Hardware Synthesis of Explicit Model Predictive Controllers

Tor A. Johansen, Warren Jackson, Robert Schreiber, Petter Tøndel

***Abstract*— The general solution to constrained linear and piecewise linear model predictive control (MPC) has recently been explicitly characterized in terms of piecewise linear (PWL) state feedback control. This means that a PWL controller can be precomputed using parametric programming, and the exact explicit MPC implementation amounts to the evaluation of a PWL function in the control unit. It has recently been shown that PWL function evaluation can be accelerated by searching a binary tree data structure, leading to highly efficient, accurate, and verifiable software implementation in low-cost embedded control units. In this work we report hardware synthesis results for this type of PWL control, and show that explicit MPC solutions can be implemented in an application specific integrated circuit (ASIC) with about 20,000 gates, leading to computation times in the microsecond scale. This opens the way for the use of highly advanced control designs such as constrained MPC in small-scale industrial and consumer electronics application areas that are characterized by fast sampling or low cost, including mechatronics, MEMS, automotive control, power electronics, and acoustics. The main limitation of the approach is that the memory requirements increase rapidly with the problem dimensions.**

I. Introduction

Recently, several control design and synthesis methods resulting in piecewise linear (PWL) state feedback control structures have been developed. These include exact explicit PWL solutions to constrained linear model predictive control (MPC) [1], [2], [3], MPC of piecewise linear systems [4], approximate explicit PWL solutions to nonlinear constrained MPC [5], [6], hybrid MPC [7], in addition to optimal constrained control allocation problems [8], [9].

These control design methods result in PWL controller functions $k : \mathbb{R}^n \rightarrow \mathbb{R}^m$ represented as

$$k(x) \quad = \quad \begin{cases} K_1 x + g_1, & \text{if } x \in X_1 \\ K_2 x + g_2, & \text{if } x \in X_2 \\ \quad \vdots & \\ K_N x + g_N, & \text{if } x \in X_N \end{cases} \tag{1}$$

where x is the input to the controller function, n is the dimension of this vector, m is the output dimension of the function k, and $K_i \in \mathbb{R}^{m \times n}$ and $g_i \in \mathbb{R}^m$ are gain matrices and vectors. The polyhedral sets $X_i \subset \mathbb{R}^n$ of the polyhedral partition $\mathcal{P} = \{X_1, ..., X_N\}$ are represented by linear inequalities (half-spaces separated by hyper-planes)

$$X_i \quad = \quad \{x \in \mathbb{R}^n | A_i x \leq b_i\} \tag{2}$$

for $i = 1, ..., N$. Such a partition may be assumed to satisfy $\text{int}X_i \cap \text{int}X_j = \emptyset$ for $i \neq j$ (they intersect only at the boundary), where $\text{int}X_i$ denotes the open interior of the closed set X_i. The PWL controller is completely characterized by the following data: $\{K_i, g_i, A_i, b_i\}_{i=1}^N$.

The controller output will be given by the PWL function $u = k(x)$ and the argument x will typically change at every sampling instant based on measurements, user input, and signals from a higher level control system. Controller implementation thus requires evaluation of a PWL function (1)-(2) at each sampling instant in the control unit.

In some variations of approximate explicit MPC, such as [10], the polyhedral sets X_i are represented by vertices

$$X_i \quad = \quad \text{conv}(v_i^1, v_i^2, ..., v_i^L) \tag{3}$$

where conv() denotes the convex hull. These representations are equivalent, but require some modification of the algorithms used for evaluation. In other variants of approximate explicit MPC, such as [6], [11], the partition has an orthogonal structure (quad-tree or $k-d$-tree [12], [13]) that may reduce computational complexity since the partition consists of hyperrectangles

$$X_i \quad = \quad \{x \in \mathbb{R}^n | \underline{b}_i \leq x \leq \bar{b}_i\} \tag{4}$$

rather than general polyhedra.

A binary search tree representation of arbitrary polyhedral PWL functions (1)-(2) was suggested in [14], [15]. It leads to very low requirements for processing in the control unit, but requires additional memory to store a precomputed binary search tree data structure. In this work we report some results on digital hardware synthesis for PWL function evaluation logic based on such a data structure.

Compared to conventional MPC, which relies on extensive numerical optimization in real time, the benefits of explicit PWL evaluation include simpler verification, low computational complexity, no recursive numerical computations, and deterministic execution. The main limitation of the explicit MPC approach is that the offline computational load (during synthesis) and control unit memory requirements usually increase quickly with the dimension and complexity of the problem, making it useful mainly for small scale problems. Still, this may not be prohibitive in some applications from areas such as automotive [16], [17], [18], biomedical systems, aerospace, power electronics [19], microelectronics/MEMS [20], acoustics, and rotating machinery [5]. Recently, the

Tor A. Johansen and Petter Tøndel are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. Their work was in supported by the Research Council of Norway.

