PAPER

Trade-Offs in Custom Circuit Designs for Subgraph Isomorphism Problems[****]

Shuichi ICHIKAWA[†], ***Regular Member***, **Hidemitsu SAITO**[†*], **Lerdtanaseangtham UDORN**[†**], ***and*** **Kouji KONISHI**[†***], ***Nonmembers***

SUMMARY Many application programs can be modeled as a subgraph isomorphism problem. However, this problem is generally NP-complete and difficult to compute. A custom computing circuit is a prospective solution for such problems. This paper examines various accelerator designs for subgraph isomorphism problems based on Ullmann's algorithm and Konishi's algorithm. These designs are quantitatively evaluated from two points of view: logic scale and execution time. Our study revealed that Ullmann's design is faster but larger in logic scale. Partially sequential versions of Ullmann's algorithm can be more cost-effective than Ullmann's original design. The hardware of Konishi's algorithm is smaller in logic scale, operates at a higher frequency, and is more cost-effective.

key words: *NP-complete, graph, algorithm, FPGA*

1. Introduction

The 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. An example of this is shown in Fig. 1. In this figure, G_β has a subgraph that is isomorphic to G_α, whereas G_γ does not.

Many application programs, including scene analysis and a chemical graph database [10], are modeled as subgraph isomorphism problems. However, a subgraph isomorphism problem is generally NP-complete [3] and difficult to compute in a practical length of time. There is a strong desire among application developers to shorten the processing time of subgraph isomorphism detection.

Many such difficult computation problems are strongly computation intensive. The amount of data is small and communication time is negligible in comparison to computation time. All these properties seem preferable for resorting to acceleration by custom computing machinery.

This paper examines various accelerator designs for subgraph isomorphism problems based on Ullmann's algorithm [11] and Konishi's algorithm [6]. These designs are quantitatively evaluated from two points of view: logic scale and execution time. According to our results, Ullmann's design is faster but larger in logic scale. Partially sequential versions of Ullmann's algorithm can be more cost-effective than his original design. Konishi's design is smaller in logic scale, operates at a higher frequency, and is more cost-effective.

In Sect. 2, the preceding studies on custom hardware for graph algorithms are summarized. Section 3 introduces three algorithms for subgraph isomorphism problems, i.e., an enumeration algorithm, Ullmann's algorithm, and Konishi's algorithm. Section 4 examines some custom circuit designs of these algorithms. In Sect. 5, these designs are quantitatively evaluated from two points of view: "logic scale" and "execution time." Section 6 briefly summarizes our results.

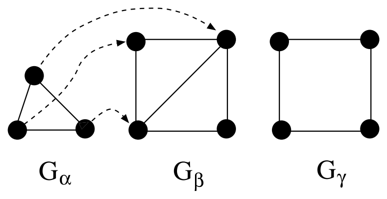

Fig. 1 Subgraph isomorphism.

2. Related Works

There are few studies on custom hardware for graph isomorphism problems, including subgraph isomorphism problems.

Ullmann [11] introduced an algorithm for subgraph isomorphism which has been very popular in recent years. Ullmann showed that the *refinement procedure* of his algorithm can be implemented by a parallel hardware for faster execution, but his paper included neither a detailed discussion nor actual implementations [11].

The graph isomorphism problem can be formulated as a constraint satisfaction problem (CSP). Swain and Cooper [9] presented a parallel hardware implementation of constraint satisfaction by arc consistency. They mentioned graph matching as a possible application of their circuitry, though their design is not optimized for graph isomorphism. Moreover, no actual implementation was described in their work.

Manuscript received August 27, 2002.
Manuscript revised February 15, 2003.

[†]The authors are with the Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi-shi, 441–8580 Japan.
[*]Presently, with Toshiba Corp.
[**]Presently, with Toyota Caelum Inc.
[***]Presently, with NTT Software Corp.

[****]This work was partially presented in the Proceedings of FPL2000, pp.729–738, LNCS 1896, Springer, 2000.

