A Comparison of Interior Point and Active Set Methods for FPGA Implementation of Model Predictive Control

Mark S. K. Lau, S. P. Yue, K. V. Ling and J. M. Maciejowski

Abstract— **A key component of model predictive control (MPC) is the solving of quadratic programming (QP) problems. Interior point method (IPM) and active set method (ASM) are the most commonly employed approaches for solving general QP problems. This paper compares several performance aspects of the two methods when they are implemented on a FPGA for MPC applications. We compare the computational complexity, storage, convergence speed, and some practical implementation issues. We find that, in general, ASM gives lower complexity and converges faster when the numbers of decision variables and constraints are small. Otherwise, IPM should be a better choice due to its scalability. We also note occasional instability of both IPM and ASM when they are implemented in our FPGA, which uses single precision floating point arithmetic. The instability is mainly due to numerical error, which is found to be more serious in ASM than IPM in our current implementations.**

I. Introduction

Model Predictive Control (MPC) has become an established control technology due to its capability of handling problems with physical constraints. Early applications of MPC were in the petrochemical industry. But its use has now been proposed for a wide variety of industries, such as ships [1], aerospace [2], road vehicles [3] and also micro scale devices [4].

The ability to solve MPC problems online becomes critical for applications that require fast response time, and for embedded applications with limited computational resource. Reconfigurable hardware, such as Field Programmable Gate Array (FPGA), is now a promising platform for deploying MPC for fast dynamic systems. The encapsulation of constrained MPC algorithms as suitable modules for embedded control has been investigated in [5], where a Handel-C implementation of an MPC algorithm was described and realized on a modest Xilinx Spartan 3L(XC3S1500L-4-fg320) FPGA. The computation carried out in the FPGA design of [5] was sequential. The opportunity for parallel computation for high speed applications was then exploited in [6].

One major component of MPC is the solving of a quadratic programming (QP) problem. Interior Point Method (IPM) and Active Set Method (ASM) are the most commonly employed approaches for solving general QP problems. The capacity of both methods for MPC applications have been investigated extensively in the literature [7], [8], [9].

In this paper, we compare several performance aspects of an IPM and an ASM when they are implemented on a FPGA for MPC applications. We compare their computational complexity, storage, speed of convergence, and some practical implementation issues.

Computational complexity is quantified by the number of floating point arithmetic operations (addition, multiplication, division) to be executed per iteration, while storage is measured by the number of floating point variables to be stored during the execution. Expressions for estimating the worst case complexity and storage are given. By extensive experiments, we examine the scalability of IPM and ASM when implemented in our FPGA. We observe how the speed of convergence changes with the size of QP problems.

We find that, in general, ASM performs better than IPM when the numbers of variables and constraints are small. Otherwise, IPM should be a better choice because of its scalability. However, we also note occasional instability of IPM and ASM when they are implemented in our FPGA, which is a single precision floating point system. The instability is mainly due to numerical error, which is found to be more serious for ASM than IPM in our current implementations. It is hoped that the results presented in this paper can provide some intuition for implementing IPM and ASM on FPGAs for MPC applications.

The paper is organized as follows: In the next section, the MPC problem is introduced, which is then converted into a standard QP problem. In Section III, the IPM and ASM algorithms that we adopt in our FPGA implementation are described. Details of the implementation are then discussed in Section IV. The main results are presented in Section V, which is followed by an aircraft control example for illustration in Section VI. Finally, a conclusion of the present paper and possible future works are given in Section VII.

II. Constrained MPC Problem

Consider a discrete linear time-invariant plant:

$$\begin{aligned} x(k+1) &= Ax(k) + B\Delta u(k), \\ y(k) &= Cx(k) \end{aligned}$$

where $y(k) \in \mathbb{R}^p$, $u(k) \in \mathbb{R}^m$ and $x(k) \in \mathbb{R}^n$ are the system output, input and internal states, respectively. The constrained MPC problem is to minimize the cost function

$$f(\Delta u) = \sum_{j=1}^{N_p} \|y(k+j) - w(k+j)\|^2 + \sum_{j=1}^{N_u-1} \|\Delta u(k+j)\|^2$$

subject to linear inequality constraints on the system outputs, inputs and states. Here w is the set point, and N_p and N_u are the prediction and control horizons, respectively.

We follow a standard approach to convert the MPC problem into a QP problem [5]-[6], [8], [10]. First, we define

$$\Psi_u = \begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & \cdots & CA^{N_p-N_u}B \end{bmatrix},$$

$$\Psi_x = \begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{N_p} \end{bmatrix} \quad \text{and} \quad z = \begin{bmatrix} \Delta u(k) \\ \Delta u(k+1) \\ \vdots \\ \Delta u(k+N_u-1) \end{bmatrix}.$$

This work was supported by A*STAR project "Model Predictive Control on a Chip", Ref: 052-118-0059.

