FPGA Implementation of Mixed Integer Quadratic Programming Solver for Mobile Robot Control

Yusuke Shimai, Junichi Tani, Hiroki Noguchi, Hiroshi Kawaguchi, and Masahiko Yoshimoto

Department of Computer Science and Systems Engineering, Kobe University
Kobe, 657-8501 Japan
simai@cs28.cs.kobe-u.ac.jp

***Abstract*—We propose a high-speed mixed integer quadratic programming (MIQP) solver on an FPGA. The MIQP solver can be applied to various optimizing applications including real-time robot control. In order to rapidly solve the MIQP problem, we implement reusing a first solution (first point), pipeline architecture, and multi-core architecture on the single FPGA. By making use of them, we confirmed that 79.5% of the cycle times are reduced, compared with straightforward sequential processing. The operating frequency is 67 MHz, although a core 2 duo PC requires 3.16 GHz in processing the same size problem. The power consumption of the MIQP solver is 4.2 W.**

I. Introduction

Recently, control for robots has attracted people. We expect that the robot enriches people living, along with improving the robot control. There have been many researches on it; the hybrid system control by solving mixed integer quadratic programming (MIQP) is one of them.

The hybrid system control can be applied to various systems. We can take control of a fuel cell reactor and gas turbine [1], control of multi-vehicle pass planning [2], and robot manipulation [3], for instance. Ref. [4] derives the hybrid system from solving an MIQP problem. However, in general, it takes long time to solve the MIQP problem, and thus a high-speed MIQP solver is demanded.

The MIQP is quadratic programming (QP) that includes integer variables. The MIQP is, however, an NP-hard problem; the computational complexity exponentially increases with the number of the variables in practice. Fig. 1 shows computing time examples when the number of the variables is varied.

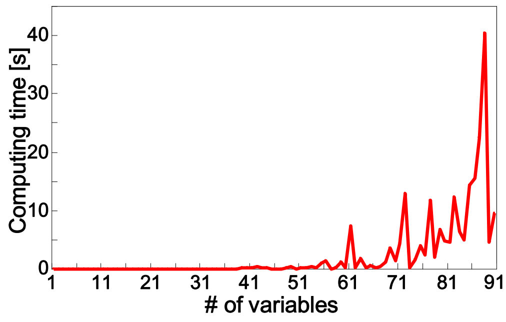


Fig. 1. Computing time of MIQP when the number of variables is changed.

The computation cost of the MIQP problem is high, but the hybrid system control with the MIQP solver gives a mobile robot intelligence; complex planning and movements can be supported online by that. In addition, the low-power calculation is required. That is, the high-speed but low-power MIQP solver is important for the mobile robot application.

In this paper, we propose a high-speed MIQP solver that can be applied to the hybrid system control. The MIQP solver is implemented on an FPGA, and its power is 4.2 W.

II. Overview of MIQP Solver

A. Definition of MIQP

QP is an optimization problem whose objective function is a quadratic function. The QP is known as a non-linear programming problem. Further, the QP involves integer variables, it becomes MIQP, which is categorized as an NP-hard problem.

The mathematical definition of the MIQP problem is:

$$\begin{aligned}
&\underset{\boldsymbol{x}}{\text{minimize}} && f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\mathbf{H}\boldsymbol{x} + \mathbf{g}^{\mathrm{T}}\boldsymbol{x} \\
&\text{subject to} && \mathbf{A}_{\mathrm{E}}\boldsymbol{x} = \mathbf{b}_{\mathrm{E}} \\
& && \mathbf{A}_{\mathrm{I}}\boldsymbol{x} \le \mathbf{b}_{\mathrm{I}} \\
&\text{where} && \boldsymbol{x} \in \mathbf{R}^{\mathrm{n_c}} \times \mathbf{Z}^{\mathrm{n_b}}, \mathbf{H} \in \mathbf{R}^{\mathrm{n \times n}} \\
& && \mathbf{A}_{\mathrm{E}} \in \mathbf{R}^{\mathrm{m_E \times n}}, \mathbf{A}_{\mathrm{I}} \in \mathbf{R}^{\mathrm{m_I \times n}} \\
& && \mathbf{b}_{\mathrm{E}} \in \mathbf{R}^{\mathrm{m_E}}, \mathbf{b}_{\mathrm{I}} \in \mathbf{R}^{\mathrm{m_I}}.
\end{aligned} \tag{1}$$

