Parallel-Processing VLSI Architecture for Mixed Integer Linear Programming

Hiroki Noguchi, Junichi Tani, Yusuke Shimai, Hiroshi Kawaguchi, and Masahiko Yoshimoto
Kobe University, Kobe, 657-8501 Japan
Phone: +81-78-803-6234, E-mail: h-nog@cs28.cs.kobe-u.ac.jp

***Abstract*—This paper describes parallel processor architecture for a mixed integer linear programming (MILP) solver to realize motion planning and hybrid system control in robot applications. It features pipeline architecture with an MILP-specific configuration and two-port SRAM. Based on the architecture, both FPGA and VLSI implementations have been done to solve sample problems including 16 variables. The FPGA implementation can reduce the power consumption to 13 W: an 85.4% reduction compared to a 3.0-GHz processor (Pentium 4; Intel Corp.). The VLSI solver further reduces the power to 6.4 W using 0.18-µm CMOS technology.**

***Index Terms*—mixed integer linear programming problem, hardware, low power**

I. Introduction

In recent years, mobile robot technology has developed remarkably. The progress of related technologies enables robots to work in dangerous places instead of people and supports comfortable human life. For instance, robots are expected to be adapted for use as nursing-care robots in societies with fewer children and more elderly people. Several studies of robots have examined path optimization, motion control, and hybrid systems [1–4]. The technologies can be formulated as an integer programming (IP) or a mixed integer programming (MIP) [5–6]. For this study, we investigate mixed integer linear programming (MILP) in MIP.

In fact, MILP is an optimization technique using maximizing or minimizing a linear function. A formulation for a maximization problem (or a minimization problem) is presented in (1), where x is a vector with variables, A is a matrix, and b and c are vectors of coefficients. In MILP, some variables are integers.

$$\begin{aligned} &\text{max or min} && z = \mathrm{cx} \\ &\text{subject to} && A_1\mathrm{x} \le \mathrm{b}_1,\ A_2\mathrm{x} = \mathrm{b}_2,\ A_3\mathrm{x} \ge \mathrm{b}_3 \\ & && \mathrm{x} \ge 0 \end{aligned} \tag{1}$$

In fact, MILP is classified as an NP-hard problem because the computing time to solve the problem increases exponentially when the number of variables increases. The path optimization problem and motion planning for robots and most other applications are needed to solve MILP in real time. Furthermore, power consumption is an important issue. It is necessary to reduce the power consumption of the calculation of the path planning, motion planning, and so on, so that the robot might function for a longer time. Implementing a solver not in a PC, but in an FPGA or VLSI is effective in terms of speed and power efficiency.

II. MILP Algorithm

Figure 1 depicts a flowchart of an MILP solver using the branch-and-bound method and the simplex method. The sub-problem is solved using the simplex method (two-phased method) and is generated recursively using the branch-and-bound method [7].

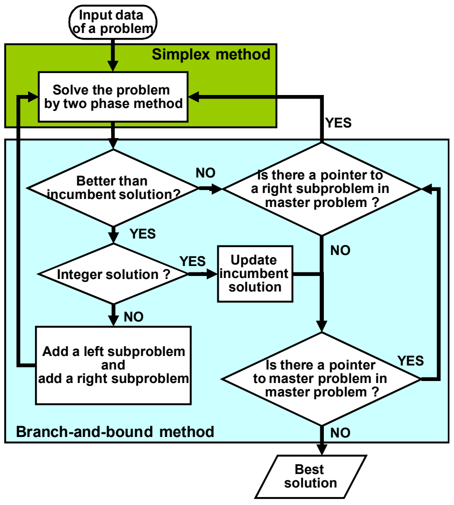

Input data of a problem
Simplex method
Solve the problem by two phase method
YES
Better than incumbent solution?
NO
Is there a pointer to a right subproblem in master problem ?
YES
Integer solution ?
YES
Update incumbent solution
NO
NO
Add a left subproblem and add a right subproblem
Is there a pointer to master problem in master problem ?
YES
Branch-and-bound method
NO
Best solution

Fig. 1. Flowchart of MILP.

A. *Branch-and-Bound Method Algorithm*

The branch-and-bound method, which uses a bounding operation and branching operation to solve MILP, is used to derive the optimized solution. First, the simplex method calculates the original problem using the simplex algorithm (the two-phased method) [8–9]. Then, according to the results from the simplex module, the branch-and-bound method operates the branching operation as one of the non-integer variables that should be an integer and generates a new LP relaxation problem. An LP relaxation problem has the same objective function and set of constraints as the original problem; its integer variables are replaced by continuous constraints. The simplex method recalculates the LP relaxation problem until all variables become integers. The branch-and-bound method also operates bounding operations to reduce unnecessary calculations.