Mark S. K. Lau, S. P. Yue and K. V. Ling are with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798. E-mail: {marklausk, spyue, ekvling}@ntu.edu.sg

M. Maciejowski is with Engineering Department, Cambridge University, Cambridge CB2 1PZ, UK. E-mail: jmm@eng.cam.ac.uk

978-3-9524173-9-3

The constrained MPC problem reduces to a QP problem:

$$\min_{z\in\mathbb{R}^{n_v}} \left\{ \frac{1}{2} z'Qz + c'z : Jz \leq g \right\}. \quad (1)$$

In the above equation, $n_v = mN_u$ denotes the number of decision variables, $Q = 2\Psi_u'\Psi_u + 2I$ is an $n_v \times n_v$ positive definite matrix, and $c = 2\Psi_u'(\Psi_x x_k - w)$ is an $n_v \times 1$ vector. Moreover J and g have the sizes $m_c \times n_v$ and $m_c \times 1$, respectively, where m_c is the total number of constraints.

The approach mentioned above can be replaced by the approach of Wright [7] (pp. 91 of [10]), where the states and inputs during the prediction horizon are kept as variables, in order to get a banded Q. This results in faster solution of the linear system of equations in both IPM and ASM for large problems. We do not consider that approach in this paper because we are assuming that our MPC problems are not large enough for that approach to pay off.

III. Optimization Algorithms

A. Interior Point Method

The idea of IPM is to approach the solution of the Karush-Kuhn-Tucker (KKT) equations by successive descent steps. Each descent step is a Newton-like step and is obtained by solving a system of linear equations. Each interior point iteration can be expensive to compute, but can make significant progress towards the solution [11].

According to an infeasible interior point framework of [7], the QP problem (1) can be solved by the algorithm below:

1: Select an initial condition (z_0, λ_0, t_0) with positive (λ_0, t_0) and select a positive termination threshold δ
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad \Lambda \leftarrow \mathrm{diag}(\lambda_k)$ and $T \leftarrow \mathrm{diag}(t_k)$
4: $\quad$ Solve for Δz with

$$(Q + J'\Lambda T^{-1} J)\Delta z = -r_d - J'T^{-1}\Lambda(-r_b - t_k + \sigma\mu_k \Lambda^{-1} e)$$

5: $\quad$ Calculate $\Delta\lambda = \Lambda T^{-1}(J\Delta z_k - r_b) - \Lambda + \sigma\mu_k T^{-1} e$ and then $\Delta t = -t_k + \Lambda^{-1}(\sigma\mu_k e - T\Delta\lambda)$
6: $\quad \alpha \leftarrow \min\left\{ \min_{i\in\mathcal{I}(\Delta t)} \left\{ -\frac{t_k(i)}{\Delta t(i)} \right\}, \min_{i\in\mathcal{I}(\Delta\lambda)} \left\{ -\frac{\lambda_k(i)}{\Delta\lambda(i)} \right\} \right\}$
7: $\quad (z_{k+1}, \lambda_{k+1}, t_{k+1}) \leftarrow (z_k, \lambda_k, t_k) + \alpha(\Delta z, \Delta\lambda, \Delta t)$
8: $\quad \mu_{k+1} \leftarrow t_{k+1}'\lambda_{k+1} m_c^{-1}$
9: $\quad$ **if** $\mu_{k+1} \leq \delta$ **then**
10: $\quad\quad$ Terminate with solution $z^* = z_{k+1}$
11: $\quad$ **end if**
12: **end for**

We shall refer to the above algorithm as the IPM algorithm. Here,

$$\begin{aligned} r_d &= Qz_k + J'\lambda_k + c, \\ r_b &= -Jz_k + g - t_k, \\ e &= (1, 1, \ldots, 1) \in \mathbb{R}^{m_c \times 1}, \\ \mathcal{I}(\Delta t) &= \{i = 1, 2, \ldots, m_c : \Delta t(i) < 0\}, \\ \mathcal{I}(\Delta\lambda) &= \{i = 1, 2, \ldots, m_c : \Delta\lambda(i) < 0\}, \end{aligned}$$

and $\sigma = 0.25$ is a pre-defined constant. The sub-optimality of z_k is measured by the quantity $\mu_k = t_k'\lambda_k m_c^{-1}$, where t_k and λ_k are iterates for the slack variables and Lagrange multipliers, and m_c is the number of constraints. As the iteration continues, μ_k is gradually reduced to zero and z_k converges to the optimal solution.

In Line 4 of the above IPM algorithm, a linear system in the form of $Ax = b$ has to be solved. In this paper, it is solved by Gauss-Jordan elimination with pivoting.

B. Active Set Method

ASM solves the QP problem (1) via identifying the active set of its solution z^*. The method begins with an initial guess $\mathcal{W}_0$ of the active set. In the next iteration, $\mathcal{W}_0$ is refined by deleting a constraint from $\mathcal{W}_0$ or adding a constraint to $\mathcal{W}_0$. In this way, a guess of the active set is refined iteratively until the exact active set is found.