where $\boldsymbol{x}$ is a variable vector, and $\mathrm{n} = \mathrm{n_c} + \mathrm{n_b}$. $\mathrm{n_c}$ is the number of continuous variables, and $\mathrm{n_b}$ is the number of integer variables $\mathbf{H}$ is a symmetric positive definite matrix. $\mathbf{g}$, $\mathbf{b}_{\mathrm{E}}$, $\mathbf{b}_{\mathrm{I}}$, $\mathbf{A}_{\mathrm{E}}$, and $\mathbf{A}_{\mathrm{I}}$, are given as constant vectors and matrices. $\mathbf{R}$ is the set of real numbers, and $\mathbf{Z}$ is the set of integers.

$f(\boldsymbol{x})$ is called an objective function, and the MIQP has a linear equality constraint and an inequality constraint. If $\boldsymbol{x}$ satisfies all constraints, it is called a feasible point.

The integer entries in $\boldsymbol{x}$ are called integer variables. If every integer variable has either 0 or 1, the programming problem is classified as a binary programming (BP) problem.

B. Choice of QP Solver Algorithm

In general, to solve the MIQP problem, integer entries are temporally assumed to be certain values, and then QP child sub-problems are iteratively solved. Thus, it would take a long time to research all candidates of the integer entries. Various methods have been proposed to rapidly solve the QP problem, out of which a gradient projection method [11], an interior point method [12], and an active set method [4] are well-known methods.

- The gradient projection method can rapidly calculate a feasible point, but it is merely applicable to a problem whose constrains have only upper/lower bounds.
- When a QP problem has many inequality constraints, the interior point method is suitable.

- In solving of the iterative QP problems, a next QP problem is slightly changed from previous one. The active set method can reuse the previous set of equations (active set). Such iterative computation is called a “warm start” [4], [7], [10]. By making use of “warm start”, the active set method is superior to the interior point method.

So, we choose the active set method for our QP solver because the “warm start” is effective to solve an MIQP problem. Strictly, we adopt a dual active set method called Goldfarb-Idnani method. The dual active set method can rapidly calculate a first point by using its duality. According to [5], without using the duality, the computing time for the first point takes one third to one half of the total time. In order to reduce this large computing time, the dual active set method using the duality is effective.

For further information about the dual active set method, see Ref. [5].

C. Branch-and-Bound Method

Branch-and-bound method is a general algorithm for solving discrete optimizations, and it can be combined with the dual active set method. It solves relaxed problems iteratively, and finally gets the optimal solution of the primal problem. The main operations are branching and bounding.

We divide a problem into child sub-problems by branching, and solve the child sub-problems. If we can notice that the child sub-problem needs not to solve any more, we remove the following problems by bounding.

III. Implementation onto FPGA

In this section, we propose MIQP solver architecture. The platform is a Celoxica RC250. The target FPGA is an Altera Stratix II.

As reference software, we adopted “QuadProg++” for implementation. It is written in C++ by the author of [6]. We use Handel-C as a hardware description language. Handel-C is a C-like language, and it is beneficial when porting the C++ program.