We use a depth-first rule as the mode of a sub-problem selection in branch-and-bound method. The depth-first rule spends fewer hardware resources, especially memory, than a breadth-first rule. The way to select a non-integer variable, which should be an integer, is to judge whether the variable is an integer in turn from the variables with the smallest suffix.

B. *Simplex Method Algorithm*

We use a two-phased method as the simplex algorithm. The simplex method requires a feasible solution as a starting point. At the first phase, we obtain the first feasible solution or information that the LP relaxation problem has no feasible solution. At the second phase, it advances from this starting point to the optimum solution. Alternatively, the information that the solution is unbounded is obtained.

C. *Analysis of Computation Amount Using Software Solver*

To clarify the computation amount in the algorithm, we analyzed it using a software solver with a PC (3.0-GHz Pentium 4; Intel Corp.). Figure 2 presents the computing time at each processing time when solving sample programs including 16 variables using the software solver.

Because the simplex algorithm includes iterated operations, it occupies 98% of all computing time. Therefore, the reduction of the number of cycles of each simplex calculation is effective in hastening the solver. Most cycle times in each simplex calculation are occupied by the pivot function (40%) and the tableau function (51%).

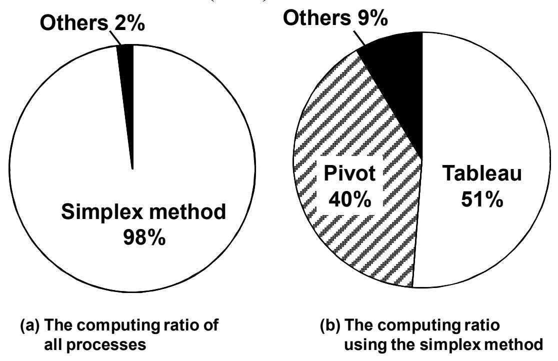


Fig. 2. Computation time of the solver implemented with software.

III. Hardware Architecture

A. *Architecture Overview*

Figure 3 portrays a block diagram of the proposed architecture. We divide the solver into four modules: input, output, simplex method, and branch-and-bound method module. Problem data are stored in the embedded RAM. Regarding implementation in the FPGA, we did not use multi-port RAM but single-port RAM instead. Therefore, we must bear in mind that more than two accesses to RAM within one clock cycle are not allowed.

B. *Applying Fixed-Point Variables*

We use fixed-point processing for decimal calculations. The fixed-point architecture uses fewer hardware resources than the floating-point architecture does. Furthermore, the calculation speed is higher in fixed-point mode. The word length of each variable was set to 32 bits, including 16-bit integer words and 16-bit fractional words because a solver that has 15 or fewer bits per fractional word cannot solve an LP relaxation problem. When decimal accuracy is more necessary, it is possible to achieve it by raising the number of fractional bits.

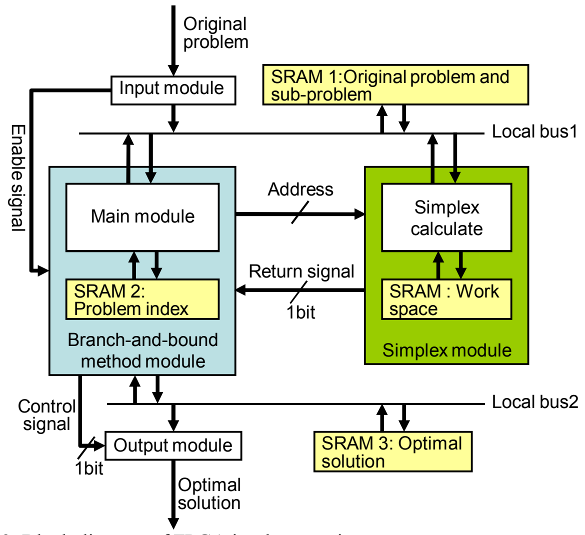


Fig. 3. Block diagram of FPGA implementation.

C. *Parallel Computing*

To reduce the computation time of the pivot function and the tableau function, we implement pipeline processing in these two functions [10]. The pipeline architecture enables us to hide the process latency and achieve high throughput because the pipelined implementation allows several processes to operate simultaneously at different stages within the hardware. The tableau module with the tableau function has nine calculation steps:

Op. A: reading the fixed-point value and integer value from SRAM.