Warren Jackson and Robert Schreiber are with Hewlett-Packard Laboratories, Palo Alto, California 94304-1126, USA

Petter Tøndel is currently with SINTEF Applied Cybernetics, Trondheim, Norway.

Manuscript received .....

problem of implementing explicit MPC efficiently on microprocessors and microcontrollers in practical application[1] has been given much attention [21], [22], [23], [24], [25].

A different approach to a computationally efficient implementation of MPC is to make the core computations of a numerical online solver as fast as possible. Suboptimal and simplified MPC strategies, [26], [27], [28], [29], allow computational complexity to be reduced at the cost of performance loss. A general purpose processor is used in [30] for the efficient implementation of MPC, while in [31] a parallel processing real-time architecture for MPC is investigated, and in [32] an FPGA implementation of an iterative numeric quadratic programming solver is proposed and tested. Dynamically reconfigurable analog/digital hardware capable of handling MPC computation requirements was proposed in [33], dynamic scheduling of real-time MPC was considered in [34], while [35], [36] describes a methods for reducing the precision of a microprocessor to the minimum while maintaining close to optimal control performance. A Logarithmic Number System (LNS) based microprocessor was proposed in [37] for computational cost savings. Compared to explicit MPC such approaches are expected to scale much better with respect to memory use as the dimensions of the problem increases.

The main contribution of the present paper is to consider the option of direct hardware synthesis of *explicit* piecewise linear MPC controllers using a high-level hardware synthesis tool [38]. It may prove to be beneficial or cost-efficient in some applications where an ASIC or FPGA implementation would be preferable to a microprocessor based implementation.

II. Polyhedral partition binary search

A typical polyhedral partition used to define a PWL function solving an MPC problem contains several hundred or thousands of polyhedra, even if complexity reduction methods such as [39], [40] are used. Often, neighboring regions contain the same linear mapping such that the number of linear mapping coefficient matrices to store is significantly less than the polyhedral regions that must be stored as linear inequality coefficient matrices.

Evaluating a PWL function (1) for a given x consists of two steps:

- Identify the polyhedral region index i such that $x \in X_i$, and
- Evaluate the corresponding linear function $K_i x + g_i$.

The second step is completely straightforward, while the first step can be implemented in at least two ways, as described below (see also [41] for a third way). Direct implementation of the first step by sequentially searching through each polyhedral region X_i in order to determine the one that satisfies $x \in X_i$ is a simple, but computationally inefficient strategy. In the worst case, N matrix operations of the form $A_i x - b_i \leq 0$ must be carried out, which is computationally expensive since the number of polyhedral regions N may amount to several thousands (see [2] for worst case complexity results). The use of a binary search tree to organize the search is a more efficient strategy [15], and will be used here.

[1]See also ParOS website http://www.parostech.com/ParOS.html, and J. A. Mandler et al. Parametric model predictive control of air separation. Ip Com prior art database.

A. Binary search tree representation

The PWL evaluation strategy described in [15] is to build a binary search tree data structure that supports efficient search for the polyhedral region X_i that satisfies $x \in X_i$ for any given x. The overall idea is based on the observation that evaluation of a linear expression $c_i^T x - d_i$ corresponding to a single hyperplane cut $c_i^T x - d_i = 0$ may significantly reduce the number of candidate polyhedral regions. This is illustrated in Figure 1, where the partition $\mathcal{P} = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7\}$ contains 7 polyhedra. Consider the point x near the center of the area. In order to determine which polyhedron this point belongs to, the linear expression $c_1^T x - d_1$ may be evaluated. This hyperplane cuts the partition into two parts: $\{X_1, X_2, X_3\}$ (below the hyperplane) and $\{X_4, X_5, X_6, X_7\}$ (above the hyperplane). With the given x, the sign of the evaluated expression $c_i^T x - d_i$ shows that x is located above the hyperplane, and one can infer that $x \in X_4 \cup X_5 \cup X_6 \cup X_7$. In other words, 3 out of 7 polyhedral regions has been excluded, and the remaining problem is greatly reduced. This procedure can be repeated, as shown in the 2nd part of Figure 1. By evaluating the linear expression $c_2^T x - d_2$ one can infer that $x \in X_4 \cup X_5$. In other words, 2 out of 4 remaining regions have been eliminated and the problem is again greatly reduced. Since X_4 and X_5 are separated by a single hyperplane, one can easily detect that $x \in X_5$ by evaluating this third linear expression. In summary, evaluation of three linear expressions is sufficient to determine which polyhedral region x belongs to in this case. We observe that each region is characterized by 3 or 4 linear inequalities, such that an exhaustive search may require the evaluation of more than 20 linear expressions in this case. For general algorithms to construct such a binary search tree, we refer to [15] and remark that the computational benefits are relatively much more significant in larger examples.