Below is the active set algorithm for our FPGA implementation. It will be called the ASM algorithm:

1: Compute a feasible initial point z_0.
2: Let the initial working set $\mathcal{W}_0$ be empty.
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: $\quad$ Solve $\begin{bmatrix} Q & J_{\mathcal{W}_k}' \\ J_{\mathcal{W}_k} & 0 \end{bmatrix} \begin{bmatrix} \Delta z_k \\ \lambda \end{bmatrix} = \begin{bmatrix} -c - Qz_k \\ 0 \end{bmatrix}$ for $(\Delta z_k, \lambda)$
5: $\quad$ **if** $\Delta z_k = 0$ **then**
6: $\quad\quad$ **if** All entries of λ are not negative **then**
7: $\quad\quad\quad$ Terminate with solution $z^* = z_k$
8: $\quad\quad$ **else**
9: $\quad\quad\quad$ Remove from $\mathcal{W}_k$ an index that corresponds to the smallest entry of λ, and then $z_{k+1} \leftarrow z_k$
10: $\quad\quad$ **end if**
11: $\quad$ **else**
12: $\quad\quad \mathcal{D}_k \leftarrow \left\{ i \notin \mathcal{W}_k : J_i\Delta z_k > 0, \frac{g_i - J_i z_k}{J_i \Delta z_k} < 1 \right\}$
13: $\quad\quad$ **if** $\mathcal{D}_k$ is empty **then**
14: $\quad\quad\quad z_{k+1} \leftarrow z_k + \Delta z$ and $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k$
15: $\quad\quad$ **else**
16: $\quad\quad\quad \alpha \leftarrow \min_{i\in\mathcal{D}_k} \left\{ \frac{g_i - J_i z_k}{J_i \Delta z_k} \right\}$ and $z_{k+1} \leftarrow z_k + \alpha\Delta z$
17: $\quad\quad\quad$ Create $\mathcal{W}_{k+1}$ by adding one element of $\mathcal{D}_k$ to $\mathcal{W}_k$
18: $\quad\quad$ **end if**
19: $\quad$ **end if**
20: **end for**

In the ASM algorithm, J_i is the ith row of J, and $J_{\mathcal{W}_k}$ is the matrix obtained by removing the ith row from J for all $i \notin \mathcal{W}_k$.

If all entries of g are positive, then $(0, 0, \ldots, 0)'$ is feasible. In this case, we may choose the initial point $z_0 = 0$. However, if a feasible point is not available, we can obtain one by solving a linear programming problem (see Section 16.4 of [11] for instance). As in the IPM algorithm, we solve the linear system in Line 4 of the ASM algorithm by Gauss-Jordan elimination with pivoting.

IV. FPGA Implementation

The ASM and IPM algorithms are implemented on a Xilinx XC2V3000 FPGA on Celoxica RC203 prototyping board. The algorithms are coded in Handel-C using Celoxica DK design suite 4.0. The Celoxica DK design suite calls Xilinx ISE 9.1i for mapping, and place and route after compiling the Handel-C codes.

IEEE single precision floating point numbers are used for the implementation of both algorithms, so that as much numerical accuracy is preserved as possible, while keeping the word length for variables reasonably short. Celoxica's non-pipelined floating point library is used in this implementation. Each operation of addition, multiplication and subtraction requires 3 clock cycles to complete, while each division operation takes 30 clock cycles.

On-chip block RAMs are used for storage of arrays to reduce the use of FPGA logic for storage. This reduces the overall resources usage on the FPGA. However, as a trade-off, the execution time of the algorithms may becomes longer, because simultaneous access to the same block RAM is not allowed. Since on-chip block RAMs are used (rather than off-chip RAMs), there is no significant delay

when accessing data. In fact, it takes just one clock cycle to read or write to the on-chip block RAM.

The FPGA prototyping board is connected to a desktop PC through an RS232 serial communication cable. The desktop PC sets up the QP problems in Matlab, and sends the matrices to the FPGA through the RS232 cable. When the computation of the QP problem solution completes, the results are passed back to the desktop PC through the same RS232 serial communication link. The number of iterations and the total clock counts needed to solve each QP problem are also noted, and sent back to the desktop PC.

It was suggested in [12] that FPGA allows one to determine the precision with which each computation is performed, whether to use fixed or floating point. But, the effects of such decisions are not investigated in our FPGA implementation.

V. A Comparison of IPM and ASM

A. Computational Complexity

The expressions in Table I give the numbers of arithmetic operations executed by one iteration of the IPM and ASM algorithms, parameterized by the number of decision variables, n_v, and the number of inequality constraints, m_c (also see [13]).

In deriving these expressions, we assume that constraints are imposed in (1) to restrict the minimum and maximum values of the inputs, outputs, internal states, or their linear combinations. Since the constraints associated with the minimum and maximum cannot be active simultaneously, the total number of active constraints cannot exceed $m_c/2$.

The expressions in Table I are derived for the worst case scenarios. For example, we assume that the number of active constraints in the ASM algorithm is always $m_c/2$ so that the matrix $J_{\mathcal{W}_k}$ in Line 4 always has $m_c/2 \times n_v$ entries. But, in reality $J_{\mathcal{W}_k}$ may have a smaller size. Also, expressions in Table I do not take into account of overheads for, say, obtaining the initial feasible point z_0, floating point numbers comparison, finding of minimum values, and addition for increment of counters. An initial feasible point can be obtained by solving a linear programming problem [14].