Op. B: type conversion from the integer value to the fixed-point value.

Op. C: multiplication by the two fractional values.

Op. D: removal of the fractional value from the fixed-point value.

Op. E: rounding off of the fixed-point value.

Op. F: addition of the two fixed-point values.

Op. G: removal of the fractional value from the fixed-point value.

Op. H: rounding off of the fixed-point value.

Op. I: renewal of the temporary fixed-point result.

Data dependency exists between Op. F and Op. I. It is necessary to consider data dependency in implementing a pipeline architecture. Figure 4 presents a pipeline process we implemented without two-port SRAM in the tableau module.

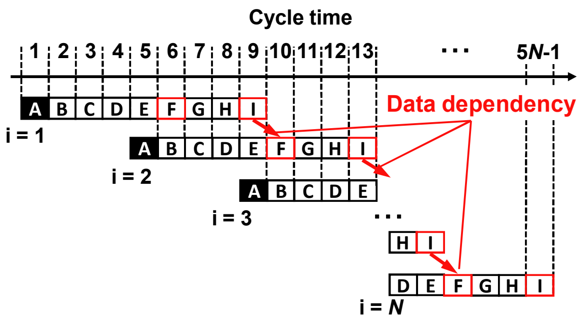
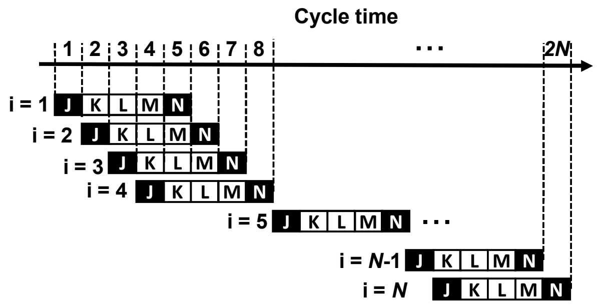


Fig. 4. Pipelined process of a tableau function on FPGA without two-port SRAM.

The pivot module with the pivot function executes two loop calculations:

Op. J: reading the fixed-point value and integer value from SRAM.
Op. K: division of the fixed-point value by the other fixed-point value.
Op. L: removal of the fractional value.
Op. M: rounding off of the fractional value.
Op. N: writing the fixed-point value to SRAM. Here is the first loop.
Op. O: reading the fixed-point value from SRAM.
Op .P: multiplication by the two fractional values.
Op. Q: removal of the fractional value.
Op. R: rounding off of the fractional value.
Op. S: reading the fixed-point value and integer value from SRAM.
Op. T: subtraction of the fixed-point values.
Op. U: removal of the fractional value.
Op. V: rounding off of the fractional value.
Op. W: writing the fixed-point value to SRAM.

In implementing the FPGA solver, the pipeline architecture is constructed considering SRAM access. Figure 5 depicts the pipeline process we implemented without two-port SRAM in the pivot module.



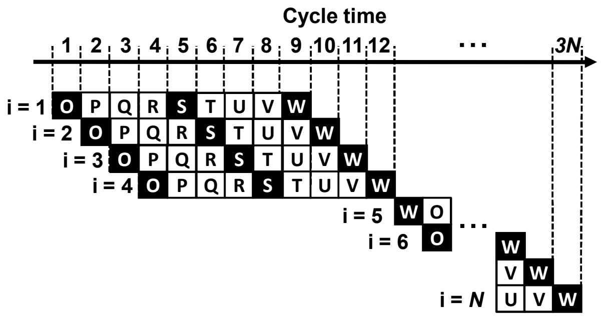


Fig. 5 Pipelined process of pivot function on FPGA without two-port SRAM.

IV. Implementations and Results

A. FPGA

We implemented the MILP solver on an FPGA board (RC250; Celoxica Ltd.) to evaluate the proposed architecture before designing the VLSI solver. Figure 6 portrays a picture of the FPGA board. The FPGA chip is a Stratix-II (Altera Corp.).

The parallel architecture implemented in FPGA increases hardware resources but it can reduce the operating frequency compared with PC. Figure 7 presents the required frequency on the condition that the FPGA terminates processes at the same computation time as the PC. We conclude that the proposed parallel computing reduces the required frequency by 57.8% (202.5 MHz) on average compared to the sequential computing (480.0 MHz). In comparison with a 3.0 GHz processor (Pentium 4; Intel Corp.), we can reduce required frequency by 93.3% using parallel processing.