There are other studies on the custom computing engine for CSP. However, some are neither implemented nor evaluated [2], [12]. The DRA chip by Gu [4] is a VLSI implementation of a discrete relaxation algorithm (DRA), which is implemented by the $3\,\mu$ NMOS process. DRA is a general computational technique and is applicable to subgraph homeomorphism problems. However, no sufficiently detailed discussion or evaluation of graph problems is found in Gu's work.

Ichikawa, Udorn, and Konishi earlier proposed a new algorithm (Konishi's algorithm) for subgraph isomorphism problems, which is suitable for hardware implementation [6], [7]. The prototype hardware of Konishi's algorithm was implemented and evaluated on an FPGA (Field Programmable Gate Array), and it outperformed the software of Ullmann's algorithm on an off-the-shelf microprocessor. These works showed that Ullmann's original circuit is very large in logic scale compared to Konishi's. However, there have been no further studies of Ullmann's circuit.

In this paper, several implementations of Ullmann's algorithm are first examined in detail. These designs and that of Konishi are then evaluated and compared from various viewpoints. The purpose of this paper is to examine the trade-offs involved in custom circuit designs for subgraph isomorphism problems in a quantitative manner.

3. Subgraph Isomorphism Problem

First, let us define the problem. 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.

3.1 Enumeration Algorithm

As is easily seen, subgraph isomorphism can be determined by brute-force enumeration with a depth-first tree-search algorithm. Figure 2 shows an example of search trees. Assume that $V_\alpha = \{w_1, w_2, w_3\}$ and $V_\beta = \{v_1, v_2, v_3, v_4\}$. At the i-th stage of the search tree, w_i is mapped to a possible vertex in V_β. 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.

3.2 Ullmann's Algorithm

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

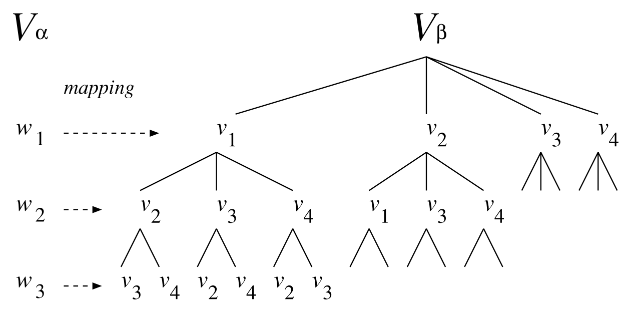
V_α
V_β
mapping
w_1
w_2
w_3
v_1 v_2 v_3 v_4
v_2 v_3 v_4 v_1 v_3 v_4
v_3 v_4 v_2 v_4 v_2 v_3

Fig. 2 Search tree.

The most popular algorithm is the one proposed by Ullmann, which is a smart tree-search algorithm with a *refinement procedure* for pruning [11]. 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. 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 dramatic, since the uncontrolled expansion of the search tree is effectively inhibited. A formal description of this refinement procedure is found in Ullmann's paper [11].

Ullmann discussed a parallel hardware implementation of his refinement procedure [11]. However, that requires $O(p_\alpha p_\beta{}^2)$ hardware resources which rapidly increases for larger p_α and p_β. Our previous study revealed that only small graphs can be handled with state-of-the-art FPGA [6]. Hence, an alternate way of pruning is required for practical implementation.

3.3 Konishi's Algorithm

The problem with a refinement procedure is that it checks not only mapped but also as yet unmapped vertices. This requires enormous resources. Ichikawa, Udorn, and Konishi [6] proposed a simplified pruning procedure which only handles mapped vertices. See Fig. 2 again. At the i-th level of the search tree, only vertices $w_1, \ldots, w_i$ ($1 \leq i \leq p_\alpha$) are mapped. In this algorithm, we only check the adjacency among these i vertices at the i-th level. 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. The simplified pruning procedure examines this necessary condition. In the following discussion, we denote this algorithm as *Konishi's algorithm* after its inventor. More details of

Konishi's algorithm are described in the paper [6].

An adjacency check of this algorithm is readily realized by referring to the adjacency matrix of G_β, instead of the expensive refinement procedure. Thus, this method reduces hardware resources to $O(p_\beta{}^2)$, which is small enough to fit into state-of-the-art hardware. In fact, this design was implemented and evaluated on a Lucent OR2C FPGA chip [6], [7].