In Table I $M_a(n_v)$, $M_m(n_v)$ and $M_d(n_v)$ are, respectively, the numbers of addition, multiplication and division required for solving the linear system in Line 4 of the IPM algorithm. For the ASM algorithm, where a linear system of size $n_v + m_c/2$ has to be solved, the number of addition, multiplication and division are $M_a(n_v + m_c/2)$, $M_m(n_v + m_c/2)$ and $M_d(n_v + m_c/2)$, respectively. Clearly M_a, M_m and M_d are determined by the user's choice of algorithm for solving Line 4. Because we use Gauss-Jordan elimination with pivoting, we have for $n = 1, 2, \ldots$

$$
\begin{aligned}
M_a(n) &= 0.5(n-1)n(n+1),\\
M_m(n) &= 0.5n^2(n+1),\\
M_d(n) &= n.
\end{aligned}
$$

For illustration, the total number of operations, $M_a + M_m + M_d$, is given in the left plot of Fig. 1. The ASM algorithm clearly requires a much higher computation cost. However, it should also be noted that Fig. 1 shows the worst case scenario. In practice, the gap between the curves of the IPM and ASM algorithms is usually much smaller.

For now, we ignore the arithmetic operations for solving the linear systems in both IPM and ASM algorithms. Then, the IPM algorithm requires approximately $n_v^2 m_c$ more additions, $n_v^2 m_c$ more multiplications, and $2m_c$ more divisions than the ASM algorithm. The extra operations are used for matrix multiplications in forming the linear system in Line 4 of the IPM algorithm (i.e., forming the matrix $J'\Lambda T^{-1}J$, and vectors r_d and r_b). The associated cost is often described as a "fixed cost" in the literature [15]. For illustration, we evaluate the expressions in Table I with the assumption $M_a = M_m = M_d = 0$, and show in Fig. 1 (right) the total number of operations required per iteration. In this case, the IPM algorithm clearly requires a much higher computation cost.

TABLE I
Worst Case Computation complexity and storage.

IPM algorithm	Number of operations/storage per iteration
Addition:	$n_v^2(m_c+1)+n_v(3m_c+1)+7m_c-1$ $+M_a(n_v)$
Multiplication:	$n_v^2(m_c+1)+n_v(4m_c+1)+7m_c+2$ $+M_m(n_v)$
Division:	$3m_c+1+M_d(n_v)$
Storage:	$n_v^2+2n_v+10m_c+19$
ASM algorithm	**Number of operations/storage per iteration**
Addition:	$n_v^2+n_v(2m_c+1)-m_c+M_a(n_v+0.5m_c)$
Multiplication:	$n_v^2+n_v(2m_c+1)+M_m(n_v+0.5m_c)$
Division:	$m_c+M_d(n_v+0.5m_c)$
Storage:	$n_v^2+n_v(2m_c+2)+m_c^2+m_c+9$

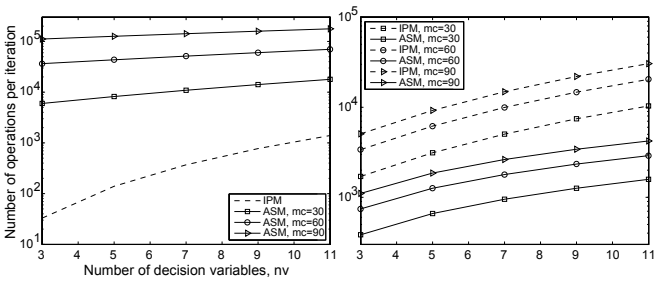

Fig. 1. Total number of operations per iteration for solving (left) the linear systems, and (right) one QP problem with $M_a = M_m = M_d = 0$.

B. Storage

Also shown in Table I are the numbers of 32-bit floating point variables needed to be stored on the FPGA chip, which is made up of configurable elements such as Look Up Tables (LUTs), Flip-Flops (FFs) and on-chip block RAMs. The variables are mainly used for storing intermediate calculation results. LUTs and FFs form part of the Configurable Logic Blocks (CLBs), which is mainly used for programming combinatorial and synchronous logic, while block RAMs are dual-port RAMs used mainly for storage.

Mapping of the variables to the block RAMs are done automatically by the compiler. Variables are not reused in the design process, as reusing of variables in hardware would create more difficulty in the routing of the design, and results in a slower attainable clock rate. Note that the storage given in Table I does not include variables used for variable swapping, loop indices and counters, flags, and the QP problem matrices. The storage required for the QP problem matrices are not included since it the same for both algorithms.

As seen from Table I, the ASM algorithm requires more storage, since it needs to solve a system of $n_v + m_c/2$ linear equations. In contrast, the IPM algorithm only needs to solve a system of n_v linear equations. Therefore, the ASM algorithm needs a larger storage for storing the matrices that are needed during the solving of the linear equations.

