IEICE TRANS. INF. & SYST., VOL.E87–D, NO.8 AUGUST 2004

PAPER Special Section on Reconfigurable Systems

The Design and Evaluation of Data-Dependent Hardware for Subgraph Isomorphism Problem*

Shoji YAMAMOTO†, **Shuichi ICHIKAWA**†a), *Members*, *and* **Hiroshi YAMAMOTO**†, *Nonmember*

SUMMARY Subgraph isomorphism problems have various important applications, while generally being NP-complete. Though Ullmann and Konishi proposed the custom circuit designs to accelerate subgraph isomorphism problem, they require many hardware resources for large problems. This study describes the design of data-dependent circuits for subgraph isomorphism problem with evaluation results on an actual FPGA platform. Data-dependent circuits are logic circuits specialized in specific input data. Such circuits are smaller and faster than the original circuit, although it is not reusable and involves circuit generation for each input. In the present study, the circuits were implemented on Xilinx XC2V3000 FPGA, and they successfully operated at a clock frequency 25 MHz. In the case of graphs with 16 vertices, the average execution time is about 7.0% of the software executed on an up-to-date microprocessor (Athlon XP 2600+ of 2.1 GHz clock). Even if the circuit generation time is included, data-dependent circuits are about 14.4 times faster than the software (for random graphs with 16 vertices). This performance advantage becomes larger for larger graphs. Two algorithms (Ullmann's and Konishi's) were examined, and the data-dependent approach was found to be equally effective for both algorithms. We also examined two types of input graph sets, and found that the data-dependent approach shows advantage in both cases.

key words: *FPGA, custom circuit, graph algorithm, NP-complete, partial evaluation*

1. Introduction

A subgraph isomorphism problem is a simple decision problem. Given two graphs, G_α and G_β, it is determined whether G_α is isomorphic to any subgraph of G_β. Figure 1 illustrates an example of this problem. In this figure, G_β has a subgraph that is isomorphic to G_α, whereas G_γ does not. The subgraph isomorphism problem has many applications; e.g., scene analysis in computer vision and search operation in chemical structural formula databases. However, subgraph isomorphism problems are generally NP-complete and computationally difficult to solve [7].

It is well known that the custom circuit for a specific application can accelerate the execution of that application. The custom circuit for subgraph isomorphism problems was first suggested by Ullmann in 1976 [13]. In his paper, Ullmann proposed an algorithm (*Ullmann's algorithm*) that accelerate the solutions to subgraph isomorphism problems by a smart pruning procedure (*refinement procedure*). He pointed out that his refinement procedure can be implemented by parallel hardware. However, Ichikawa et al. [9] later evaluated Ullmann's design on FPGA, and reported that Ullmann's circuit requires too many resources for practical use.

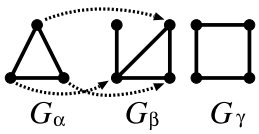

Fig. 1 Subgraph isomorphism.

Ichikawa, Udorn, and Konishi [10] proposed another algorithm (*Konishi's algorithm*), which adopts a pruning procedure that is simpler than the refinement procedure. Konishi's algorithm can be implemented in smaller logic scale than Ullmann's. This means that Konishi's algorithm can handle larger problems than Ullmann's algorithm in the same logic scale. When the problem size is fixed, two or more Konishi's units can be implemented in the area of one Ullmann's unit, to solve problems in parallel. Konishi's circuit is known to be more cost-effective than Ullmann's in some circumstances [9].

Ichikawa and Yamamoto [11] proposed to reduce the logic scale of Ullmann's circuit by generating a data-dependent circuit, which is a logic circuit specialized for its input data. They reported that the logic scale can be reduced to 1/20 of the original in the best case. Though the data-dependent circuit must be generated for each data, the total execution time was expected to be shorter than that of software in large problems, even including the circuit generation time [11].

The previous work [11], however, still had the following problems.

- The results were based on logic synthesis and technology mapping. Placement, routing, and implementation were not accomplished. It is not verified yet that this approach is actually effective on an FPGA platform.
- Only one algorithm (Ullmann's algorithm) was examined. It is not evident that the data-dependent approach is also advantageous for other algorithms.
- Trees were assumed as input graphs. It is not evident that the data-dependent approach is advantageous for general graphs.

Thus, this study aims to examine the following items.

- To measure the logic scale, circuit generation time, and

Manuscript received December 8, 2003.
Manuscript revised March 5, 2004.

†The authors are affiliated with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441–8580 Japan.

*This work partially appeared as an extended abstract in the Proceedings of FPL2003, pp.1024–1027, LNCS 2778, Springer (2003).

a) E-mail: ichikawa@tutkie.tut.ac.jp

Table 1 Related works.

References	Note
Chakradhar, Agrawal [4]	VLSI for independent set problem
Babb, Frank, Agarwal [1]	Dynamic computation structure for closed semiring problems
Dandalis, Mei, Prasanna [6]	Domain specific mapping for graph problems
Huelsbergen [8]	Dynamic graph processor
Bingham, Serra [2]	Data dependent circuit for Hamiltonian cycle problem
Mencer, Huang, Huelsbergen [12]	HAGAR: Multi-context graph processor
Ichikawa, Yamamoto [11]	Data dependent circuit for subgraph isomorphism problems

execution time of the data-dependent circuits for subgraph isomorphism problems by implementing them on an actual FPGA platform.
- To implement and evaluate the data-dependent circuits of another (Konishi's) algorithm.
- To examine the data-dependent circuits for general input graphs.

There have been few studies on the data-dependent circuits for graph problems, which are summarized in Table 1. However, subgraph isomorphism problems were not examined except in the authors' previous study [11] and the present study.

The rest of this paper is constructed as follows. Section 2 introduces Ullmann's algorithm and Konishi's algorithm along with their custom circuit designs. Section 3 outlines the design issues of data-dependent circuits. Section 4 presents some evaluation results on the Xilinx XC2V FPGA platform. The last section, Section 5, briefly summarizes our results.

2. Algorithms for Subgraph Isomorphism Problems

This section briefly describes the algorithms and custom circuit designs for subgraph isomorphism problems. More details will be found in the preceding studies [9], [10], [13].