The procedure illustrated above is completely general, and corresponds to the construction and traversal of a binary search tree where at each node there is a linear expression corresponding to a hyperplane that cuts the remaining partition into three parts: Polyhedra that are completely on one side of the hyperplane, polyhedra that are completely on the other side of the hyperplane, and polyhedra that are cut by the hyperplane. Estimating that each of these parts are of similar size, each node in the search tree will exclude approximately 1/3 of the polyhedral regions. This leads to a search tree depth D

$$D \quad \approx \quad 1.7 \log_2 N \tag{5}$$

that is estimated to be logarithmic in N the number of polyhedra in the partition, [15].

B. Binary tree search algorithm

The PWL function evaluation problem consists of executing at each sample two sets of nested loops

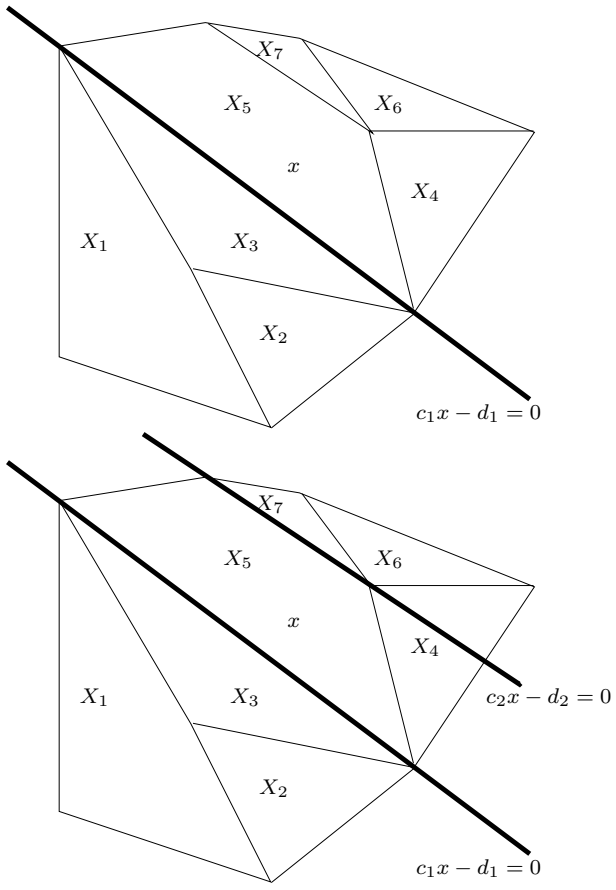

Fig. 1. Hyperplane cuts of polyhedral partition.

- Tree search loop: Starting at the root node of the binary search tree the loop iterates through the nodes until a leaf node is reached. At each loop iteration, an inner loop evaluates the hyperplane cut $c_i^T x - d_i$. A binary tree branch is based on the sign of this expression. The leaf node identifies the linear expression to be evaluated to compute the PWL function value.
- Control evaluation loop: Evaluation of the linear expression $k(x) = K_i x + g_i$ where the index i corresponds to the leaf node. This is a nested loop corresponding to a matrix multiplication operation.

Together the two loops form a control block. The total time for execution of a control block is the sum of the times for the two parts.

C. Numerical round-off errors

Due to round-off errors in numerical computations that leads to the polyhedral representation of the PWL function, see [2], [3], the mathematical partition property $\text{int}X_i \cap \text{int}X_j = \emptyset$ for $i \neq j$ and $X_1 \cup X_2 \cup ... \cup X_N = \overline{X}$ ($\overline{X}$ is the whole region of interest) will hold only approximately due to numerical errors. This means that some regions may slightly overlap, and there may be small gaps between some regions. The binary search procedure will automatically complete the partition since when a leaf node is reached, the corresponding linear function will be evaluated without regard to numerical errors in the representation of the polyhedra. This means that the polyhedra will be extended to cover the small gaps. Likewise, overlapping regions will be uniquely resolved, and the boundary linear functions will be extrapolated outside the original partition if x happens to be located outside the partition.

Numerical round-off errors cannot accumulate in the tree search loop since the only information carried from one iteration to the next is the binary branching decision. The resulting insensitivity to numerical errors means that fixed point arithmetics may be utilized as an alternative to floating-point arithmetics without any complications. Consequently, the accuracy of implementation will degrade gracefully as the number of bits used to represent the numerical data decreases (roundoff errors), in the sense that numerical instability will not occur. The hardware synthesis tests that we performed used 32-bit integer arithmetic to represent the PWL function parameters and 16-bit integers for array indices. Less costly circuits may be achievable if shorter integers can be used, but scaling and accuracy becomes an issue.

n	Number of input variables (parameters), i.e. $\dim(x) = n$
m	Number of output variables, i.e. $\dim(u) = m$
N	Number of polyhedral regions in partition
M	Number of nodes in search tree
D	Depth of search tree (maximum number of nodes that must be traversed to reach a leaf node)
H	Number of hyperplanes

TABLE I

KEY PARAMETERS OF THE PWL FUNCTION EVALUATION PROBLEM.