A. Divider of Fixed Point

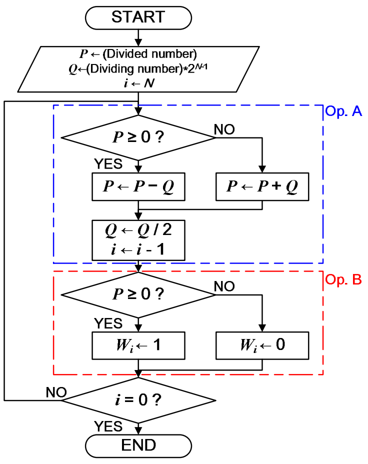


Fig. 2. Flowchart of non-restoring method.

To reduce the hardware cost, we implemented a fixed point divider with a non-restoring method. Fig. 2 shows the flowchart of the non-restoring method. W_i means the i-th bit of the quotient W. N is the bit width of the fixed point (= 36). Since Op.A and Op.B in the figure are sequentially computed, they can be parallelized with pipeline architecture. We will discuss this pipeline in Subsection C.

B. Reusing of the First Point

The first point can be reused for all child sub-problems. Namely, the first point is on the top of the hierarchy, and thus we don’t have to compute the first point two or more times. From this, we implement the MIQP solver, in which the first point is only computed once and then we reuse it. Consequently, we can reduce the cycle times.

C. Implementation of the Pipeline Architecture

To make the MIQP solver further faster, we parallelize the computation. However, it becomes difficult if it needs multiple-access memory, because only single-port SRAM is available on the FPGA without area overhead. To implement parallel processing, we propose pipeline architecture for our MIQP solver.

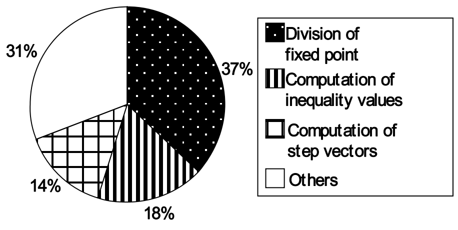


Fig. 3. The computing ratio of the solver.

Fig. 3 shows the ratio of the whole computation. This was obtained from the MIQP solver implemented on the FPGA in the case where it computed sequentially but reused the first point. The three of the largest portions are: the division of the fixed point, the computation of the inequality values, and the computation of the step vectors. They are suitable for pipeline architecture because of the loop computations; we parallelize them with pipelining.

As a representative, we explain the pipeline of the division. Because the division does not require memory access, a simple pipeline can be implemented. Fig. 4 illustrates the pipeline. Op.A and Op.B correspond to those of Fig. 2. By this pipeline implementation, the cycle times are reduced from $2N$ (= 72) to N+1 (= 37).

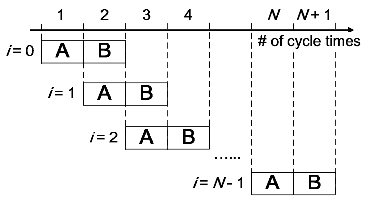


Fig. 4. Pipeline in division.

We also implemented the computation of the inequality values and the computation of the step vectors with a pipelined architecture. They generate dot product of vectors, and each of the cycle times is reduced from 4n (= 64) to n+3 (= 19) in theory (n is defined in Section II.A).

We show the overall results of the three kinds of pipelines. Fig. 5 shows the experimental results of the three parts. In Fig. 5 (i), 47.0% of the cycle times is reduced in the sample problem 2. This is the ideal value expected in the pipelined divider. In Figs.

5 (ii) and (iii), more than 75% of the cycle times are reduced because of the effectiveness of the pipelines and simple parallelized computation. In the sample problem 2, n is set to 16, and cycle times are reduced from 64 cycles to 19 cycles. Consequently, the reduction ratio is 70.3% by the pipelines in theory. The additional 6% (= 76% – 70%) is the contribution of the simple parallelized computation.