2.1 Subgraph Isomorphism Problem

Assume that a graph G is defined by (V, E), in which V is the set of vertices and E is the set of edges. A graph $G_\alpha = (V_\alpha, E_\alpha)$ is the *subgraph* of another graph $G_\beta = (V_\beta, E_\beta)$, if both $V_\alpha \subseteq V_\beta$ and $E_\alpha \subseteq E_\beta$ hold. G_α is *isomorphic* to G_β, if and only if there is a 1:1 correspondence between V_α and V_β that preserves adjacency. The subgraph isomorphism problem is a decision problem to determine whether G_α is isomorphic to a subgraph of G_β. An example of this is shown in Fig. 1.

Let p_α and q_α be the number of vertices and edges of a graph G_α, respectively[†]. In the same manner, p_β and q_β stand for the number of vertices and edges of graph G_β. If $p_\alpha > p_\beta$ holds, it is evident that G_α is not isomorphic to any subgraph of G_β. Therefore, we assume $p_\alpha \leq p_\beta$ in the following discussion.

2.2 Enumeration Algorithm

As is easily seen, subgraph isomorphism can be determined by brute-force enumeration with a depth-first tree-search algorithm. Assume that $V_\alpha = \{u_1, u_2, u_3\}$ and $V_\beta = \{v_1, v_2, v_3, v_4\}$. At the i-th stage of the search tree, u_i is mapped to a possible vertex in V_β (Fig. 2).

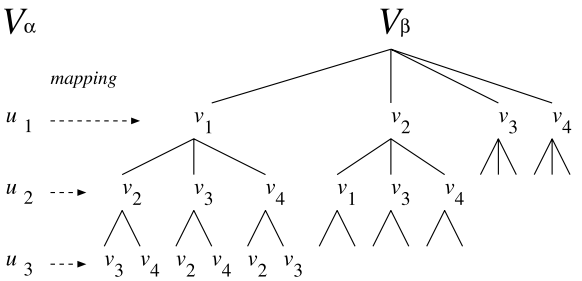

Fig. 2 Search tree.

At each leaf, an adjacency condition is checked by examining the correspondence of edges from E_α to E_β. If all adjacency relations are preserved at a leaf, a subgraph isomorphism is found. This check corresponds to determining the following condition, where the function f is the mapping function from V_α to V_β.

$$\forall\{u_i, u_j\} \in E_\alpha \Rightarrow \{f(u_i), f(u_j)\} \in E_\beta \quad (1)$$

This naive tree-search algorithm involves very long execution time because of its vast search space. The number of leaves is ${}_{p_\beta}P_{p_\alpha} = p_\beta!/(p_\beta - p_\alpha)!$, which increases quickly as p_α and p_β grow. Some procedure is required to prune futile sub-trees, thus shortening execution time.

2.3 Ullmann's Algorithm

Ullmann's algorithm [13] is very popular for subgraph isomorphism problems. Ullmann's algorithm is a smart tree-search algorithm with a *refinement procedure* for pruning. For G_α to be isomorphic to a subgraph of G_β, adjacent vertices in G_α must be mapped to adjacent vertices in G_β. If this condition is not satisfied, there is no chance of finding a subgraph isomorphism and the subtree under this node can be pruned. The essence of the refinement procedure is to check this requirement recursively.

In Ullmann's algorithm, the refinement procedure is invoked at every node (including internal nodes). This involves some overhead at each internal node, but the performance gain is generally dramatic, since the uncontrolled

[†]We use these notations to follow in Ullmann's paper [13].

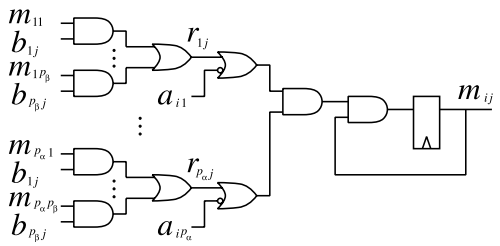

Fig. 3 Element circuit for refinement procedure.

expansion of the search tree is effectively inhibited. A formal description of this refinement procedure is found in Ullmann's paper [13].

The requirement of refinement procedure is represented by the following condition [13].

$$\underset{1\le x\le p_\alpha}{\forall x}\left((a_{ix}=1)\Rightarrow \underset{1\le y\le p_\beta}{\exists y}\ (m_{xy}\cdot b_{yj}=1)\right) \tag{2}$$

Here, $M=[m_{ij}]\,(1\le i\le p_\alpha, 1\le j\le p_\beta)$ is a matrix of temporary variables. The value of m_{ij} is 1 if it is possible to map $u_i\in V_\alpha$ to $v_j\in V_\beta$; otherwise $m_{ij}=0$. $A=[a_{ij}]\,(1\le i,j\le p_\alpha)$ and $B=[b_{ij}]\,(1\le i,j\le p_\beta)$ are the adjacency matrices of graph G_α and G_β, respectively. The adjacency matrix is a data structure to represent a graph. For example, the element $a_{ij}\in A$ is set to 1 if the edge $\{u_i,u_j\}\in E_\alpha$ exists; otherwise $a_{ij}=0$.

This condition can be implemented by the following logic equations. The refinement procedure repeatedly calculates the following values at each node of the search tree until M converges.

$$r_{xj} = \exists y\,(m_{xy}\cdot b_{yj}) \tag{3}$$

$$m_{ij} = m_{ij}\cdot(\forall x\,(\overline{a_{ix}}\vee r_{xj})) \tag{4}$$

Ullmann [13] pointed out that the above logic functions can be implemented by parallel hardware. His circuit for the refinement procedure consists of a $p_\alpha\times p_\beta$ array of the element circuit, which is illustrated in Fig. 3. It is readily seen that this circuit corresponds to Eq. (3) and (4).

Ullmann's circuit requires $O(p_\alpha {p_\beta}^2)$ hardware resources which rapidly increase for larger p_α and p_β. Our previous study revealed that only small graphs can be handled with state-of-the-art FPGA [10]. Hence, an alternate way of pruning is required for practical implementation.

