European Control Conference*. Budapest, Hungary: European Union Control Association, Aug. 2009.

[12] M. Mönnigmann and M. Kastsian, "Fast explicit MPC with multiway trees," in *Proc. 18th IFAC World Congress, Milano, Italy*, Sept. 2011, pp. 1356–1361.

[13] J. L. Jerez, G. A. Constantinides, and E. C. Kerrigan, "An FPGA implementation of a sparse quadratic programming solver for constrained predictive control," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 209–218.

[14] T. Geyer, "Generalized model predictive direct torque control: Long prediction horizons and minimization of switching losses," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, 2010, pp. 6799–6804.

[15] ——, *Low complexity model predictive control in power electronics and power systems*. Cuvillier Verlag, 2005.

[16] E. G. Talbi, *Parallel combinatorial optimization*. Wiley-Blackwell, 2006.

[17] M. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, July 2008.

[18] T. Geyer, "Computationally Efficient Model Predictive Direct Torque Control," *IEEE Transactions on Power Electronics*, vol. 26, no. 10, pp. 2804–2816, Oct. 2011.