4. Implementation Issues

In this section, several design alternatives are described and examined in detail. The evaluation results of these designs are shown in Sect. 5.

4.1 Ullmann's Original Design

Ullmann's idea [11] is to calculate $M = [m_{ij}]$ $(1 \leq i \leq p_\alpha,\ 1 \leq j \leq p_\beta)$ by parallel hardware. The element circuit that calculates m_{ij} is shown in Fig. 3. Here, $A = [a_{ij}]$ $(1 \leq i, j \leq p_\alpha)$ is the adjacency matrix of graph G_α, and $B = [b_{ij}]$ $(1 \leq i, j \leq p_\beta)$ is the adjacency matrix of G_β. The matrix M contains temporary variables used in the refinement procedure. Let us denote the element in Fig. 3 by *sub_comb*. The whole circuit that updates M is implemented by $p_\alpha \times p_\beta$ matrix of sub_comb.

As each r_{ij} in Fig. 3 consists of $O(p_\beta)$ gates, $O(p_\alpha p_\beta)$ gates are required for each m_{ij}. The total hardware resource for $p_\alpha p_\beta$ units of sub_comb would be $O(p_\alpha p_\beta{}^2)$, since r_{kj} $(1 \leq k \leq p_\alpha)$ can be shared among m_{ij} $(1 \leq i \leq p_\alpha)$. Some control circuits are additionally required for search tree traversal and termination check, but they are not dominant. We denote this combinatorial implementation as *comb* in the following discussion.

4.2 Sequential Implementations of Ullmann's Circuit

4.2.1 Design Approaches

Although the combinatorial implementation is too costly, it is possible to reduce the amount of hardware by designing a sequential circuit. In sequential designs, the subcircuits are time-shared by using input multiplexers and sequence controllers. Consequently, a sequential circuit incurs more processing time than a combinatorial circuit.

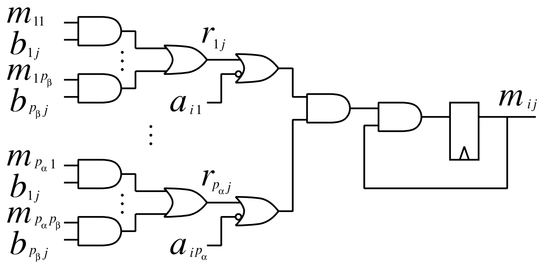

Fig. 3 Combinatorial circuit for m_{ij} (sub_comb).

The following are three trivial designs to reduce the number of sub_comb. Figure 4 illustrates the ideas behind these partially sequential designs.

- **seq_i** Modify M row by row, using p_β units of sub_comb.
- **seq_j** Modify M column by column, using p_α units of sub_comb.
- **seq_i_j** Modify M element by element, using a single unit of sub_comb.

Another possible design is to make sub_comb sequential. The p_α-input AND of sub_comb can be implemented sequentially. This idea is illustrated in Fig. 5. This circuit is denoted as *sub_comb_x* in this paper. We can compose the following sequential implementations with sub_comb_x.

- **seq_x** Modify all m_{ij} in parallel, using $p_\alpha p_\beta$ units of sub_comb_x.
- **seq_i_x** Modify M row by row, using p_β units of sub_comb_x.
- **seq_j_x** Modify M column by column, using p_α units of sub_comb_x.
- **seq_i_j_x** Modify M one by one, using a single unit of sub_comb_x.

The design seq_i_j_x is theoretically possible, but it is overly serialized and too slow for practical use, e.g., making it impossible even to finish the simulation of the benchmark data set in practical time. Therefore, we disregard seq_i_j_x in the following discussion.