2.4 Konishi's Algorithm

The problem with a refinement procedure is that it checks not only mapped but also as yet unmapped vertices, thereby entailing enormous resources. Ichikawa, Udorn, and Konishi [10] proposed a simplified pruning procedure which only handles mapped vertices. For G_α to be isomorphic to a subgraph of G_β, it is necessary that any subgraph of G_α be isomorphic to a subgraph of G_β. Konishi's simplified pruning procedure examines this necessary condition.

Let us see Fig. 2 again. At the i-th level of the search tree, only vertices $u_1,\ldots,u_i\ (1\le i\le p_\alpha)$ are mapped. In Konishi's algorithm, we only check the adjacency among these i vertices at the i-th level. To be exact, only the adjacency related to u_i is examined at the i-th level. As the vertices are mapped incrementally in the search tree, we only have to check the vertex mapped at the i-th level. More details of this algorithm are found in our previous report, in which the adjacency check was implemented by sequential hardware with the edge list stored in RAM [10]. This design was chosen to shrink the logic scale, although it is serialized and slow. Another problem of this design is that it is not suited for data-dependent approach as that is. Data-dependent approach has hardly any effects to sequential circuit, because simple constant propagation only works in combinatorial circuits†. Thus, we designed a brand-new implementation of Konishi's algorithm in the present investigation.

2.5 A New Design of Konishi's Algorithm

The adjacency check circuit of Konishi's algorithm can be implemented by combinatorial logic gates. This makes the logic scale larger, while making the execution faster. The authors examined two designs [15], and adopted the following one.

The new adjacency checker calculates the following necessary condition at each level of a search tree. This necessary condition is very similar to Ullmann's condition, except that $d_{ij}\ (1\le i,j\le p_\alpha)$ is introduced instead of a_{ij}.

$$\underset{1\le x\le p_\alpha}{\forall x}\left((d_{ix}=1)\Rightarrow \underset{1\le y\le p_\beta}{\exists y}\ (m_{xy}\cdot b_{yj}=1)\right) \tag{5}$$

Here, $d_{ij}=a_{ij}$ if $j<i$ holds; otherwise $d_{ij}=0$. This is the mechanism that limits the adjacency check to the vertices that are already mapped.

The above condition can be implemented by the following equations with temporary variables $X=[x_i]$, $Y=[y_j]$, $W=[w_j]$, and $Z=[z_j]\,(1\le i\le p_\alpha, 1\le j\le p_\beta)$.

$$x_i = (d_{i1}\cdot v_1)\vee\ldots\vee(d_{ip_\alpha}\cdot v_{p_\alpha}) \tag{6}$$

$$y_j = (x_1\cdot m_{1j})\vee\ldots\vee(x_{p_\alpha}\cdot m_{p_\alpha j}) \tag{7}$$

$$w_j = (v_1\cdot m_{1j})\vee\ldots\vee(v_{p_\alpha}\cdot m_{p_\alpha j}) \tag{8}$$

$$z_j = (b_{j1}\cdot w_1)\vee\ldots\vee(b_{jp_\beta}\cdot w_{p_\beta}) \tag{9}$$

$$OK = (\overline{y_1}\vee z_1)\cdot\ldots\cdot(\overline{y_{p_\beta}}\vee z_{p_\beta}) \tag{10}$$

In the above equations, $V=[v_i]\,(1\le i\le p_\alpha)$ represents the level of search tree, where $v_i=1$ if $i=k$ and $v_i=0$ if $i\ne k$ at the k-th level of the search tree. The variable OK represents whether the necessary condition holds or not.

The block diagram of this adjacency checker is shown in Fig. 4. The blocks D, M1, M2, B, and C correspond to Eq. (6), (7), (8), (9), and (10), respectively. It should be noted that this adjacency checker of Konishi's algorithm is

†More details on the data-dependent circuit will be found in Sect. 3 and [11].

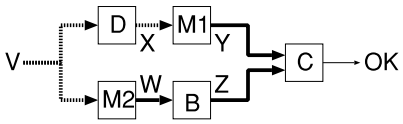

Fig. 4 Adjacency check circuit for Konishi's algorithm.

fully combinatorial and finishes the operation in one cycle, while Ullmann's refinement procedure generally requires multiple cycles before M converges.

Now that the adjacency checker is combinatorial, the data-dependent Konishi circuit can be easily composed by assigning constant values to $A = [a_{ij}]$ and $B = [b_{ij}]$. Design issues regarding the data-dependent circuit are described in the next section.

3. Design Issues Relating to Data-Dependent Circuits

3.1 Data-Dependent Circuit

Generally, the logic circuit can be reduced, if any input of the logic circuit turns out to be constant. Figure 5 illustrates some examples of this fact. If any inputs of an AND gate turn out to be zero, the output becomes zero (*constant propagation*). This reduction can be applied recursively, consequently reducing the logic scale of the circuit. The derived circuit would operate at a higher frequency than the original one, because the logic depth and wiring delay can thereby be reduced. As the consequent circuit becomes dependent on the input data instance, it is called a *data-dependent circuit* in the following discussion.

This technique is not limited to hardware. Even in conventional software, we can generate an input-specific program from an original program and its input data. This is known as *software specialization* or *partial evaluation* [5], which makes a program smaller and faster.

3.2 Issues in Data-Dependent Approaches

The obvious drawback of the data-dependent approach is that the derived circuit is *not reusable*, because it is specialized to an input instance. This naturally means that (1) we have to generate the circuit for each input, and (2) reconfigurable devices such as FPGA must be used.

The total execution time T of a data-dependent circuit is given by the sum of the circuit generation time T_{gen} and the execution time T_{exec}.

$$T = T_{gen} + T_{exec} \tag{11}$$

Here, T_{gen} consists of the time for HDL source code generation, logic synthesis, technology mapping, placement, routing, bitstream generation, and configuration of FPGA. T_{gen} is strongly related to the logic scale, since a larger circuit usually takes more time for circuit generation. T_{exec} is estimated by the product of the cycle count and cycle time. The cycle count depends on the algorithm, while the cycle time depends on the logic depth of the circuit.

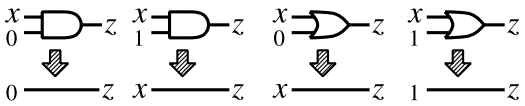

Fig. 5 Reduction of logic gates.