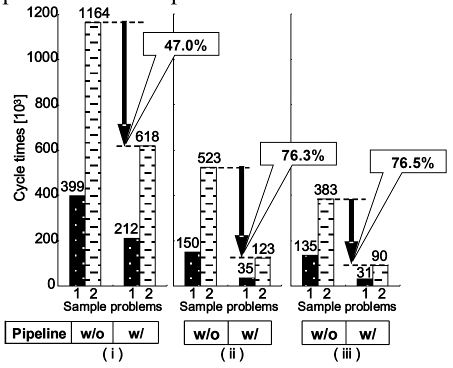


Fig. 5. Cycle times comparison between sequential and pipelined implementations: (i) division of fixed point, (ii) computation of inequality values, and (iii) computation of step vectors.

D. MIQP Solver Architecture with Multi-QP Solver Cores

The Pipelines explained in the previous subsection are implemented onto one QP solver core. We extend it to multi-QP solver cores; the multi cores solve different QP child sub-problems together. Fig. 6 shows the block diagram of our multi-core MIQP solver architecture. The volumes of SRAM1 and SRAM2 are decided to solve an MIQP problem whose variable $\boldsymbol{x}$ has 16 entries.

First, the input module receives the original problem from a host PC through the USB interface, and sends it to the first point calculator. Then, the first point calculator writes both the original problem and the first point data to SRAM1. SRAM1 will be written child sub-problems later that have inequality constraints.

K independent QP solver cores and branch-and-bound method module are controlled by the sequence control module. Each QP solver core is given a different QP problem through the sequence control module, and solves it asynchronously. The solutions are sent to branch-and-bound method module; then it updates the optimal solution and the optimal values in SRAM2. At the same time, child sub-problems to be solved later are generated and saved in SRAM1.

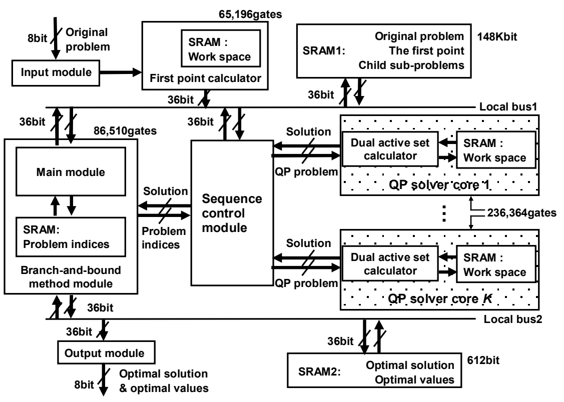


Fig. 6. Block diagram of implemented MIQP solver architecture.

After all child sub-problems are exhausted and the MIQP problem was solved, the output module sends the optimal data through the USB interface.

In addition, we make data structure of the child sub-problems in SRAM1 simpler. Furthermore, the cycle times of the branch-and-bound method is less than 1% of the total cycle times; the QP solver cores need not to wait the calculation of the branch-and-bound method module.

The generated child sub-problems are saved at a queue prepared in SRAM1. The MIQP solver assigns the child sub-problems to QP solver cores. The MIQP solver decides the assignment when the QP solver cores become idle. Fig. 7 explains the sequence with two QP solver cores. The first problem 0 is solved by core1 as previously explained. Then, the child sub-problems 1 and 2 are generated. They are assigned to the cores 1 and 2 respectively so that each QP solver core doesn't become idle. Next, if the problem 2 was solved faster than the problem 1, the problem 3 is assigned to core 2 because the core 1 has not solved the problem 1. Like this, we parallelize the QP solver cores efficiently, and can reduce the idle time of them.

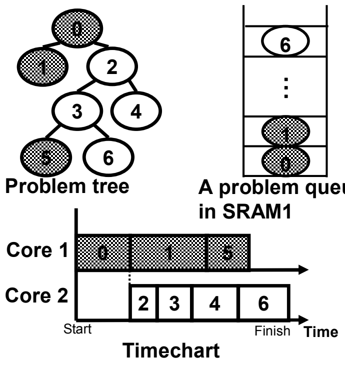


Fig. 7. Assignment of child sub-problems.