The required orders of combinatorial gates are

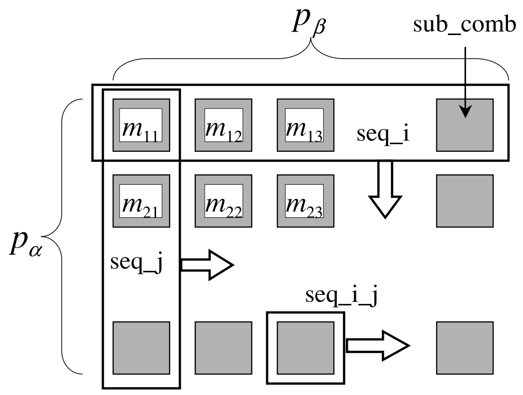

Fig. 4 Sequential implementations: seq_i, seq_j, seq_i_j.

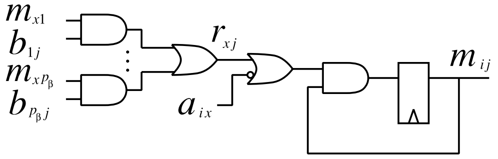

Fig. 5 Sequential circuit for m_{ij} (sub_comb_x).

Table 1 Memory implementation.

Design	Matrix		
	A	B	M
comb	FF	FF	FF
seq_i	RAM	FF	FF
seq_i_j	RAM	RAM	FF
seq_j	FF	RAM	FF
seq_x	RAM	FF	FF
seq_i_x	RAM	FF	FF
seq_j_x	RAM	RAM	FF

summarized in Table 3 in Sect. 5 for each design. You can see how the hardware requirements decrease in sequential implementations.

The order of memory resources is $O({p_\beta}^2)$ for each design. It is essential to implement adjacency matrices (A, B, M), which cost $O({p_\beta}^2)$. There are additional memory resources (e.g., for state machines), but they are negligible.

Even when the required order of memory is the same, the implementation of memory could be different. Just consider the adjacency matrix B, which consists of ${p_\beta}^2$ elements. If all the elements are used at the same time, this matrix must be implemented by a $p_\beta \times p_\beta$ array of flip-flops. On the other hand, if the circuit is sequential and uses only p_β elements out of ${p_\beta}^2$ at a time, this matrix can be implemented by a $p_\beta \times p_\beta$ RAM. Generally speaking, RAM is much more dense and suitable for integration. Table 1 summarizes the implementations of matrices in each design.

4.2.2 Short Circuit Evaluation

In Ullmann's algorithm, the refinement procedure fails if any row of M turns out to be all zeros. This is called the "FAIL exit" [11], and if such a failure occurs on any node in the search tree, the subtree under that node is pruned. This condition is implemented by a p_β-input OR-gate for each row. In Ullmann's original design, p_α OR-gates are used to detect this condition simultaneously for all rows.

This brings up another advantage of sequential implementation. For example, consider the case of seq_i (Fig. 4), in which M is updated row by row. If the fail condition is detected in the first row, we need not calculate any other rows, but can just terminate the refinement procedure and return a "FAIL" to prune the subtree below this node. This will save many cycles in sequential execution. This kind of optimization is commonly known as "short circuit evaluation of boolean expression" in the compiler area [1].

Such a short circuit evaluation is also applicable to other designs. For example, it is effective for seq_x. As you can see in Fig. 5, m_{ij} is never turned on but only turned off by AND-gate in seq_x. While m_{ij} is sequentially updated by increasing x from 1 to p_α, some of the row of the matrix M can become all zeros before x comes to p_α, saving many cycles of returning a FAIL from the refinement procedure.

It is very simple as well as economical to implement short circuit evaluations in sequential designs. Thus, we adopt this mechanism in all sequential designs in the following discussions. The effect of the short circuit evaluation is shown in Table 5 of Sect. 5.

4.3 FPGA-Specific Issues

There are technology-specific issues besides the general issues. Let us take a close look at FPGA technology here, since we adopt FPGA as the evaluation platform in the following discussion.

A SRAM-based FPGA device generally consists of an array of logic elements and wiring resources. Such a logic element has various names, e.g., PFU (Programmable Function Unit) in Lucent OR2C FPGA, LE (Logic Element) in Altera APEX FPGA, and CLB (Configurable Logic Block) in Xilinx Virtex FPGA. The functions and structure of PFU, LE, and CLB are similar to each other. They consist of the LUT (look-up table)[†], FF (flip-flop), tri-state output buffer, and some control circuits. In the following discussion, we adopt Lucent OR2C FPGA [8], which is used in the evaluation of designs.