III. HARDWARE SYNTHESIS

In this section we consider digital hardware synthesis of the binary tree search algorithm in section II-B, and how the complexity of the resulting hardware design scales with the problem parameters. The key parameters characterizing the PWL function complexity are defined in Table I.

A. Hardware synthesis approach

The hardware design program PICO-Express (a product of Synfora Inc.), based on the HP Labs PICO research [38] takes the C source code of the PWL function evaluation algorithm, as described in section II-B, as an input along with some assumptions about the memory bandwidth. This program then computes an efficient hardware architecture including cache size, functional register assignments, and ALUs. It also generates estimates of performance, execution time, and memory requirements. PICO produces the hardware design expressed at the register-transfer level (RTL) in a hardware design language, either VHDL or Verilog. The RTL design is tested and verified by PICO as well. This design can be implemented in an FPGA or an ASIC. In either case, routing and placement tools, as well as libraries of parameterized macrocells (for the RTL level components such as adders and registers) are used to generate the FPGA or ASIC implementation. The main differences between ASIC and FPGA implementations are as follows. ASICs can be faster, can include analogue as well

as digital signals, can demonstrate lower power consumption and are less expensive for large unit volumes. FPGAs are more flexible during design time, less expensive for small lots, and require less lead time.

The tree search loop and control evaluation loop are processed in a pipeline. It consists of a pipe-fill phase, the prolog, where iterations are initiated at regular intervals. The initiation interval is the number of clock cycles between sequential starts of the inner loop. No iterations are complete during the prolog. Next, the loop enters the steady-state where for every initiation interval one iteration is completed and one new iteration is initiated. When the loop nears termination, it enters the pipe-drain state, known as the epilog, in which the pipeline is drained by allowing iterations to complete without new iterations being initiated. PICO synthesizes a hardware design with the requested initiation interval using heuristics to reduce the hardware cost. (If an unachievable initiation interval is requested, PICO reports that the requested throughput is impossible using its library of functional elements.)

Both the tree search loop and the control evaluation loop are doubly nested loops. In the tree search loop, whose synthesis we report on here, the outer loop is a loop over depth in the search tree. The inner loop is a loop of n iterations for the evaluation of the dot product of the input vector x with one hyperplane normal vector c_i. An initiation interval equal to one can be achieved by PICO, since the inner loop recurrence (evaluation of $c_i^T x$) is through an addition, and PICO can generate designs using an one-cycle adder. There is also an important dependence of larger latency. At the end of the inner loop, the dot product $c_i^T x$ is first compared with a constant d_i, then a branch is taken to select one of two possible child nodes in the search tree, then the index of the selected node is used to start the lookup of the parameters of the next hyperplane from memory. To accommodate the latency of these operations and still achieve an initiation interval equal to one, the inner loop is padded with a few "slack" iterations at the front. These do no work, but they increase the number of inner-loop iterations between the completion of one dot product and the start of the next one, so that there are enough cycles to cover this latency. For the case studies, all the tables of hyperplane normals c_i and pointer arrays that define the search tree structure are assumed to be stored in fast SRAM in the accelerator device. PICO can also synthesize designs in which data arrays are kept in main memory.

B. Benchmark problems

The characteristics of the benchmark problems used in our case study are given in Table II. The double integrator and helicopter examples are described in more detail in [14], and the quadruple integrator example in [41]. The MPC horizon h is the number of samples the MPC looks into the future when optimizing the control input in order not to violate the constraints at some future point in time. Please see these references for a description of the control design, performance, and additional details.

Problem	n	m	N	M	D	H
Double integrator $h=1$	2	1	5	13	4	10
Double integrator $h=5$	2	1	25	87	7	86
Double integrator $h=10$	2	1	95	323	9	346
Double integrator $h=15$	2	1	215	815	11	818
Quadruple integrator $h=2$	4	1	3	5	2	14
Quadruple integrator $h=6$	4	1	31	215	8	268
Quadruple integrator $h=10$	4	1	81	715	11	1188
Helicopter $h=1$	6	2	47	527	11	82
Helicopter $h=2$	6	2	152	15395	19	1107

TABLE II

BENCHMARK PROBLEM CHARACTERISTICS, WHERE h IS THE MPC HORIZON.