Fig. 8 discusses the efficiency of the multi-QP solver cores. Here, cycle times are normalized relative to solving on a single core. The normalized cycle times are decreased with the number of the QP solver cores. As described, the ratio of the branch-and-bound method accounts is less than 1%, this delay is not considered in this estimation.

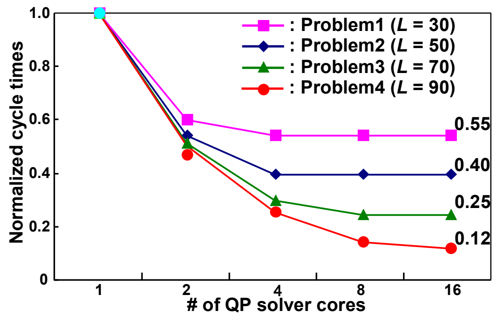


Fig. 8. Normalized cycle times when the number of multi-QP solver cores are changed. The parameter is *n*.

In Fig. 8, *L* is the number of entries in the variable $\boldsymbol{x}$. We must solve more child sub-problems as *L* becomes larger. This is the reason why the problem 4 (*L* = 90) is the most effective to the multi-QP solver cores. In large-scale problems, many child sub-problems can be assigned to many QP solver cores.

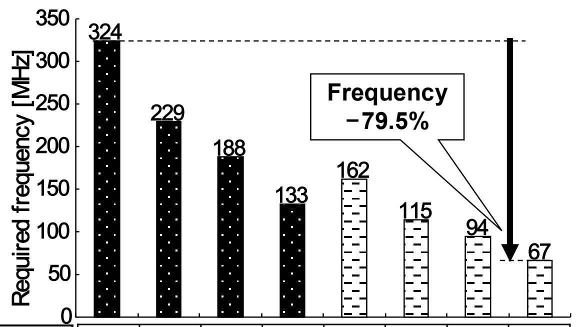
IV. Implementation Results

In this section, we describe overall results of the MIQP solver in terms of speed and gate number.

Fig. 9 shows the required frequencies on the FPGA when a problem has 16 entries of variable ***x***; the required frequency is what the FPGA needs to finish calculation in the same time as a PC (Intel core 2 duo 3.16GHz and 3 GB memory). "Reusing" means the re-usage of the first point, and "Pipeline" means the pipelined architecture described in Section III.C. The number of QP solver cores is either of one or two.

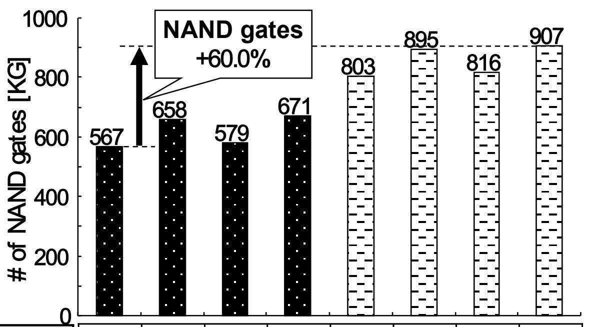
The 2-cores implementations are indeed twice higher performance than the 1-core implementation. The 2-core implementation with "Reusing" and "Pipeline" achieves 67 MHz operation which means 79.5% reduction of the required frequency, compared with the case that these proposed methods are not implemented. In order to further reduce the required frequency, more QP solver cores are needed.

Fig. 10 shows the number of NAND gates on the same conditions as Fig. 9. The largest overhead is derived from the 2-core implementation; one QP solver core occupies 236,364 NAND gates.



Reusing	w/o	w/	w/o	w/	w/o	w/	w/o	w/
Pipeline	w/o	w/o	w/	w/	w/o	w/o	w/	w/
# of cores	1	1	1	1	2	2	2	2

Fig. 9. Frequency comparison.