As discussed in Sect. 4.2.1, memory can be implemented either with LUT or with FFs. In the case of OR2C FPGA, 16×4 bits can be implemented by using LUT in one PFU, while only 4 bits are implemented by using FFs in one PFU. However, the use of FFs does not necessarily mean the additional use of PFUs in FPGA. One should keep in mind that each PFU includes flip-flops. If any combinatorial gates are implemented by LUTs, there already exist some flip-flops within such LUTs whether they are necessary or not. Hence, FFs are often absorbed by the PFUs for combinatorial gates, resulting in no additional use of PFUs. The design itself and the mapping quality determine how many FFs could be absorbed.

Similar tricks exist for multiplexers, which can be implemented either with LUTs or with tri-state buffers in a SRAM-based FPGA. If multiplexers are implemented with LUTs, they will consume PFUs. On the other hand, tri-state buffers are found in every PFUs but they rarely used. If these idle buffers are utilized, multiplexers can be implemented at almost no cost. This trick also reduces the depth of logic, thereby making the circuit operational at a higher frequency. In the following evaluations, we tried to make full use of tri-state buffers for multiplexers.

4.4 Hardware Implementation of Konishi's Algorithm

The hardware implementation of Konishi's algorithm

[†]LUT is a kind of RAM.

is also examined here as a practical alternative to Ullmann's circuit. Since this algorithm had already been implemented and evaluated on the Lucent OR2C FPGA in the previous project [6], [7], we simply adopt this design in the following discussion.

The prototype was implemented on the OPERL board [5], which is a run-time reconfigurable PCI card with two Lucent OR2C FPGAs [8]. One is USER FPGA (OR2C15A), which contains an application circuit. USER FPGA is programmed and accessed from the host computer using PCI bus. Another is PCI FPGA (OR2C15A), which contains PCI interface circuitry and a run-time reconfiguration controller for USER FPGA. We implemented a unit that can handle up to $(p_\alpha, p_\beta) = (15, 15)$, which is a good fit with the basic component of OR2C FPGA (16×4 bit SRAM). This unit operates at 16.5 MHz, which is a half of PCI clock frequency. The unit could have been pipelined for 33 MHz operation to derive twice the performance, but we chose to keep things simple for this prototype. Even a single unit of the 16.5 MHz prototype outperformed the software implementation on a 400 MHz AMD K6-III processor. More details are found in references [6], [7].

5. Evaluation

Here we present various aspects of designs which include the original Ullmann's circuit (*comb*), the sequential circuits described in Sect. 4.2.1, and the circuit of Konishi's algorithm (*konishi*).

5.1 Logic Scale and Operational Frequency

The order of resources is important because it limits the scalability. However, the actual resource count on a certain technology is also important for determining the constant factor. Even designs that require the same order of resources can show great differences on an actual resource count.

To investigate the constant factor, we have to assume a specific technology or implementation. Here, we adopt Lucent OR2C series FPGA [8] as a measure. As we already have Konishi's hardware working on that device, it is a natural choice. We tuned each design for OR2C FPGA as much as possible by using a technology-dependent macro library. Though each implementation is not guaranteed to be best, we believe our design is not far from the optimal. Moreover, p_α and p_β must be fixed to make an implementation. Here, we designed the circuit for $(p_\alpha, p_\beta) = (15, 15)$.

The design environment is summarized in Table 2. The logic of each design is described in VHDL using the OR2C macro library. This VHDL code is processed by a logic synthesis system. Then the derived netlist is mapped onto OR2C technology to extract the logic scale and gate delay. In this feasibility study, placement and routing are not performed. Table 3 summarizes the PFU count and the operational frequency of each design. The operational frequency in Table 3 is based on the estimated gate delay, i.e., the wiring delay is not counted here.

Table 2 Design environment.

Item	Environment
Logic Synthesis	Synopsys Design Compiler
Technology Mapping	Lucent ORCA Foundry 9.35
Target Device	Lucent OR2CxxA FPGA

Table 3 Design results.