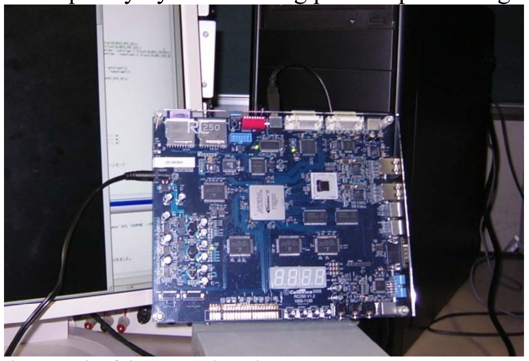


Fig. 6. Photograph of the FPGA board.

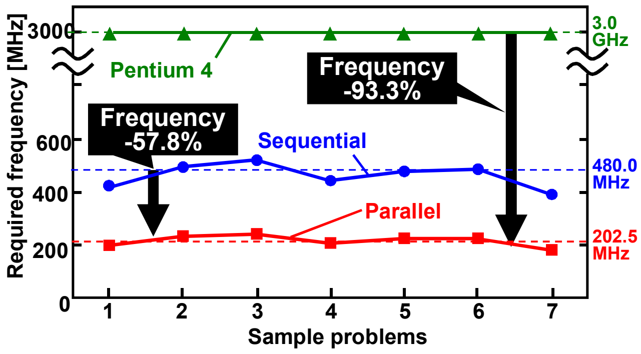


Fig. 7. Required frequencies: PC, FPGA sequential processing, and FPGA parallel processing.

B. VLSI Implementation

By implementing the FPGA solver, we clarify that the composition which implements pipeline architecture into the pivot module and tableau module considering the increase of hardware resource but the reduction of cycle number improves efficiency. Therefore, we designed the VLSI solver using CMOS 180-nm process technology for receiving the advantage of implementing hardware. Figure 8 portrays the designed VLSI solver. The core size is 3.5 × 3.5 mm^2. The VLSI integrates all modules and SRAMs shown in Fig. 3.

The VLSI implementation enables lower power consumption than the FPGA solver. The VLSI solver can mount two-port SRAMs, which improves the pipeline architecture's efficiency. Figure 9 portrays the pipeline process using a two-port SRAM in a pivot module. Consequently, the VLSI solver can reduce the cycle number. Figure 10 shows that the VLSI solver can further lower the frequency by introducing the two-port RAM. Results show that the required frequency can be lowered to as little as 16.8% of that of a sequential processing FPGA. When the VLSI solver operates at 100 MHz: it achieves a 1.24-times faster computation time than that of a Pentium 4 processor.

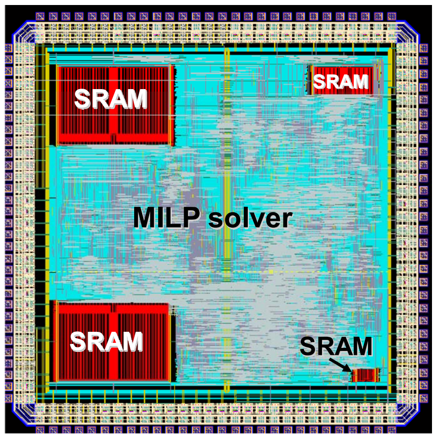


Fig. 8. VLSI MILP solver. The nominal supply voltage is 1.8 V.

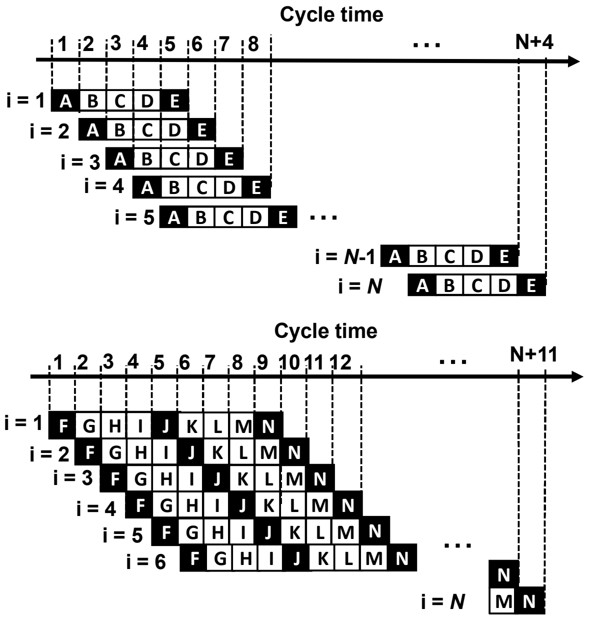
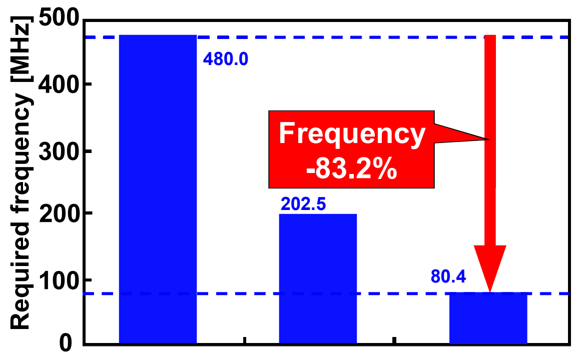