Although there are many algorithms for a problem, the selection is not easy. Even the fastest algorithm is of no use, if it requires too many hardware resources for practical applications. A complicated design may reduce cycles, while it can make cycle time larger.

Particularly for data-dependent circuits, T_{gen} must be considered. Although fast algorithms make T_{exec} smaller, they often require more hardware resources and make T_{gen} larger. It should be also noted that both T_{gen} and T_{exec} are dependent on the input data. Therefore, the total execution time T is not so obvious without empirical studies.

3.3 Design Methodology

The simplest way to implement a data-dependent circuit is to assign constant values to the inputs of the original circuit description. For example, we can fix input graphs and generate a data-dependent circuit by giving constant values to $A = [a_{ij}]$ and $B = [b_{ij}]$ in Fig. 3. Then, logic synthesis tools automatically try to reduce logic gates by applying constant propagation recursively. A serious problem is that it takes too much time and memory space.

Ichikawa et al. [11] earlier conducted a preliminary study on data-dependent circuits of Ullmann's algorithm, and found that simple preprocessing techniques drastically reduce circuit generation time T_{gen}. These techniques are adopted in the following evaluations in Sect. 4. An earlier study provides more details on the techniques [11].

The data-dependent versions of Konishi's algorithm are also implemented and evaluated in Sect. 4. We adopted the new design described in Sect. 2.5, and generated data-dependent designs with the preprocessing techniques similar to that used for Ullmann's data-dependent circuit [11].

4. Evaluation

In this section, the following 4 designs are examined.

Uo The original Ullmann circuit described in Sect. 2.3.
Ud The data-dependent version of Ullmann's circuit. This design is basically the same as BOTH2 in the previous study [11] except that the target FPGA architecture is different. Though BOTH2 was originally targeted for Lucent OR2C FPGA, Ud is retargeted for Xilinx XC2V FPGA [14] in the present study.
Ko The new design of Konishi's algorithm described in Sect. 2.5.
Kd The data-dependent version of Konishi's circuit (Ko).

Though the datapath of Ud is simplified from that of Uo by constant propagation, the control circuit of Ud is exactly the same as that of Uo. That is, the cycle count of Ud is

exactly the same as for Uo if the input data are the same. Since the same clock signal is used for all designs (as shown in Sect. 4.1), the execution time of Ud and Uo is the same for the same input data. This relationship also holds for Kd and Ko. Therefore, we do not have to measure the execution time of Uo and Ko in the following discussion.

4.1 Evaluation Environment

The evaluation environment is summarized in Table 2. Our target FPGA board is MIRE-MULTI3000, which contains a Xilinx XC2V3000 FPGA [14] and a 50 MHz clock generator. In this study, all designs use 25 MHz clock, which is generated from an on-board 50 MHz master clock. Some pins of XC2V3000 are connected to a parallel I/O board, which is used to start, to sense, and to stop the circuit in the FPGA from the host computer. On-board LEDs were also used for debugging and verification.

Data-dependent circuits (Ud and Kd) are designed on a host computer, which is a commodity PC with an Athlon XP processor. First, the source code generator emits a VHDL source code, which is dependent on input graphs G_α and G_β. This VHDL code is passed to a logic synthesis system to generate the netlist of a data-dependent circuit. The clock constraint of synthesis is set to 27 MHz to give a 3-ns margin to the following process. The netlist is then passed to Xilinx ISE for mapping, placement, routing, and bitstream generation. The bitstream is downloaded by a JTAG cable to configure the FPGA. The circuit generation time T_{gen} includes the time from VHDL code generation to bitstream download. The host computer then starts the circuit, waits until the end of execution, and stops the circuit, while measuring the execution time t_{exec}.

Although each data-dependent circuit was verified at 25 MHz system clock, the measured execution time t_{exec} does not fairly represent the possible execution time. Therefore, the maximal operational frequency f_{max} of each data-dependent circuit was examined after placement and routing, and the possible execution time T_{exec} was estimated by the following equation.

$$T_{exec} = t_{exec} \times \frac{25 \times 10^6}{f_{max}} \tag{12}$$

This T_{exec} is used as the execution time of a data dependent circuit in the following discussion[†].

4.2 Data Set

Obviously, the execution time of a subgraph isomorphism problem is heavily dependent on the input graphs. Therefore, the input data set must be carefully chosen in evaluation. In this study, we only examine the cases of $p_\alpha = p_\beta$ to reduce simulation parameters. Though it would be interesting to investigate more other cases, we decided to leave them to future studies. The number of vertices is hence designated by p, instead of p_α and p_β, in the following discussion. Each result in this section is the average of 100 pairs of G_α and G_β, which are random graphs [3]. A perfect graph K_p consists of p vertices and $p(p-1)/2$ edges, while a random graph with p vertices is generated to include each edge of K_p at a constant probability ϵ (*edge probability*). The symbol ϵ_α and ϵ_β designate the edge probability of graph G_α and G_β, respectively.

Figure 6 displays the software execution time of Ullmann's algorithm for various graph sets. The evaluation environment is shown in Table 2. In Fig. 6, RG0306 stands for the data set of 100 random graph pairs of $(\epsilon_\alpha, \epsilon_\beta) = (0.3, 0.6)$. As readily seen, RG0306 is the most time-consuming data set, while RG0603 is the fastest. RG0303 and RG0606 are between RG0306 and RG0603. Since the acceleration by custom circuit is particularly needed for time-consuming cases, we examine the data set RG0306 in

[†]It should be noted that this T_{exec} was derived from 27 MHz timing constraint. Faster data-dependent circuits might be generated with tighter timing constraints, in exchange for longer T_{gen}.

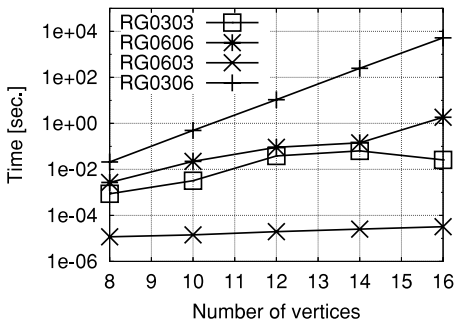

Fig. 6 Software execution time of subgraph isomorphism problems.