C. Convergence Speed

Now, we compare the average numbers of iterations required by the IPM and ASM algorithms for obtaining the solution of a QP problem. We use sequential FPGA implementations of the

algorithms to solve QP problems with different values of n_v and m_c, where n_v and m_c are the numbers of decision variables and constraints, respectively. For each $(n_v, m_c) \in \{3, 5, 7, \ldots, 13\} \times \{31, 41, 51, \ldots, 111\}$, one hundred randomly generated QP problems are input to the IPM algorithm. Then, for each of these QP problems, we record the number of iterations and computation time required to successfully obtain its solution. Finally, the average number of iteration is given by the total number of iterations divided by the number of successes. (The unsuccessful cases are mainly caused by numerical errors, which will be discussed in Section V-D.) The average computation time is computed in the same fashion. Then, the same set of one hundred QP problems is tested on the ASM algorithm.

The QP problems are randomly generated by the method proposed in [16]. QP problems generated by this method always have the solution $(1, 1, \ldots, 1)$ and a feasible point $(0, 0, \ldots, 0)$. Hence, we use the initial point $z_0 = (0, 0, \ldots, 0)$ for both of the IPM and ASM algorithms. Also, the number of active constraints at the solution never exceeds the number of decision variables. This is to ensure that the Lagrangian multipliers always exist at the solution. Finally, the QP problems are scaled such that the eigenvalues of Q lie in the range from 0.5 to 20. The QP problems are generated by a PC using Matlab, and then sent through RS232 to the RC203 prototyping board.

Figures 2 (a) and (b) present the average number of iterations required to solve a single QP problem. For the IPM algorithm, the average number of iterations lies between 10.5 and 13.5 for most values of (n_v, m_c) and is insensitive to n_v and m_c. For the ASM algorithm, the average number of iteration grows roughly linearly with n_v and m_c, and almost reaches 50 for $(n_v, m_c) = (13, 111)$.

Hence, we can conclude that the IPM algorithm has a better scalability than the ASM algorithm, as the average number of iterations required by the former to solve one QP problem weakly depends on the numbers of constraints and decision variables.

It is well-known that ASM has an exponential number of iterations in the worst case, whereas IPM has only a polynomial number. But, in practice the iteration count needed by ASM may be much lower than the worst-case number of iterations. This is also evident from Figs. 2 (a) and (b).

Next, we examine the average computation time required to solve a single QP problem. The results are given in Figs. 2 (c) and (d). Our first observation is that, although we have just seen that the average number of iterations by the IPM algorithm is insensitive to n_v and m_c, the average computation time indeed shows a sharp increasing trend. This is because the number of arithmetic operations required by the IPM algorithm per iteration grows rapidly with n_v and m_c, as seen from Table I.

Second, as shown in Fig. 2 (c), the IPM algorithm requires a longer computation time than the ASM algorithm when n_v does not exceed nine. This is due to the expensive "fixed costs" of the IPM algorithm, which can have a significant influence on the computation cost when either n_v or m_c is small.

But, when n_v is larger than nine, the ASM algorithm requires a longer computation time. This is contributed partially by the large number of iterations required by the ASM as we have seen in Fig. 2 (a), and partially by the overwhelming number of arithmetic operations required in each iteration for solving the linear system.

In the above experiments, the FPGA implementation of the IPM algorithm is sequential. Actually, we have also repeated the experiments using a parallel version of the same IPM algorithm. See [6] for the implementation details. We found that the parallel version took only about half of the computation time shown in Figs. 2 (c) and (d). We have also compared the efficiency of the parallelized IPM and the (sequential) ASM algorithms. We found that the parallelized IPM algorithm took a shorter average computation time to solve a QP problem whenever n_v is only larger