Design	Logic Scale			Freq.
	Gate	Memory	PFU	(MHz)
comb	$O(p_\alpha {p_\beta}^2)$	$O({p_\beta}^2)$	2754	22.5
seq_i	$O(p_\alpha {p_\beta}^2)$	$O({p_\beta}^2)$	1770	27.6
seq_i_j	$O(p_\alpha p_\beta)$	$O({p_\beta}^2)$	467	34.2
seq_j	$O(p_\alpha p_\beta)$	$O({p_\beta}^2)$	583	27.9
seq_x	$O({p_\beta}^2)$	$O({p_\beta}^2)$	671	23.1
seq_i_x	$O({p_\beta}^2)$	$O({p_\beta}^2)$	529	34.0
seq_j_x	$O({p_\beta}^2)$	$O({p_\beta}^2)$	387	34.0
konishi	$O({p_\beta}^2)$	$O({p_\beta}^2)$	160	35.9

As seen in Table 3, the PFU count varies considerably according to the design, even when the order of resources is the same. As the largest chip of OR2C FPGA is an OR2C40A that contains 900 PFU, comb and seq_i do not fit in a single OR2C FPGA chip. Since the PFU count is strongly related to cost, we use it as a measure of implementation costs in the following discussion.

5.2 Execution Time and Area–Time Product

Another important point is performance. Operating frequency alone is not enough for performance measurement. Even for the same set of input graphs, each design requires a different number of cycles, because the sequence control and logic configuration differ in each design. Therefore, in addition to the estimated operational frequency, we have to count the number of cycles using cycle-accurate simulators.

Please note that the cycle count is strongly dependent on each instance of input graphs. Thus, we have to pay great attention to the nature of the input data (graphs). Here, we take *edge density* to represent the nature of input graphs. 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 trivial that the following relationship holds: $0 \le ed \le 1$.

In this study, we examine four sets of edge densities $(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2)$, and $(0.4, 0.4)$. The average of 100 trials on randomly generated connected graphs is measured for each set of (ed_α, ed_β) using the cycle-accurate simulators of each design.

Through operational frequency and cycle counts,

Table 4 Execution time and AT product.

	$(ed_\alpha, ed_\beta) = (0.2, 0.2)$		$(ed_\alpha, ed_\beta) = (0.2, 0.4)$		$(ed_\alpha, ed_\beta) = (0.4, 0.2)$		$(ed_\alpha, ed_\beta) = (0.4, 0.4)$	
Design	Time [sec.]	AT product	Time [sec.]	AT product	Time [sec.]	AT product	Time [sec.]	AT product
comb	1.78e–02	4.89e+01	1.86e+01	5.14e+04	5.93e–04	1.63e+00	3.89e–02	1.07e+02
seq_i	5.16e–02	9.13e+01	5.99e+01	1.06e+05	1.20e–03	2.12e+00	1.04e–01	1.84e+02
seq_i_j	4.73e–01	2.21e+02	5.75e+02	2.68e+05	8.60e–03	4.02e+00	8.88e–01	4.15e+02
seq_j	6.36e–02	3.71e+01	6.59e+01	3.84e+04	2.11e–03	1.23e+00	1.41e–01	8.23e+01
seq_x	5.83e–02	3.91e+01	6.85e+01	4.59e+04	1.36e–03	9.11e–01	1.16e–01	7.78e+01
seq_i_x	3.68e–01	1.95e+02	5.12e+02	2.71e+05	4.35e–03	2.30e+00	6.31e–01	3.34e+02
seq_j_x	4.91e–01	1.90e+02	5.85e+02	2.26e+05	9.67e–03	3.74e+00	9.37e–01	3.62e+02
konishi	3.15e+00	5.04e+02	1.55e+02	2.49e+04	3.47e–02	5.55e+00	1.57e+00	2.51e+02