Problem	Gates (kGates)	Memory (kBytes)	Clock cycles	Loop time (μs)
Double integr. $h=1$	19.3	0.7	33	1.65
Double integr. $h=5$	19.6	31	48	2.40
Double integr. $h=10$	19.7	452	58	2.90
Double integr. $h=15$	19.8	2680	68	3.40
Quadruple integr. $h=2$	19.7	0.4	31	1.55
Quadruple integr. $h=6$	20.2	235	73	3.65
Quadruple integr. $h=10$	20.4	3414	94	4.70
Helicopter $h=1$	21.7	202	129	6.45
Helicopter $h=2$	22.9	62873	201	10.5

TABLE III

HARDWARE SYNTHESIS RESULT SUMMARY WITH A CLOCK FREQUENCY ON 20 MHZ. THE SYMBOL h DENOTES THE MPC HORIZON. THE TIMES WILL BE NEARLY A FACTOR OF 10 FASTER FOR A CLOCK FREQUENCY OF 200 MHZ. DATA ARE THOSE PROVIDED BY PICO EXPRESS 1.3.

C. Number of clock cycles per control blocks

The total number of clock cycles required to execute one control block is summarized in Table III and illustrated in Figure 2. The number of clock cycles does not explicitly depend on the number of parameters but rather on the search depth of the tree. The number of clock cycles, NClock, is approximately given by

$$\text{NClock} \quad \approx \quad D(n+1+\text{SLACK}) + m(n+1+\text{SLACK}) \tag{6}$$

where SLACK is the number of padding iterations needed, as discussed above. It is 2 at 10 - 20 MHz clock speeds and 4 for 200 MHz clock speeds. This variable represents the time it takes to set up the inner loop and the number of clock cycles for memory access. The first term is the number of clock cycles for the tree search; the second is the number of clock cycles needed for evaluation of $k(x)$ once the appropriate polyhedron has been determined. Because the search depth increases as $\log_2(M)$, the execution time also increases as $\log_2(M)$. This scaling can continue until the number of nodes is so large that the data can no longer be held in scratch SRAM memory and must be added as general system memory. The limits depend on the technology of the chip but roughly 20MB of SRAM is put on current Intel chips so this represents a break point. In the future this number will go up. Beyond this limit, the access time jumps to roughly 50 nsec. For clock speeds of 10 and 20 MHz, this is not a problem but for 200 MHz clock speeds, this transition can lead to slower performance requiring a larger value for SLACK. In

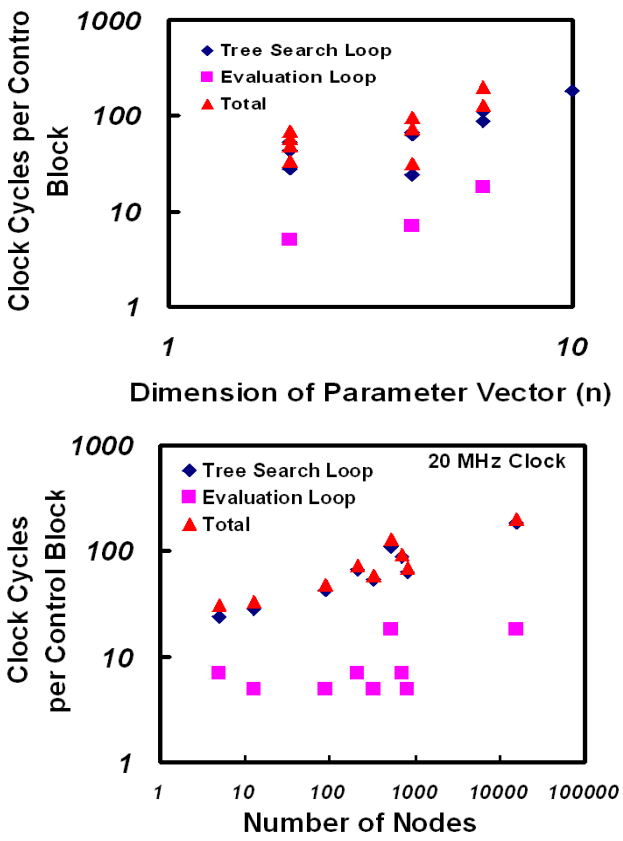

Fig. 2. The number of clock cycles required to evaluate the PWL controller, as a function of the dimension $n = 2, 4, 6$ of the parameters vector x (upper) and as a function of the number of search tree nodes (lower).