Reusing	w/o	w/	w/o	w/	w/o	w/	w/o	w/
Pipeline	w/o	w/o	w/	w/	w/o	w/o	w/	w/
# of cores	1	1	1	1	2	2	2	2

Fig. 10. # of NAND gates comparison.

The area-overhead from "Reusing" comes from copying the first point to another module. The area-overhead by "Pipeline" is the extra registers array. They are much smaller than a multi QP solver core; so "Reusing" and "Pipeline" are cost-effective. In the case that we implement both the proposed methods on the 2-core QP solver core, the area-overhead is 60.0%. However, note that the required frequency is about one-fifth (speed is about five times) in this case.

According to Quartus II version 7.2, the power consumption of the MIQP solver is 4.20 W. This low-power feature will give on-line complex planning and movement to a mobile robot with a small-capacitance battery.

V. Conclusion

We proposed: reusing the first point, pipelined architecture, and multi-core architecture for an MIQP solver on an FPGA. As a result, we achieved 79.5% reduction of the cycle times, compared to a straight-forward implementation with sequential processing. The MIQP solver operates at 67 MHz and 4.2 W although a PC requires 3.16 GHz and much power. We observed that the multi QP solver cores are effective in a large-scale problem. If we use larger-scale problems and implement more QP solver cores, the efficiency for reduction of cycle times will become better.

References

[1] P.Costamagna, L.Magistri, A.F.Massardo, "Design and part-load performance of a hybrid system based on a solid oxide fuel cell reactor and a micro gas turbine" Journal of Power Sources 96 (2001) 352-368.

[2] Masakazu Mukai, Takehito Azuma, Masayuki Fujita, "A Collision Avoidance Control for Multi-Vehicle Using PWA/MLD Hybrid System Representation" IEEE International Conference on Control Applications 2004.

[3] Yingjie Yin, Shigeyuki Hosoe, Zhiwei Luo, "A mixed logic dynamical modeling formulation and optimal control of intelligent robots" Optim Eng (2007) 8: 321–340.

[4] Daniel Axehill, "Applications of Integer Quadratic Programming in Control and Communication," Linköping Studies in Science and Technology, thesis no. 1218, pp. 9-45, 2005.

[5] D.Goldfarb, A.Idnani, "A numerically stable dual method by solving strictly convex quadratic programs," Mathematical Programming, Prog. 27, pp. 1-33, 1983.

[6] Luca Di Gaspero, "Luca Di Gaspero's home site", http://www.diegm.uniud.it/digaspero/, (2009/02/20access).

[7] Dimitar Dimitrov, Hans Joachim Ferreau, Pierre-Brice Wieber, Moritz Diehl, "On the Implementation of Model Predictive Control for On-line Walking Pattern Generation" IEEE International Conference on Robotics and Automation, 2008.

[8] Dimitar Dimitrov, Pierre-Brice Wieber, Olivier Stasse, Hans Joachim Ferreau, Holger Diedam, "An Optimized Linear Model Predictive Control Solver for Online Walking Motion Generation" IEEE International Conference on Robotics and Automation, 2009.

[9] Sven Leyffer, "Deterministic Methods for Mixed Integer Nonlinear Programming", PhD thesis, Department of Mathematics & Computer Science University of Dundee, 1993.

[10] Daniel Axehill, Anders Hansson "A Mixed Integer Dual Quadratic Programming Algorithm Tailored for MPC", Proceedings of the 45th IEEE Conference on Decision & Control Manchester Grand Hyatt Hotel San Diego, CA, USA, December 13-15, 2006.

[11] J. B. Rosen, "THE GRADIATION PROJECTION METHOD FOR NONLINEAR PROGRAMMING. PART I. LINEAR CONSTRAINTS", Society for Industrial and Applied Mathematics, Vol. 8, No.1, March, 1960.

[12] S. J. Wright, "Interior-point methods for optimal control of discrete-time systems", Journal of Optimization Theory and Applications, 77:161 – 187, 1993.