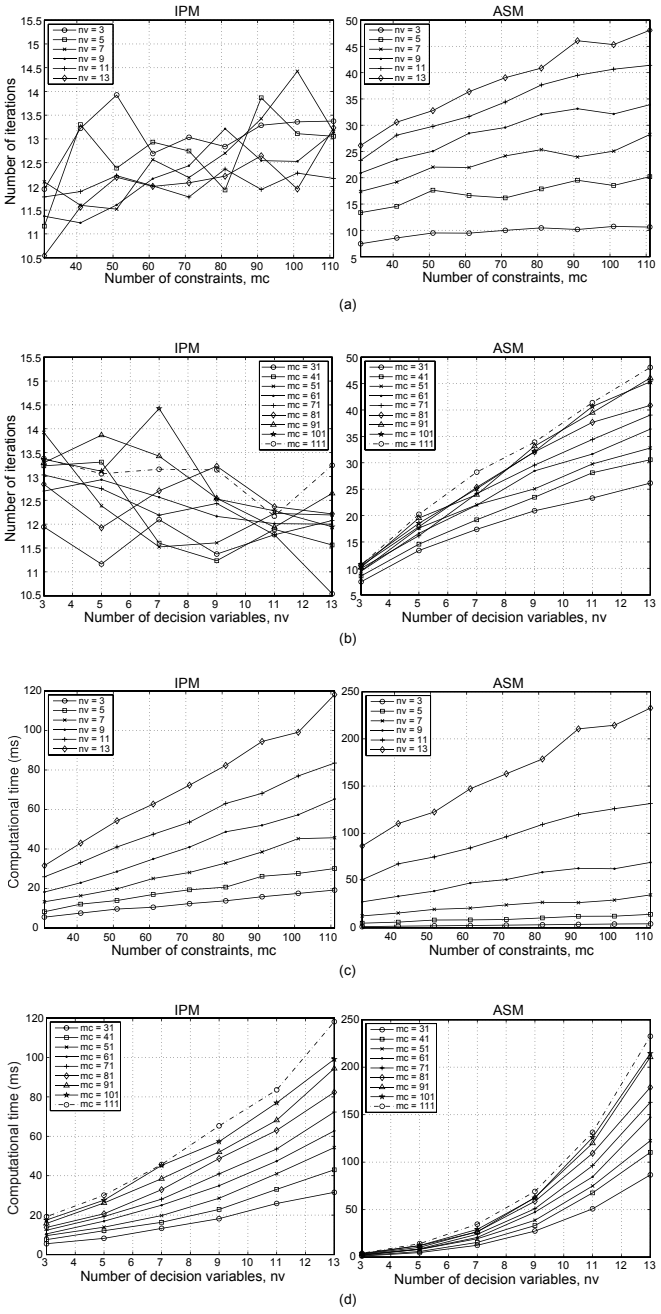

Fig. 2. Average number of iterations per QP problem versus (a) number of constraints; (b) number of decision variables. Average computation time per QP problem versus (c) number of constraints; (d) number of decision variables.

TABLE II
RESOURCE USAGE AND ACHIEVABLE CLOCK RATE.

Resources	Interior point	Active set
Requested clock rate	25 MHz	25 MHz
LUTs	8,755 (30%)	8,773 (30%)
FFs	2,153 (7%)	3,540 (12%)
Block RAMs	22 (22%)	59 (61%)

than five, in contrast to nine for the sequential IPM algorithm as discussed in the previous paragraph.

We remark that Gauss-Jordan elimination was used to solve the linear systems in the above experiments. This method may not be the most efficient method for solving linear systems. However, it was sufficient for our purpose of comparing the IPM and ASM algorithms. We expect to see the same trends as shown in Figs. 2 (a)-(d) if another method was used in the experiments to solve the linear systems.

In the above-mentioned experiments, the resource is allocated as shown in Table II. The same amount of resource is allocated for each value of (n_v, m_c), and the resource usage shown in Table II is based on the largest QP problem size, i.e., $n_v = 13$ and $m_c = 111$. From Table II, it can be seen that the ASM and IPM algorithms consumes similar amount of LUTs and FFs. But for block RAMs, the ASM algorithm needs almost three times as many block RAMs as the IPM algorithm. This is because the ASM algorithm may need to solve a system of $n_v + m_c/2$ equations, while IPM only needs to solve a system of n_v equations.

D. Numerical Error

In our FPGA implementation, numerical values are represented using an IEEE 32-bit floating point format. In some occasions, numerical error occurs and then propagates through the iterations. This may eventually cause unboundedness of the iterates or instability of the IPM or ASM algorithms.

On average, for about 3 out of 2000 randomly generated QP problems, the IPM algorithm either results in an unbounded z_k, or terminates at a point where the objective function value deviates too severely from the optimal value. An explanation for the failure cases is as follows: In each iteration $1/t_k(i)$ has to be computed in order to form the inverse of T. Recall that $t_k(i)$ is the slack variable for the ith inequality constraint. Hence, if the ith constraint is active at the solution, $t_k(i)$ would approach to zero as iteration goes on. Then taking the inverse $1/t_k(i)$ may cause severe numerical error. Implementation of IPM on single precision systems and the associated difficulty of numerical error have been mentioned in [17].

Compared to the IPM algorithm, the failure rate for the ASM algorithm is much higher, which is about 73 out of 2000 QP problems on average. Again, the failure cases are mainly due to numerical error, which usually occurs when the linear system is solved for a very small Δz_k. Unfortunately, the convergence of the ASM algorithm indeed requires Δz_k to be close to a zero vector. This potentially leads to a high failure rate.

VI. An MPC Example

We consider an example of Cessna Citation 500 Aircraft, which is taken from Example 2.7 of [10]. The aircraft has the following continuous time state-space model:

$$A = \begin{bmatrix} -1.2822 & 0 & 0.98 & 0 \\ 0 & 0 & 1 & 0 \\ -5.4293 & 0 & -1.8366 & 0 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} -0.3 \\ 0 \\ -17 \\ 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -128.8 & 128.8 & 0 & 0 \end{bmatrix}.$$

The model has the elevator angle (rad) as its only input, and the pitch angle (rad), altitude (m) and altitude rate (m/sec) as its outputs. The elevator angle is limited to $\pm 15°$ (± 0.262 rad) and the elevator slew rate is limited to $\pm 30°$/sec (± 0.524 rad/sec). These are limits imposed by equipment design and cannot be exceeded.