Table 2 Evaluation environment.

Item	Note
Host Computer	Athlon XP 2600+ (2.1 GHz), Memory 1 GB, Windows SP4
Evaluation Board (FPGA)	YDK MIRE-MULTI3000 (Xilinx Virtex-II XC2V3000FF1152-4)
JTAG cable	Xilinx Parallel Cable IV
Parallel I/O	ADTEK aPCI–P31A
VHDL Source Code Generation	Written in C, compiled with gcc-3.2 on Cygwin
Synthesis	Synopsys FPGA Compiler II (2001.08-FC3.7)
	(Clock Constraint: 27 MHz, used with options - Optimize: Area, Effort: Low)
FPGA CAD	Xilinx ISE 5.2i (for Virtex-II architecture)
Software Implementation	Ullmann's algorithm, Written in C, compiled with gcc-3.2.2
	Athlon XP 2600+ (2.1 GHz), Memory 512 MB, Red Hat Linux 9

the following discussion. Nonetheless, the data-dependent approach is equally applicable to other cases, e.g., RG0303 and RG0606.

4.3 Logic Scale

Figure 7 displays the average logic scale of Ud and Kd for RG0306 ($8 \le p \le 16$). For comparison, the original designs Uo and Ko are also plotted on Fig. 7. The logic scale is measured by the number of *slices* [14].

For example, in case of $p = 16$, the logic scale of Kd was 59.9% of Ko, and that of Ud was 29.7% of Uo on average. The reduction of logic scale is larger, if ϵ_α and ϵ_β are smaller. The average usage of slices is 3.5% (Kd) and 8.2% (Ud) of an XC2V3000 FPGA.

Ud was 1.43 times larger than Ko and 2.38 times larger than Kd for $p = 16$. This ratio becomes larger when p is larger. It is because the resource requirement of Ud and Uo is $O(p^3)$, and $O(p^2)$ for Kd and Ko [9].

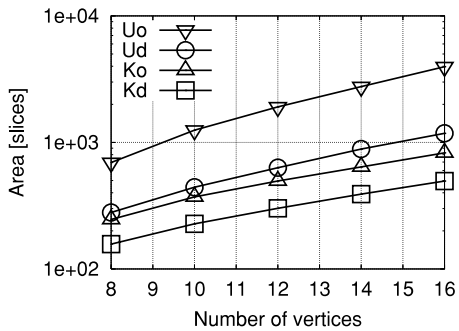

Fig. 7 Logic scale of RG0306.

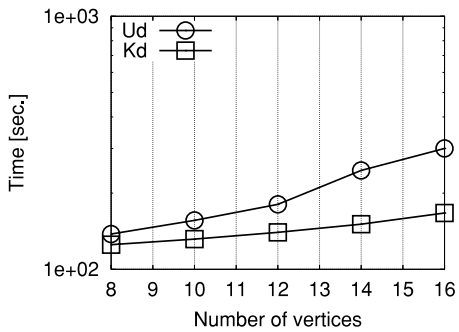

Fig. 8 Circuit generation time of RG0306.

4.4 Circuit Generation Time

The circuit generation time is very important in the data-dependent approach, as stated in Sect. 3.2. The measurement results of circuit generation time T_{gen} for RG0306 are summarized in Fig. 8 and Table 3.

For example, T_{gen} of Ud is 1.80 times larger than that of Kd for $p = 16$. This difference expands for larger p. The dominant factor is logic synthesis in Kd, and P&R (placement and routing) in Ud.

4.5 Execution Time

First of all, the software implementation of Ullmann's algorithm was evaluated by an Athlon XP processor. The execution environment is detailed in Table 2. The evaluation results of software are denoted by **Soft** in Fig. 9. The performance optimization of this software is very important, because it is used as a basis of evaluation in the following discussion. We adopted the following optimization techniques to improve software performance.

- Since many bit operations are required, 32 logic values were packed in one integer variable to implement 32 logic operations with one logical instruction.
- The iteration of the innermost loop was designed to be long enough to suppress the startup overhead of the loop.
- The innermost loop was designed to avoid *stride accesses* to utilize data cache fully.

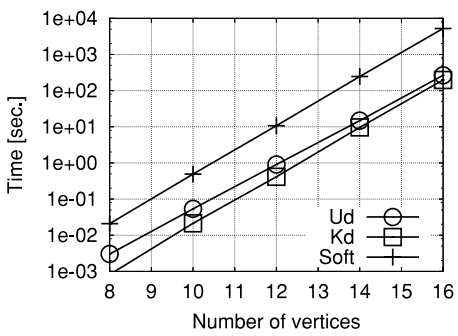

Fig. 9 Execution time of RG0306.

Table 3 Detailed circuit generation time T_{gen} of RG0306 (sec.).

Design	p	Area [slice]	VHDL	Synthesis	Mapping	P&R	Bitstream	Download	T_{gen}
Kd	8	157.07	0.017	24.74	5.25	29.83	23.45	41.86	125.14
	10	227.27	0.018	28.11	5.77	32.36	23.58	41.81	131.65
	12	301.65	0.019	32.54	6.46	35.52	23.65	41.81	140.00
	14	390.29	0.021	38.93	7.29	38.84	23.84	41.81	150.73
	16	495.12	0.022	48.45	8.03	44.63	24.00	41.82	166.95
Ud	8	279.66	0.021	32.91	6.34	33.17	23.62	41.81	137.87
	10	442.72	0.024	41.07	7.73	41.45	24.03	41.82	156.13
	12	630.60	0.028	51.74	9.44	53.08	24.40	41.81	180.51
	14	886.50	0.033	67.76	11.76	99.34	25.11	41.81	245.82
	16	1180.01	0.040	89.55	14.53	128.53	25.81	41.82	300.28

- Data structures were designed to sustain locality of memory access. Moves and copies of data are avoided wherever possible.
- Common subexpressions were eliminated by reusing their values.

