Evaluation of Accelerator Designs for Subgraph Isomorphism Problem

Shuichi Ichikawa[1], Hidemitsu Saito[1,2], Lerdtanaseangtham Udorn[1,3], and Kouji Konishi[1,4]

[1] Department of Knowledge-based Information Engineering, Toyohashi University of Technology, Toyohashi 441-8580, Japan
ichikawa@tutkie.tut.ac.jp
[2] Presently with Toshiba Corp.
[3] Presently with Toyota Caelum Inc.
[4] Presently with NTT Software Corp.

Abstract. Many applications can be modeled as subgraph isomorphism problems. 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, and compares them quantitatively from two points of view: cost and performance. An algorithm that is suited for hardware implementation is also proposed. The hardware for the proposed algorithm is much smaller on logic scale, and operates at a higher frequency than Ullmann's design. The prototype accelerator operates at 16.5 MHz on a Lucent ORCA 2C15A, which outperforms the software implementation of Ullmann's algorithm on a 400 MHz Pentium II.

1 Introduction

Many applications, including scene analysis and chemical structural formula databases, are modeled as subgraph isomorphism problems. However, a subgraph isomorphism problem is generally NP-complete [1] and difficult to compute in practical time. There is a strong desire among application developers to shorten the processing time of subgraph isomorphism detection.

Many such hard computation problems are heavily computation intensive. The amount of data is small and communication time is negligible in comparison to computation time. All these properties would seem preferable for acceleration of the process by custom computing machinery.

In this paper, various aspects of hardware accelerators for subgraph isomorphism problems are discussed. Design alternatives are examined mainly from two points of view: cost and performance. A prototype accelerator that is implemented on FPGA is then described.

2 Related Work

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

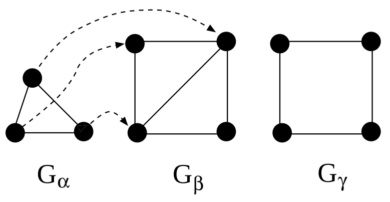

Fig. 1. Subgraph Isomorphism

Ullmann [2] introduced an algorithm for subgraph isomorphism that has been very popular in recent years. Ullmann showed that the *refinement procedure* of his algorithm can be implemented by parallel hardware for faster execution, but neither a detailed discussion nor real implementation was included.

The graph isomorphism problem can be formulated as a constraint satisfaction problem (CSP). Swain and Cooper [3] presented a parallel hardware design for CSP with arc consistency. They mentioned graph matching as a possible application for their circuitry, although their design is not optimized for graph isomorphism. In addition, no real implementation was described in their work.

There are some more studies on custom computing engines for CSP. However, some have been neither implemented nor evaluated [4] [5]. The DRA chip by Gu [6] is a VLSI implementation of a discrete relaxation algorithm (DRA), which is implemented with a 3μ NMOS process. DRA is a general computational technique and is applicable to subgraph homeomorphism problems. However, no more detailed discussion or evaluation of graph problems is found in Gu's work.

In this paper, various implementations of Ullmann's algorithm are examined in detail. Then, a new algorithm is proposed that requires much less in the way of hardware resources. This new algorithm is implemented using a Lucent OR2C FPGA.

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. $G_\alpha = (V_\alpha, E_\alpha)$ is the *subgraph* of $G_\beta = (V_\beta, E_\beta)$, 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 1:1 correspondence between V_α and V_β that preserves adjacency.

A subgraph isomorphism problem is a decision problem to determine whether G_α is isomorphic to a subgraph of G_β. For example, see Figure 1. G_α is isomorphic to a subgraph of G_β. On the other hand, G_γ has no subgraph that is isomorphic to G_α.