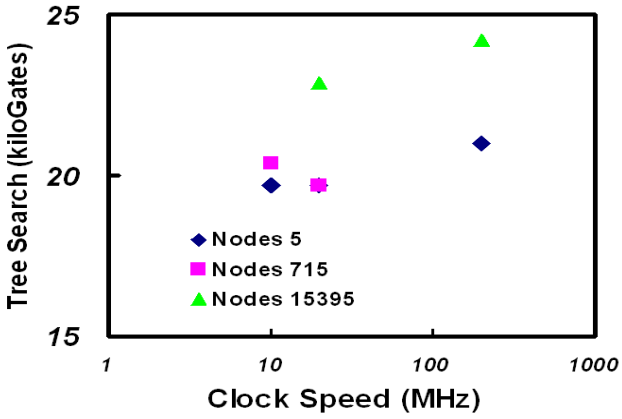

Fig. 3. Number of clock cycles to evaluate PWL controller, as a function of the chip clock speed. There is little overhead for increasing the clock speed up to 200 MHz. Higher speeds incur a significant penalty for memory access.

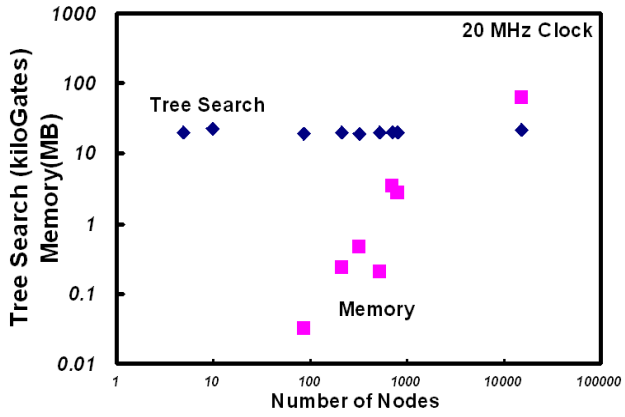

Fig. 4. The number of gates (diamonds) for the tree search and memory (squares) for the benchmark problems.

Figure 3 the dependence of the tree search clock cycles as a function of clock speed and number of nodes is shown. The dependence of the tree search on clock speed is not strong up to 200 MHz indicating that for these problems clock speed translates directly into improved control loop speed. It should be mentioned that for 200 MHz clock speeds, even the slowest benchmark problem can execute in about 1.1 μs.

D. Number of gates and chip circuit area

The number of gates as a function of problem dimensions are given in Table III and Figure 4. Basically, the number of gates (20 kGates) for the tree search does not depend strongly on the problem parameters. The number of gates for the function evaluation also does not increase much with problem complexity. This result occurs because, in all cases, we have fixed the performance at one loop iteration per clock cycle, and PICO has produced the least costly hardware that it can, while achieving this fixed (independent of parameters) throughput. Total latency, as indicated above, depends strongly on the parameters. If we were to change the performance requirement up or down, we would see an increase or a decrease in the gate count.

The size of the embedded SRAMs does not change, however, with performance, and these may dominate the chip cost. However, the number of SRAM cells scales directly with the number of search tree nodes, see Figure 4. For larger problems, this cost is by far the dominant one for implementing the chip and may be prohibitive for some applications.

PICO's output is a hardware design expressed in Verilog, which is an industry-standard hardware design language. In order to generate an ASIC implementation, the Verilog specification (combined with the rest of the specification of an entire system-on-chip design, normally) is input to a logic-synthesis step, which in turn creates a netlist, input to a place-and-route step. We used a standard logic synthesis tool, Synopsys DC Ultra, on the Helicopter $h = 2$ design. The process targeted was TSMC at 0.13 microns. Synopsys gives a process-specific estimate of chip area for an ASIC implementation. For this design, its estimate is 91,000 μ^2 (square microns); of this total, 53,000 μ^2 was for sequential circuit elements (the memory used for the hyperplace normals take up most of this in this, a memory-heavy case) and 38,000 μ^2 was for the combinational circuit elements, the gates.

E. Discussion

If the clock frequency is assumed to be 200 MHz, then the control loop execution time would range from 0.2 μs to 1.1 μs for the benchmark problems, see Table III and

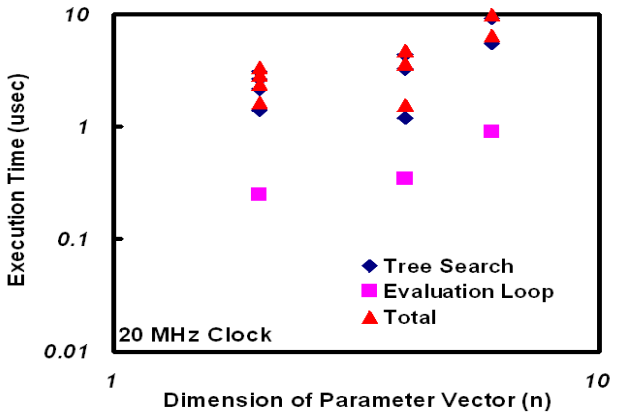

Fig. 5. Execution times for the various benchmark problems with a 20 MHz clock frequency. For 200 MHz clock rates the times would be about 10 times faster.