Figure 9 shows that Kd and Ud are much faster than Soft. For instance, with $p = 16$, Kd is 26.5 times faster than Soft on Athlon XP processor (2.1 GHz clock). Ud was 19.6 times faster on the same condition. Such performance gain becomes larger for larger p, because Kd and Ud have $O(p)$ and $O(p^2)$ parallelism in the hardware, while Soft is sequential. It should be also noted that an XC2V3000 FPGA may contain at most 28 units of Kd, since a Kd unit occupies only 3.5% of an XC2V3000 in case of $p = 16$.

Kd is faster than Ud in Fig. 9 in spite of its simpler pruning scheme. For $p = 16$, the execution time of Kd and Ud were 3.77% and 5.09% of that of Soft (5.22×10^3 seconds), respectively, because the adjacency check of Kd is faster than for Ud, as stated in Sect. 2.5. For larger graphs, Ud is expected to be faster because the refinement procedure of Ud restricts the search space more efficiently than Kd.

The execution time grows exponentially according to the graph size p, which is readily seen in Fig. 9. If the problem size is a few times larger, the execution time would be longer by a few orders of magnitude. The custom circuit is particularly required for large problems, which take too long to execute by software. In that sense, the data-dependent approach is significant, because it makes it possible to implement larger problems with the same hardware resources.

4.6 Total Execution Time

Figure 10 shows the total execution time, $T = T_{gen} + T_{exec}$ of Kd and Ud, along with the software execution time (Soft). Kd and Ud are faster than Soft for $p \geq 15$, even if the circuit generation time is included. For example, in case of $p = 16$, the total performance gain of Kd and Ud are 14.4 and 9.2, respectively.

By comparing Fig. 8 and Fig. 9, it is obvious that T_{exec} overwhelms T_{gen} with large graphs ($p > 16$). For small graphs ($p < 16$), T_{gen} is superior to T_{exec}. As the T_{gen} of Kd is smaller than Ud, Kd is a better solution than Ud for small graphs ($p < 16$) of RG0306. However, these small graphs are not suited to the data-dependent approach, because Soft is fast enough in this area.

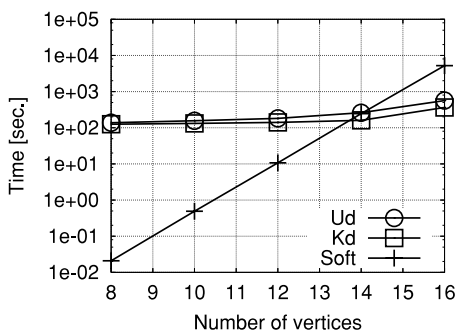

Fig. 10 Total execution time of RG0306.

Although large graphs are of interest, it is too time-consuming and impractical to measure 100 input pairs to obtain the average value for such graphs. Therefore, we introduce and use approximation models in the next section.

4.7 Approximation and Extrapolation

The resource requirement of Kd and Ud are known to be $O(p^2)$ and $O(p^3)$, respectively [9]. The logic scale (or area) of Kd and Ud are hence approximated by the following polynomials, where $Area_K$ and $Area_U$ stand for the area of Konishi's circuit and Ullmann's circuit. The constant factors $k_i\,(i = 0, 1, 2, 3)$ are fitting parameters that are specific to each equation.

$$Area_K(x) = k_0 + k_1 x + k_2 x^2 \tag{13}$$

$$Area_U(x) = k_0 + k_1 x + k_2 x^2 + k_3 x^3 \tag{14}$$

The approximation equation of T_{gen} is not obvious, but we assumed the following equation based on measurement results. This approximation fits to reality fairly well.

$$T_{gen}(x) = k_0 e^{k_1 x} + k_2 \tag{15}$$

As shown in Sect. 2.2, the search space of subgraph isomorphism problems with p vertices is approximately $p!$ (i.e., the factorial of p). Although pruning procedures can reduce the search space to some extent, they can not reduce the computational complexity of the problems. Therefore, T_{exec} is approximated by the following equations, where the function $fact(x)$ stands for Stirling's formula to approximate $x!$. This approximation equation was confirmed by measurements, and it fits to reality very well.

$$T_{exec}(x) = k_0 \cdot fact(k_1 x) \tag{16}$$

$$fact(x) \approx \sqrt{2\pi x}\left(\frac{x}{e}\right)^x \tag{17}$$

Now, we extract fitting parameters from the measurement results. In case of Ko and Uo, we only have to generate one circuit for each p, because Ko and Uo are independent of the inputs. Therefore, it was possible for us to measure the Uo and Ko for large p. $Area_{Uo}$ was modeled from the measurement results of $8 \leq p \leq 40$, and $Area_{Ko}$ was modeled from $8 \leq p \leq 100$. $Area_{Ud}$ and $Area_{Kd}$ were modeled from the technology mapping results of $8 \leq p \leq 40$ and $8 \leq p \leq 70$, respectively.† Soft were extracted from the measurements of $8 \leq p \leq 16$. Table 4 summarizes the fitting parameters for RG0306 data sets.

Figure 11 displays the approximated logic scale of Ko, Kd, Uo, and Ud. In the figure, the symbol Ko' is used to distinguish the approximated values from the measured values

†Although we did not place, route, and execute the data-dependent circuits of $p > 16$, we synthesized and mapped them to derive area estimation. The mapping reports for large p make the area estimation model more precise.

Table 4 Fitting parameters for RG0306.

Item	Design	k_0	k_1	k_2	k_3
Area	Ko	−96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	47.704	−0.573	1.805	—
	Ud	254.621	−37.782	4.040	0.143
T_{gen}	Kd	4.620	0.158	109.009	—
	Ud	11.520	0.184	84.711	—
T_{exec}	Soft	6.468e-05	0.710	—	—
	Kd	2.640e-06	0.708	—	—
	Ud	1.696e-05	0.667	—	—

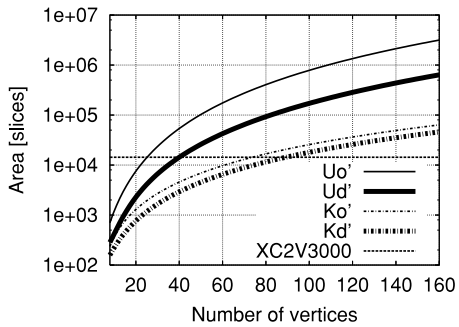

Fig. 11 Estimated logic scale of RG0306.