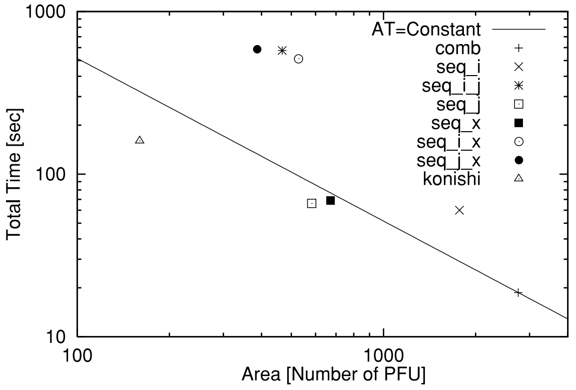

Fig. 6 Area vs. time.

the execution time can be estimated. Table 4 summarizes the execution time and Area–Time product of each design for each (ed_α, ed_β) pair. Here, the AT product is defined as the product of the PFU count and execution time. Cost is regarded as almost proportional to the PFU count (area), and performance is defined by the reciprocal of execution time. Therefore, the AT product is regarded as a measure of the ratio of cost to performance.

Figure 6 displays the relationship between the area and the execution time of each design, where the total execution time is used for $(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2)$, and $(0.4, 0.4)$. The slanted line in the figure is the $AT = constant$ line that corresponds to Ullmann's original design (comb). In Fig. 6, we can see that seq_j, seq_x, and konishi designs are cost-effective solutions to the subgraph isomorphism problem.

The resource requirement of comb and seq_i is the cubic of the number of vertices, while the requirement for the other designs is quadric. The designs seq_i, seq_j, and seq_x are serialized only one-fold, while seq_i_j, seq_i_x, and seq_j_x are serialized two-fold. Considering all these factors, only seq_j and seq_x can be cost-effective alternatives to comb. In fact, these two are slightly more cost-effective than comb in Fig. 6. Their advantages over comb are not so obvious, but could include the following:

- The use of RAM instead of FF made these designs denser.
- Simpler logic made the operational frequency higher.
- Short circuit evaluation saved some cycles in seq_x.

It may seem rather odd that Konishi's circuit is also more cost-effective than Ullmann's original design. One reason is that Konishi's circuit is very small and is a good fit with the OR2C FPGA, which can improve the AT product. On the other hand, its ineffectiveness on pruning can make the AT product worse. As seen in Table 4, it shows an inferior AT product in case of $(ed_\alpha, ed_\beta) = (0.2, 0.2)$, $(0.4, 0.2)$, and $(0.4, 0.4)$. However, Konishi's algorithm works very well in the case of $(ed_\alpha, ed_\beta) = (0.2, 0.4)$, where the average execution time is far longer than in the other three cases. The results in Fig. 6 show the sum of all four cases, in which case $(0.2, 0.4)$ dominates the others.

Why does Konishi's algorithm work so well when the average execution time is great? Note that subgraph isomorphism would very likely be found if $ed_\alpha < ed_\beta$ holds. In such cases, pruning does not work well even with a refinement procedure because there are really many isomorphisms in many subtrees. For an extreme example, assume that G_β is a perfect graph. You will find subgraph isomorphisms in every leaf of the search tree, making pruning impossible. Thus, the hardware resources invested in pruning would be of no use at all. If the execution time does not differ much, Konishi's circuit will be more cost-effective than Ullmann's because it is much smaller.

The lesson to be learned is that one must choose a suitable design, taking account of the nature of the application and input graphs. In some cases, Konishi's algorithm would be preferable because it is cost-effective. If more performance is required, seq_j would be a reasonable selection, although it is more expensive than konishi. Seq_x also seems good, but the logic scale of seq_x is $O(p_\beta{}^2)$ instead of $O(p_\alpha p_\beta)$ of seq_j. As $p_\alpha \leq p_\beta$ generally holds, seq_j would scale better than seq_x for larger graphs.

5.3 Short Circuit Evaluation

Table 5 summarizes the ratio between the savings realized by short circuit evaluation and those of the original cycles without it. As can readily be seen, the effect is

Table 5 The effect of short circuit evaluation.