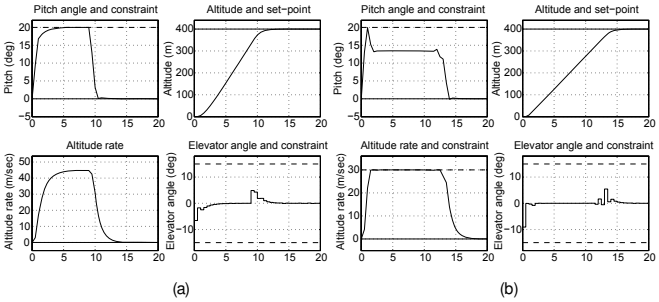

Fig. 3. Response of aircraft by ASM (a) without altitude rate constraints; (b) with altitude rate constraints. The horizontal axes are time in seconds.

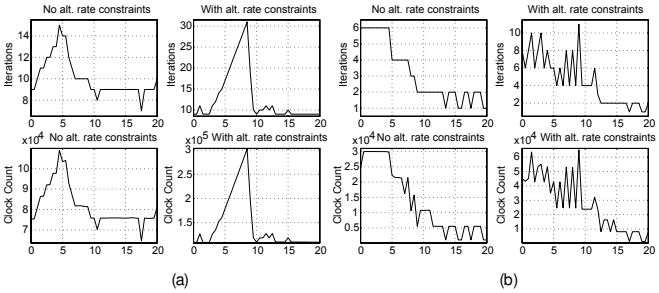

Fig. 4. Iteration and clock counts for (a) the IPM algorithm; (b) the ASM algorithm. The horizontal axes are time in seconds.

TABLE III
Resource usage and achievable clock rate.

Resources	Interior point	Active set
Requested clock rate	25 MHz	25 MHz
LUTs	14,688 (51%)	8,301 (28%)
FFs	2,802 (9%)	2,954 (10%)
Block RAMs	19 (22%)	44 (45%)

TABLE IV
Computation time per QP problem in the aircraft simulation.

Aircraft QP problem	Interior point	Active set
Without altitude rate constraint	3.26 ms	0.55 ms
With altitude rate constraint	5.71 ms	1.10 ms

For passenger comfort, the pitch angle limit is $\pm 20°$ (± 0.349 rad). The set points for pitch angle and altitude rate are held at zero meter, and the set point for altitude is held at 400 meters. Namely, $w(k+j) = \begin{bmatrix} 0 & 400 & 0 \end{bmatrix}'$.

A MPC controller is designed with sampling interval of 0.5 second, $N_p = 10$ and $N_u = 3$. The constraints are imposed:

$$|u| \leq 0.262, \; |\Delta u| \leq 0.524, \quad \text{and} \quad |y_1| \leq 0.349.$$

For this MPC problem, each QP problem has 32 constraints.

A. Simulation Results

We tackle the above-mentioned MPC problem with a FPGA implementation of the ASM algorithm. The objective is to justify our FPGA implementation of the algorithm. The simulation result is shown in Fig. 3 (a). It clearly shows that all outputs (pitch angle, altitude and altitude rate) satisfy the given constraints (dashed lines), and converge gradually to the set points (solid horizontal lines).

The altitude rate was not constrained in Fig. 3 (a). We now tackle the same MPC problem with altitude rate constrained to 30 m/sec (see the third case of Example 2.7 of [10]). The number of constraints is now increased to 52. Again, our FPGA implementation

of the ASM algorithm is able to solve the MPC problem without violating the given constraints, as shown in Fig. 3 (b).

B. Resource Usage

LUTs and FFs on a FPGA are used for logic implementation. It is shown in Table III that the IPM algorithm requires almost twice as much as that is required by the ASM algorithm for the above example. This is because, unlike the FPGA implementation used in Section IV with resource usage shown in Table II, in this Section, the IPM algorithm is implemented with parallelism and pipelining [6]. As a result, there are totally 2 floating point adders, 3 floating point multipliers, 3 floating point subtracters, and 1 floating point divider in the FPGA implementation of the IPM algorithm. These floating point operations takes up additional FPGA resources in the implementation of the IPM algorithm. In contrast, ASM algorithm is purely sequential and uses only 1 floating point adder, 1 floating point multiplier, 1 floating point subtractor, 1 floating point divider.

Although the ASM algorithm requires less LUTs and FFs, it does requires more block RAMs, as seen from Table III. Here, the block RAM are used to store intermediate floating point matrices. Because the ASM algorithm needs to solve linear systems of larger size, more block RAMs are needed.

The resource usage reflected in Table III is of the larger of the two aircraft examples, i.e., QP problems of 3 decision variables and 52 constraints.

C. Speed Performance

The clock speed for both implementations of the IPM and ASM algorithms are set to 25MHz. It is shown in Table IV that the ASM method is about 5 to 6 times faster than the IPM algorithm. This indeed is consistent with our observation in Section V-A that the IPM algorithm usually requires higher computation cost to solve a QP problem when the number of decision variable is small. Finally Fig. 4 (a) and (b) show the number of iterations and clock counts required for solving the aircraft simulation examples.