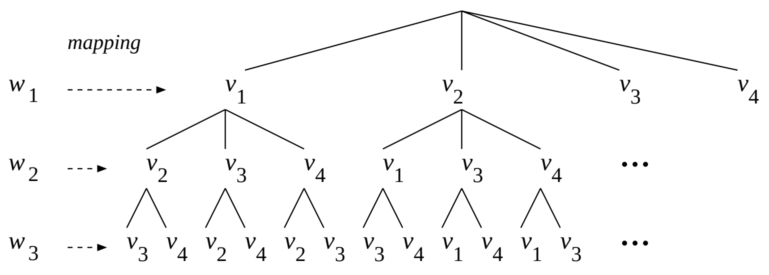
mapping
w_1
w_2
w_3
v_1 v_2 v_3 v_4
v_2 v_3 v_4 v_1 v_3 v_4 ...
v_3 v_4 v_2 v_4 v_2 v_3 v_3 v_4 v_1 v_4 v_1 v_3 ...

Fig. 2. Search Tree

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 a search tree. 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, the adjacency condition is checked by examining the correspondence of the edges from E_α to E_β. Subgraph isomorphism is found when all adjacency is preserved at a leaf.

3.2 Ullmann's Algorithm

The naive tree-search algorithm described in the previous section requires an impractical execution time due to the vast search space. The number of leaves is ${}_{p_\beta}P_{p_\alpha}$, where $p_\alpha = |V_\alpha|$ and $p_\beta = |V_\beta|$. This grows quickly as p_α and p_β grow. Thus, some procedure is required to prune unnecessary sub-trees to shorten the execution time.

The most popular algorithm is the one proposed by Ullmann, which is a smart tree-search algorithm with a *refinement procedure* for pruning [2]. 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 subgraph isomorphism. The central idea 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 in each internal node, but the performance gain is drastic because the expansion of the search tree is repressed effectively. Ullmann formulated the refinement procedure as follows. Let $A = [a_{ij}](1 \le i,j \le p_\alpha)$ and $B = [b_{ij}](1 \le i,j \le p_\beta)$ be the adjacency matrices of G_α and G_β, respectively. Matrix $M = [m_{ij}](1 \le i \le p_\alpha, 1 \le j \le p_\beta)$ is defined as follows: If the mapping from $v_{\alpha i} \in V_\alpha$ to $v_{\beta j} \in V_\beta$ is possible, $m_{ij} = 1$. Otherwise, $m_{ij} = 0$. Then, the following procedure is applied until no element of M is updated. The $r_{xj}(1 \le x \le p_\alpha, 1 \le j \le p_\beta)$ are temporal variables.

$$r_{xj} = (\exists y)(m_{xy} \cdot b_{yj}) \tag{1}$$

$$m_{ij} = m_{ij} \cdot (\forall x)(\bar{a}_{ix} \vee r_{xj}) \tag{2}$$

For more details of the refinement procedure, see Ullmann's paper [2].

Ullmann discussed the parallel hardware implementation of the refinement procedure [2], but it requires $O(p_\alpha {p_\beta}^2)$ hardware resources. This grows rapidly for larger p_α and p_β, and our preliminary results of logic synthesis show that only a small graph can be handled by the state-of-the-art FPGA (See Section 5). Thus, some other way is required for practical implementation.

3.3 Proposed Algorithm

A problem of the refinement procedure is that it checks not only mapped vertices but also not-yet-mapped vertices. This involves huge resources. In this study, we examine a simplified pruning procedure that only handles mapped vertices. See Figure 1 again. At the i-th level of the search tree, only vertices $w_1, ..., w_i (1 \leq i \leq p_\alpha)$ are mapped. Here, we only check the adjacency of 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 is isomorphic to a subgraph of G_β. Our simplified pruning procedure checks this necessary condition. A complete description of this algorithm is found in another paper [8].

The adjacency check of this algorithm is simply realized by referring to the adjacency matrix of G_β, instead of an 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 for acceleration of subgraph isomorphism. On the other hand, the search space reduction with this algorithm is modest, because the ability to prune is inferior to that of Ullmann's refinement procedure. This can make execution time longer. The overall balance is dependent on the mixture of various implementation factors.

In the next section, some design alternatives are presented and examined in detail.