Fig. 9. Pipelined process of pivot function with two-port SRAM.



Pivot	FPGA (Seq)	FPGA (Par)	VLSI
Tableau	FPGA (Seq)	FPGA (Par)	VLSI

Fig. 10. Required frequencies: FPGA sequential processing, FPGA parallel processing and VLSI.

C. *Power Comparisons*

The FPGA solver and the VLSI solver can achieve low power consumption by the effect of the frequency reduction.

Figure 11 presents a power comparison among the following four cases: a PC with a Pentium 4 processor, an FPGA with sequential processing, an FPGA with parallel processing, and VLSI. Although the hardware is larger with parallel processing, average power consumption in the parallel processing is smaller because of the reduction of the required frequency. As a result, the total power consumption can be reduced by 85.4% compared to that of a PC with a Pentium 4 processor. Furthermore, the VLSI solver reduced the power consumption to 50.8% in comparison to the FPGA parallel solver.

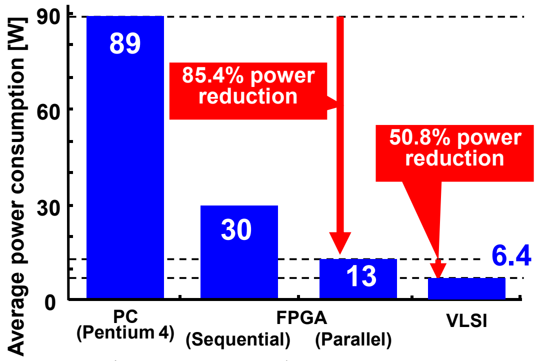


Fig. 11. Power comparison: PC, FPGA, and VLSI.

V. Summary and Discussion

This paper presented the FPGA and VLSI implementation of a solver using the simplex algorithm and the branch-and-bound method. In the VLSI solver, the required frequency for 16-variable MILP is 80.4 MHz, which achieves a frequency reduction of 83.2% compared to results obtained using sequential computing in FPGA. The power of the VLSI is 6.4 W, which is almost half that of the FPGA.

In this study, only one simplex module was mounted, but parallelism can be raised by increasing the module number. We will continue to improve the hardware solver to realize a robot that can operate as a hybrid system or which can plan an optimum path to support the comfortable life of people everywhere.

References

[1] E. Masehian, G. Habibi, "Motion Planning and Control of Mobile Robot Using Linear Matrix Inequalities (LMIs)," *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4277–4282, Feb. 2007.

[2] M. G. Earl, R. D'Andrea "Iterative MILP Methods for Vehicle Control Problems," 43rd IEEE Conference on Decision and Control, December 14–17, 2004.

[3] C. S. Ma, R. H. Miller, "MILP Optimal Path Planning for Real-Time Applications," Proceedings of the 2006 American Control Conference, June 14–16, 2006.

[4] J. Lygeros, D. N. Godbole, S. Sastry "Verified hybrid controllers for automated vehicles," *IEEE Transactions Automation Contribution*, vol. 43, no. 4, pp. 509–521, 1998.

[5] Wolsey, L. "Integer Programming," John Wiley & Sons, 1998.

[6] A. Schrijver, "Theory of Linear and Integer Programming," *Wiley and Sons*, 1972.

[7] M. Sakawa, "Optimization of discrete systems," 2000.

[8] M. Sakawa, "Optimization of linear systems," 1984.

[9] S. S. Morgan, "A comparison of simplex method algorithms," Master's thesis, University of Florida, 1997.

[10] C. V. Ramamoorthy, H. F. Li, "Pipeline Architecture," *ACM Computing Surveys (CSUR)*, vol. 9, no. 1, pp. 61–102, March 1977.