Figure 5. This result is rather important in that it indicates that control for mechanical, thermal, and acoustic time scales can be handled by these controllers with speed to spare, e.g. [16], [17], [18]. In particular, lower level constrained multidimensional controls in mechanical systems may readily be implemented using hardware implementation of explicit MPC. Because the complexity of the problem grows with the dimension of the parameter vector x and the MPC horizon h, [2], [14] these numbers should be minimized in order to improve the performance. The biggest cost factor appears to be the memory needed for storing the hyperplane normals; reducing the complexity of the PWL function representation is in fact an active area of research [42], [14], [4], [39], [40], [11]. Moreover, a tradeoff between computational and memory requirements can be made by constructing a binary search tree of less depth such that at each leaf node a number of candidate linear expression can be compared using a sequential search [43] or other evaluation methods [41].

IV. Conclusion

We have shown that small-scale explicit MPC solutions exactly represented as PWL functions can be efficiently implemented in an ASIC using about 20 kGates and resulting in execution times around 1 μs with a 200 MHz clock frequency. The binary search tree representation of polyhedral PWL mappings is the key data structure that allows an efficient binary tree search to be applied. The cost of implementation is largely determined by the requirement for memory to store the data structures needed to hold the PWL functions and the associated search tree. Methods for complexity reduction and PWL function approximation will greatly reduce implementation cost.

Acknowledgements

We thank Synfora, and in particular Darren Cronquist, for discussions and for his help with the hardware synthesis using PICO.

References

[1] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit solution of model predictive control via multiparametric quadratic programming," in *Proc. American Control Conference, Chicago*, 2000, pp. 872–876.

[2] ——, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3–20, 2002.

[3] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, pp. 489–497, 2003.

[4] P. Grieder, M. Kvasnica, M. Baotic, and M. Morari, "Low complexity control of piecewise affine systems with stability guarantee," in *American Control Conference, Boston*, 2004.

[5] T. A. Johansen, "On multi-parametric nonlinear programming and explicit nonlinear model predictive control," in *IEEE Conf. Decision and Control, Las Vegas, NV*, vol. 3, 2002, pp. 2768–2773.

[6] ——, "Approximate explicit receding horizon control of constrained nonlinear systems," *Automatica*, vol. 40, pp. 293–300, 2004.

[7] A. Bemporad, F. Borrelli, and M. Morari, "Optimal controllers for hybrid systems: Stability and piecewise linear explicit form," in *Proc. Conference on Decision and Control, Sydney*, 2000.

[8] T. A. Johansen, T. I. Fossen, and P. Tøndel, "Optimal constrained control allocation via multi-parametric programming," *J. Guidance, Control and Dynamics*, vol. 28, pp. 506–515, 2005.

[9] T. A. Johansen, T. P. Fuglseth, P. Tøndel, and T. I. Fossen, "Optimal constrained control allocation in marine surface vessels with rudders," in *IFAC Conf. Manoeuvring and Control of Marine Craft, Girona*, 2003.

[10] A. Bemporad and C. Filippi, "Suboptimal explicit RHC via approximate multiparametric quadratic programming," *J. Optimization Theory and Applications*, vol. 117, pp. 9–38, 2003.

[11] T. A. Johansen and A. Grancharova, "Approximate explicit model predictive control via orthogonal search tree," *IEEE Trans. Automatic Control*, vol. 48, pp. 810–815, 2003.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, pp. 509–517, 1975.

[13] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry, 2nd edition*. Springer-Verlag, Berlin, 2000.

[14] P. Tøndel and T. A. Johansen, "Complexity reduction in explicit model predictive control," in *Preprints, IFAC World Congress, Barcelona*, 2002.

[15] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, pp. 945–950, 2003.

[16] A. Bemporad, F. Borrelli, L. Glielmo, and F. Vasca, "Optimal piecewise-linear control of dry clutch engagement," in *3rd IFAC Workshop on Advances in Automotive Control, Karlsruhe, Germany*, 2001.

[17] F. Borelli, A. Bemporad, M. Morari, M. Fodor, and D. Hrovat, "A hybrid approach to traction control," in *Proc. Hybrid Systems, Computation and Control, Rome*, 2001.

[18] P. Tøndel and T. A. Johansen, "Lateral vehicle stabilization using constrained optimal control," in *Proc. European Control Conference, Cambridge, UK*, 2003.

[19] G. Papafotiou, T. Geyer, and M. Morari, "Hybrid modelling and optimal control of switch-mode dc-dc converters," in *IEEE Workshop on Computers in Power Electronics (COMPEL), Champaign, IL*, 2004.

[20] W. Jackson, M. P. J. Fromherz, D. K. Biegelsen, J. Reisch, and D. Goldberg, "Constrained optimization based control of real time large-scale systems: Airjet object movement system," in *Proc. IEEE Conf. Decision and Control, Orlando*, 2001.