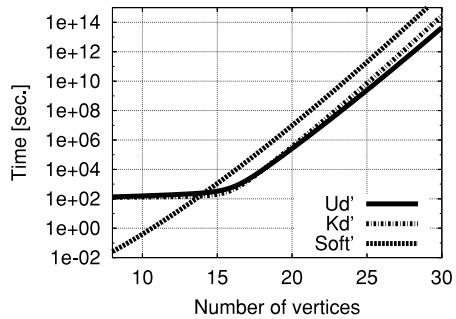

Fig. 12 Estimated total execution time of RG0306.

(Ko). As an XC2V3000 contains 14336 slices, the corresponding line is also shown in the figure. From Fig. 11, an XC2V3000 is expected to contain a Uo design of $p < 25$. Ud is smaller than Uo and thus fits into an XC2V3000 when $p < 40$. Ko and Kd are still smaller. An XC2V3000 can contain a Ko of $p < 70$ or a Kd of $p < 90$. These figures seem promising for practical applications.

Figure 12 displays the approximated total execution time of Soft, Kd, and Ud. The respective performance of Kd and Ud would be about 30 times and 100 times higher than Soft for $p = 30$. It is readily seen that there is a phase transition around $p = 16$. For $p > 16$, T_{exec} overwhelms T_{gen}. On the other hand, T_{gen} is superior to T_{exec} for $p < 16$.

Figure 13 displays the estimated area-time product, which is the product of the number of slices and the execution time. The area-time product is regarded as a measure of the ratio of cost to performance, because the cost is almost proportional to the area, while the performance is defined by the reciprocal of execution time. According to Fig. 13, Kd seems more cost-effective than Ud for the RG0306 data set.

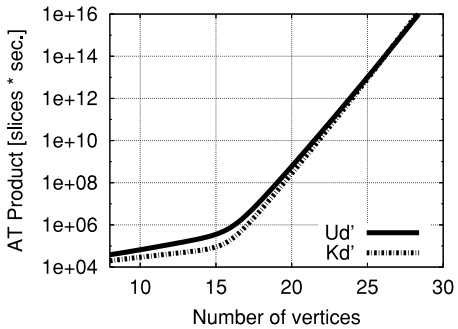

Fig. 13 Estimated area-time product of RG0306.

4.8 Another Data Set

Having examined the RG0306 data set in the above discussion, we examine another data set in this section. The new data set is called IN0306, and has the following properties.

1. All graphs are connected. Though a random graph is not always connected, many applications assume that the input graph is connected.
2. All input graph pairs have subgraph isomorphism. In some applications, it is not a problem whether G_α is included in G_β or not; Rather, the problem is where it is located.

In each graph pair of IN0306, G_α and G_β have an edge density of 0.3 and 0.6, respectively. Edge density ed is a parameter that represents a character of a graph. Assume a graph with p vertices and q edges. Edge density ed is defined by the following equation: $ed = (2\,q)/(p\,(p-1))$. That is, ed is the ratio of the number of edges to that of the perfect graph K_p. It is obvious that the following relationship holds: $0 \leq ed \leq 1$. It is also obvious that RG0306 and IN0306 have the same number of edges on average for the same number of vertices.

We generated data-dependent circuits, and measured the data for IN0306 as well as RG0306. We omit the detailed results of IN0306 for lack of space, and show only the output of approximated models of IN0306. The fitting parameters are summarized in Table 5, which was derived on the same condition as Table 4.

The logic scale of IN0306 is almost the same as for RG0306 (Fig. 14). Since the logic scale of data-dependent circuit is mainly dependent on the number of edges, data-dependent approach is almost equally effective for RG0306 and IN0306.

For large graphs of IN0306, Kd is slower than Ud, although it is still much faster than Soft (Fig. 15). Konishi's algorithm is not well suited for a connected graph, while Ullmann's refinement procedure works efficiently for IN0306. For the same reason, the area-time product of Kd is slightly inferior to Ud for $p > 20$ (Fig. 16).

Table 5 Fitting parameters for IN0306.

Item	Design	k_0	k_1	k_2	k_3
Area	Ko	−96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	42.210	0.537	1.790	—
	Ud	571.270	−107.502	8.726	0.061
T_{gen}	Kd	5.358	0.152	105.809	—
	Ud	16.145	0.166	73.852	—
T_{exec}	Soft	9.208e-05	0.657	—	—
	Kd	2.354e-06	0.684	—	—
	Ud	2.411e-05	0.608	—	—

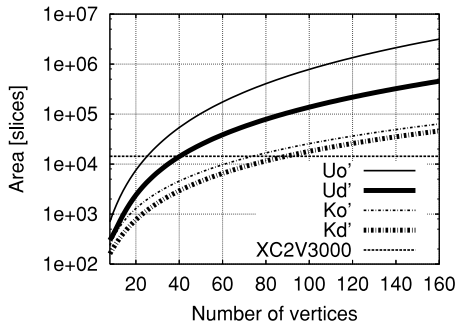

Fig. 14 Estimated logic scale of IN0306.

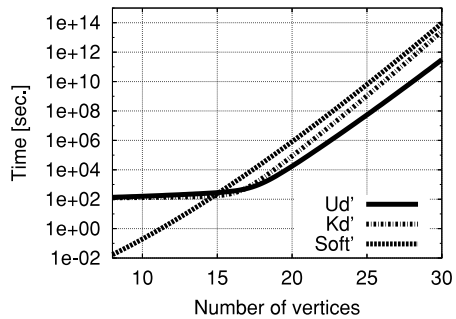

Fig. 15 Estimated total execution time of IN0306.

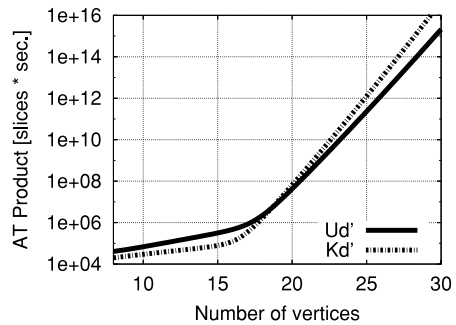

Fig. 16 Estimated area-time product of IN0306.