VII. Conclusion and Future Works

In this paper, we have compared the performances of interior point method (IPM) and active set method (ASM) for a FPGA implementation for MPC applications. We have compared the computational complexity, storage, and speed of convergence of the methods. We have found that, in general, ASM should perform better than IPM when the numbers of variables and constraints are small. Otherwise, IPM should be a better choice due to its scalability. However, we have also noted occasional instability of both IPM and ASM when it is implemented in our FPGA, which is a single precision system. The instability was mainly due to numerical error, which was found to be more serious in ASM than in IPM in our current implementations.

We mention a few directions worthy of further pursuit:

1) In the analysis of this paper, we have ignored the computation cost for obtaining an initial feasible point for the ASM algorithm. If the feasible set is a hyper-cube, then the initial feasible point can be obtained by simply selecting the center of the hyper-cube. Otherwise, the initial feasible point needs to be obtained by, for example, solving a linear programming problem. To estimate the influence of the overhead on the overall performance of the ASM algorithm, a study of the computation complexity is necessary.
2) In this paper, we have used Gauss-Jordan elimination to solve the linear systems in both IPM and ASM algorithms. Since the major portion of computation cost by IPM or ASM is spent on solving the linear system, it is worthwhile to see if a more efficient method (for example, LU decomposition) can be adopted. Also, we have observed that numerical error often occurs when the linear systems are being solved. The numerical error may be alleviated by, say, a minimum residual method, for solving the linear systems.
3) We may adopt the method of [7] to formulate our MPC problem into a QP problem. In this method, the states and inputs during the prediction horizon are kept as variables, in order to get a banded Q. For large QP problems, this may results in lesser storage, and faster solution of the linear system of equations in both IPM and ASM.
4) In MPC applications, constraints are usually imposed to restrict the minimum and maximum values of the inputs, outputs, or internal states. This may result in a sparse J with special structures. It is meaningful to see if these structures can be exploited to reduce the storage and computational complexity of solving the linear systems.

References

[1] T. Perez, G. C. Goodwin, and C. W. Tzeng, "Model predictive rudder rolls stabilization control for ships," in *Proceedings of 5th IFAC Conference on Manoeuvring and Control of Marine Craft*, 2000.

[2] A. Richards and J. P. How, "Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility," in *Proceedings of the 2003 American Control Conference*, vol. 5, 2003, pp. 4034–4040.

[3] M. Morari, M. Baotic, and F. Borrelli, "Hybrid systems modeling and control," *European Journal of Control*, vol. 9, no. 2-3, pp. 177–189, 2003.

[4] L. G. Bleris, J. G. Garcia, M. G. Arnold, and M. V. Kothare, "Model predictive hydrodynamic regulation of microflows," *Journal of Micromechanics and Microengineering*, vol. 16, pp. 1792–1799, 2006.

[5] K. V. Ling, S. P. Yue, and J. M. Maciejowski, "A FPGA implementation of model predictive control," in *Proceedings of the 2006 American Control Conference*, 2006, pp. 1930–1935.

[6] K. V. Ling, B. F. Wu, and J. M. Maciejowski, "Embedded model predictive control (MPC) using a FPGA," in *Proceedings of the 17th IFAC World Congress*, 2008, pp. 15 250–15 255.

[7] S. J. Wright, "Applying new optimization algorithms to model predictive control," in *Proceedings of Chemical Process Control V*, 1997, pp. 147–155.

[8] 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*, vol. 99, no. 3, pp. 723–757, December 1998.

[9] R. A. Bartlett, L. T. Biegler, J. Backstrom, and V. Gopal, "Quadratic programming algorithms for large-scale model predictive control," *Journal of Process Control*, vol. 12, no. 7, pp. 775–795, 2002.

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

[11] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[12] G. Constantinides, "Precision analysis for fixed-point computation," in *Reconfigurable Computing*, S. Hauck and A. DeHon, Eds. Morgan Kaufmann, 2007, ch. 23, pp. 475–501.

[13] S. P. Yue, "A methodology for implementing constrained MPC on reconfigurable hardware," School of Electrical and Electronic Engineering, Nanyang Technological University, Tech. Rep., February 2007.

[14] R. Fletcher, *Practical Methods of Optimization*, 2nd, Ed. John Wiley & Sons, 1987.

[15] R. A. Bartlett, A. Wächter, and L. T. Biegler, "Active set vs. interior point strategies for model predictive control," in *Proceedings of the 2000 American Control Conference*, vol. 6, 2000, pp. 4229–4233.

[16] M. L. Lenard and M. Minkoff, "Randomly generated test problems for positive definite quadratic programming," *ACM Transactions on Mathematical Software*, vol. 10, no. 1, pp. 86–96, March 1984.

[17] J. H. Jung and D. P. O'Leary, "Implementing an interior point method for linear programs on a CPU-GPU system," *Electronic Transactions on Numerical Analysis*, vol. 28, pp. 174–189, 2008.