[21] R. Ross, "Startup company will offer model based predictive control on a single chip," *Control Engineering Europe*, pp. 8–9, November 2003.

[22] ——, "Revolutionising model-based predictive control," *IEE Computing and Control Engineering*, pp. 26–29, December 2003.

[23] P. Dua, V. Sakizlis, L. Kershenbaum, and E. Pistikopoulos, "Model-based parametric controller for the operation of an experimental reactor," in *ESCAPE 14, Lisbon, Portugal*, 2004, pp. 337–340.

[24] E. N. Pistikopoulos, "On-line optimization via off-line optimization! - a guided tour to parametric programming and control," in *IFAC Symposium on Dynamics and Control of Process Systems (DYCOPS-7), Boston*, 2004.

[25] P. Dua, "Model based and paramtric control for drug delivery systems," Ph.D. dissertation, Imperial College London, 2005.

[26] M. Cannon and B. Kouvaritakis, "Efficient constrained model predictive control with asymptotic optimality," *SIAM J. Optimization and Control*, vol. 41, pp. 60–82, 2002.

[27] B. Kouvaritakis, J. A. Rossiter, and J. Schuurmans, "Efficient robust predictive control," *IEEE Trans. Automatic Control*, vol. 45, pp. 1545–1550, 2000.

[28] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," *IEEE Trans. Automatic Control*, vol. 44, pp. 648–654, 1999.

[29] B. Kouvaritakis, M. Cannon, and J. A. Rossiter, "Who needs QP for linear system MPC anyway?" *Automatica*, vol. 38, pp. 879–884, 2002.

[30] L. G. Bleris and M. V. Kothare, "Real-time implementation of model predictive control," in *Proc. American Control Conference, Portland, OR*, 2005, pp. 1752–1757.

[31] G. Hassapis, "Implementation of model predictive control using real-time multiprocessing computing," *Microprocessors and Microsystems*, vol. 27, pp. 327–340, 2003.

[32] M. H. He and K. V. Ling, "Model predictive control on a chip," in *The 5th International Conference on Control and Automation, Hungary, Budapest*, 2005, pp. 528–531.

[33] O. A. Palusinski, S. Vrudhula, L. Znamirowski, and D. Humbert, "Process control for microreactors," *Chemical Engineering Progress*, pp. 60–66, 2001.

[34] D. Henrikson, A. Cervin, J. Akesson, and K. Arzen, "Feedback schedule of model predictive controllers," in *Proc. 8th IEEE Real-time and embedded technology and applications symposium, San Jose, CA*, 2002.

[35] L. G. Bleris, M. V. Kothare, J. G. Garcia, and M. G. Arnold, "Towards embedded model predictive control for system-on-a-chip applications," *J. Process Control*, vol. 16, pp. 255–264, 2005.

[36] ——, "Embedded model predictive control for system-on-a-chip applications," in *Proceedings of the 7th IFAC Symposium on Dynamics and Control of Process Systems (DYCOPS-7), Boston, MA*, 2004.

[37] J. G. Garcia, M. G. Arnold, L. G. Bleris, and M. V. Kothare, "LNS architectures for embedded model predictive control processors," in *Int. Conf. Compilers, Architectures and Synthesis for Embedded Systems, Washington DC*, 2004, pp. 79–84.

[38] R. Schreiber, S. Aditya, S. Mahlke, V. Kathail, B. R. Rau, D. Cronquist, and M. Sivaraman, "PICO-NPA: High-level synthesis of nonprogrammable hardware accelerators," *Journal of VLSI Signal Processing*, vol. 31, pp. 127–142, June 2002.

[39] T. Geyer, F. Torrisi, and M. Morari, "Optimal complexity reduction of piecewise affine models based on hyperplane arrangements," in *American Control Conference, Boston*, 2004, pp. 1190 – 1195.

[40] A. Bemporad, K. Fukuda, and F. D. Torrisi, "Convexity recognition of the union of polyhedra," *Computational Gemometry*, vol. 18, pp. 141–154, 2001.

[41] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari, "Efficient on-line computation of explicit model predictive control," in *Proc. IEEE Conf. Decision and Control, Orlando*, vol. 2, 2001, pp. 1187–1192.

[42] P. Tøndel, T. A. Johansen, and A. Bemporad, "Computation and approximation of piecewise affine control via binary search tree," in *IEEE Conf. Decision and Control, Las Vegas, NV*, vol. 3, 2002, pp. 3144–3149.

[43] T. A. Johansen, I. Petersen, and O. Slupphaug, "Explicit suboptimal linear quadratic regulation with input and state constraints," *Automatica*, vol. 38, pp. 1099–1112, 2002.