5. Conclusion

This study described the design and evaluation of data-dependent circuits for subgraph isomorphism problems. The original contributions of this study are summarized as follows:

- Data-dependent approach was shown to be superior to the software on a state-of-the-art microprocessor, even if the circuit generation time is included.
- Data-dependent circuits were actually implemented on Xilinx XC2V FPGA, and successfully operated on an FPGA board.
- The superiority of data-dependent approach is not specific to an algorithm or an data set. Two different algorithms and two distinct data sets were examined, and the superiority of data-dependent approach was sustained in all these cases.

For example, with random graphs having 16 vertices, the execution time was about 7.0% of the software executed on an up-to-date microprocessor (Athlon XP 2600+ of 2.1 GHz clock). Even if the circuit generation time is included, Konishi's data-dependent circuits were 14.4 times faster than the software in the case of 16 vertices. The data-dependent approach was equally effective for both of the two algorithms (Ullmann's and Konishi's). For data-dependent circuits of random graphs, Konishi's design was generally more cost-effective than Ullmann's. Though Konishi's circuit was slightly slower than Ullmann's in case of large graphs, the difference was little. For data-dependent circuits of connected graphs, Ullmann's circuit was faster and more cost-effective for large graphs.

It should be noted that an all-round algorithm is not required for data-dependent circuits. In data-dependent circuit design, we can choose an algorithm on a case-by-case basis, considering the nature of the input instance. Since the inputs of many application programs show evident biases, we can provide a data-specific optimization technique for these frequently expected input patterns. According to the results of this study, a data-dependent approach is expected to be effective regardless of data sets and algorithms.

In this study, the estimations of logic scale and execution time for large problems were shown. However, the data-dependent circuits for such large problems are not yet actually implemented and evaluated. The scalability of data-dependent approach is still an open problem and left for future studies.

The authors expect that a data-dependent approach is applicable to many other problems, particularly to hard-computation problems. Data-dependent approach is well suited for reconfigurable logic technology, and may be used in custom computing applications with FPGA devices.

Acknowledgments

This work was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS), the grant from the Okawa Foundation for Information and Telecommunications, and the grant from Amano Institute of Technology. Support for this work was provided by the 21st Century COE Program "Intelligent Human Sensing" from the Ministry of Education, Culture, Sports, Science and Technology. The custom circuits in this

study were designed with Synopsys CAD tools through the chip fabrication program of the VLSI Design and Education Center (VDEC), the University of Tokyo.

References

[1] J. Babb, M. Frank, and A. Agarwal, "Solving graph problems with dynamic computation structures," SPIE Photonics East: Reconfigurable Technology for Rapid Product Development and Computing, pp.225–236, 1996.
[2] J.D. Bingham and M. Serra, "Solving Hamiltonian cycle on FPGA technology via instance to circuit mappings," Workshop on Engineering Reconfigurable Hardware/Software Objects, Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA2000), CSREA Press, June 2000.
[3] B. Bollobás, Random Graphs, second ed., Cambridge University Press, 2001.
[4] S.T. Chakradhar and V.D. Agrawal, "A novel VLSI solution to a difficult graph problem," Proc. 4th CSI/IEEE Int'l Symp. VLSI Design, pp.124–129, IEEE Computer Society, 1991.
[5] C. Consel and O. Danvy, "Tutorial notes on partial evaluation," Proc. 20th ACM Symp. on Principles of Programming Language, pp.493–501, ACM, 1993.
[6] A. Dandalis, A. Mei, and V.K. Prasanna, "Domain specific mapping for solving graph problems on reconfigurable devices," Proc. 6th Reconfigurable Architecture Workshop, IPPS/SPDP Workshops 1999, pp.652–660, IEEE, 1999.
[7] M.R. Garey and D.S. Johnson, Computers and Intractability, Freeman, 1979.
[8] L. Huelsbergen, "A representation for dynamic graphs in reconfigurable hardware and its application to fundamental graph algorithms," Proc. ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA2000), pp.105–115, ACM, 2000.
[9] S. Ichikawa, H. Saito, L. Udorn, and K. Konishi, "Trade-offs in custom circuit designs for subgraph isomorphism problem," IEICE Trans. Inf. & Syst., vol.E86-D, no.7, pp.1250–1257, July 2003.
[10] S. Ichikawa, L. Udorn, and K. Konishi, "An FPGA-based implementation of subgraph isomorphism algorithm," IPSJ Transactions on High Performance Computing Systems, vol.41, no.SIG5 (HPS1), pp.39–49, 2000.
[11] S. Ichikawa and S. Yamamoto, "Data dependent circuit for subgraph isomorphism problem," IEICE Trans. Inf. & Syst., vol.E86-D, no.5, pp.796–802, May 2003.
[12] O. Mencer, Z. Huang, and L. Huelsbergen, "HAGAR: Efficient multi-context graph processors," Proc. 12th Int'l Conf. Field-Programmable Logic and Applications (FPL2002), LNCS2438, pp.915–924, Springer, 2002.
[13] J.R. Ullmann, "An algorithm for subgraph isomorphism," J. ACM, vol.23, no.1, pp.31–42, 1976.
[14] Xilinx Inc., Virtex™-II Platform FPGAs: Introduction and Overview, 2002.
[15] H. Yamamoto and S. Ichikawa, "Evaluation of data-dependent circuits for adjacency check," Proc. IEICE Gen. Conf. 2003, p.64, 2003.

Shoji Yamamoto received his B.E. degree in 2002 and M.E. degree in 2004 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. He is presently with Buffalo Inc.

Shuichi Ichikawa received his D.S. degree in Information Science from the University of Tokyo in 1991. He has been affiliated with Mitsubishi Electric Corporation (1991–1994), Nagoya University (1994–1996), and Toyohashi University of Technology (since 1997). Currently, he is an associate professor of the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. His research interests include parallel and distributed processing, high-performance computing, and custom computing systems. He is a member of ACM, IEEE, and IPSJ.

Hiroshi Yamamoto received his B.E. degree in 2003 from the Department of Knowledge-based Information Engineering of Toyohashi University of Technology. Presently, he is studying for his master's degree at that institution. He is a student member of the IPSJ.