	(ed_α, ed_β)		
Design	$(0.2, 0.2)$	$(0.4, 0.2)$	$(0.4, 0.4)$
seq_i	0.17%	1.09%	0.75%
seq_i_j	0.17%	1.33%	0.78%
seq_x	0.32%	1.44%	1.19%
seq_i_x	0.22%	1.83%	1.01%

modest and less than 2% of the execution time. We gave up measuring the case $(ed_\alpha, ed_\beta) = (0.2, 0.4)$, because it takes too long to finish the simulations. In any event, we could not expect good results in this case. As the short circuit evaluation only works to make pruning faster, it would scarcely be effective when pruning itself does not work well.

6. Conclusion

Though we designed and evaluated the circuits for small graphs, it is possible to implement custom circuits for larger graphs by using recent large-scale FPGA chips. For example, seq_j involves only the $O(p_\alpha p_\beta)$ hardware resources as shown in Table 3. Quadratic increase of resources is not too much for recent advances in VLSI technology. Custom circuits seem promising for large graphs for which software cannot yield prompt solutions.

As shown in Fig. 6, it is not always cost-effective to adopt a fully-parallel design. Partially sequential designs can be more cost-effective. It is also important to select an adequate design which takes account of the nature of the input data. An example of this is found in Table 4.

Acknowledgment

This work was partially supported by a Grant-in-Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS), as well as grants from the Telecommunications Advancement Foundation (TAF), the Hori Information Science Promotion Foundation, and the Okawa Foundation for Information and Telecommunications. This work was also supported in 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 VLSI Design and Education Center (VDEC), the University of Tokyo.

References

[1] A.V. Aho, R. Sethi, and J.D. Ullman, Compilers - Principles, Techniques, and Tools, ch. 8.4, Addison Wesley, 1987.

[2] C. Cherry and P.K.T. Vaswani, "A new type of computer for problems in propositional logic, with greatly reduced scanning procedures," Information and Control, vol.4, pp.155–168, 1961.

[3] M.R. Garey and D.S. Johnson, Computers and Intractability, Freeman, 1979.

[4] J. Gu, W. Wang, and T.C. Henderson, "A parallel architecture for discrete relaxation algorithm," IEEE Trans. Pattern Anal. Mach. Intell., vol.PAMI-9, no.6, pp.816–831, Nov. 1987.

[5] S. Ichikawa and T. Shimada, "Reconfigurable PCI card for personal computing," Proc. 5th FPGA/PLD Design Conference & Exhibit, Tokyo, pp.269–277, Chugai, 1997. (in Japanese).

[6] 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. (in Japanese).

[7] S. Ichikawa, L. Udorn, and K. Konishi, "Hardware accelerator for subgraph isomorphism problems," Proc. IEEE Symp. Field Programmable Custom Computing Machines (FCCM2000), pp.283–284, IEEE Computer Society, 2000.

[8] Lucent Technologies Inc., ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series FPGAs Data Sheet, 1996.

[9] M.J. Swain and P.R. Cooper, "Parallel hardware for constraint satisfaction," Seventh National Conference on Artificial Intelligence (AAAI '88), pp.2:682–686, Morgan Kaufmann, 1988.

[10] N. Trinajstić, Chemical Graph Theory (2nd Ed.), CRC Press, 1992.

[11] J.R. Ullmann, "An algorithm for subgraph isomorphism," J. ACM, vol.23, no.1, pp.31–42, 1976.

[12] J.R. Ullmann, R.M. Haralick, and L.G. Shapiro, "Computer architecture for solving consistent labelling problems," Computer Journal, vol.28, no.2, pp.105–111, May 1985.

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

Hidemitsu Saito received his B.E. degree in 1998 and M.E. degree in 2000 from the Department of Knowledge-based Information Engineering of the Toyohashi University of Technology. He is presently with Toshiba Corporation.

Lerdtanaseangtham Udorn received his B.E. degree in 1998 and M.E. degree in 2000 from the Department of Knowledge-based Information Engineering of the Toyohashi University of Technology. He is presently with Toyota Caelum, Inc.

Kouji Konishi received his B.E. degree in 1997 and M.E. degree in 1999 from the Department of Knowledge-based Information Engineering of the Toyohashi University of Technology. He is presently with NTT Software Corporation.