4 Implementation Issues

Ullmann's idea [2] is to calculate $M = [m_{ij}]$ $(1 \leq i \leq p_\alpha, 1 \leq j \leq p_\beta)$ in parallel, as described in Section 3.2. The element circuit to calculate m_{ij} is shown in Figure 3. Let us call this element *sub_comb*. The whole circuit is implemented by the $p_\alpha \times p_\beta$ matrix of sub_comb. Note that the $r_{kj} (1 \leq k \leq p_\alpha)$ of Figure 3 can be shared among $m_{ij} (1 \leq i \leq p_\alpha)$. Therefore, the required hardware resource would be $O(p_\alpha {p_\beta}^2)$. This combinatorial implementation is referred to as *comb* in the following discussion.

Combinatorial implementation is too costly. It is possible to reduce the amount of hardware needed by designing a sequential circuit, in exchange for

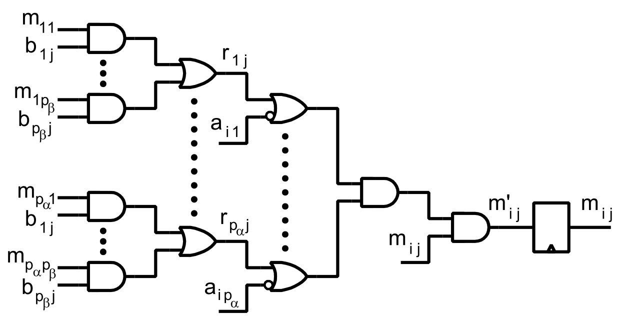

Fig. 3. Combinatorial Circuit for m_{ij}

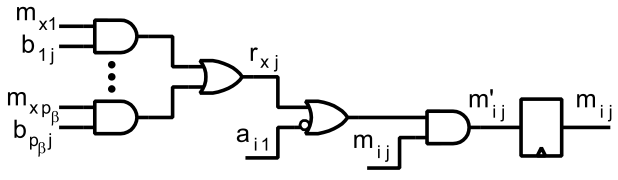

Fig. 4. Sequential Circuit for m_{ij}

increased processing time. The following are three trivial ways to reduce the number of sub_combs.

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 Figure 4. Let us name this circuit *sub_comb_x*. The followings are sequential implementations which use 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 unit of sub_comb_x.

There are pros and cons of sequential circuits. Additional costs can emerge from an input multiplexer and sequence controller in a sequential circuit. Memory cost sometimes decreases because the number of read/write ports can be smaller in a sequential implementation than in a combinatorial implementation.

5 Evaluation

This section presents the various aspects of design, which include Ullmann's original circuit (*comb*), the proposed circuit (*proposed*), and sequential circuits described in the previous section. We do not have enough space to describe each design in detail, so we limit ourselves to the summary shown in Table 1. Please note that the logic scale shown in Table 1 only counts logic gates. In addition to them, $O({p_\beta}^2)$ memory cells are required in each design for adjacency matrices.

Table 1. Design Results

Design	Order(logic)	PFU(total)	Freq(MHz)
comb	$p_\alpha {p_\beta}^2$	2754	22.5
seq_i	$p_\alpha {p_\beta}^2$	1770	27.6
seq_i_j	$p_\alpha p_\beta$	467	34.2
seq_j	$p_\alpha p_\beta$	583	27.9
seq_x	${p_\beta}^2$	671	23.1
seq_i_x	${p_\beta}^2$	529	34.0
seq_j_x	p_β	387	34.0
proposed	$p_\alpha \log p_\beta$	160	35.9

The order of resources is important, because it limits the scalability. However, the real resource count for a certain technology is also important in understanding the constant factor. Even the designs that require the same order of resources can show big differences in real resource count.

To investigate the constant factor, we have to assume some technology or implementation. Here, we adopt the Lucent OR2C series FPGA [7] as a measure. We tuned each design for OR2C FPGA to the extent possible. Though each implementation is not guaranteed to be the best, we believe it is not too far away. In addition, 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 logic of each design is described in VHDL, embedding native OR2C macro library. VHDL description is then processed by Synopsys FPGA Compiler to derive a netlist. The logic synthesis system provides very sophisticated features that we could have utilized. However, we did not use most of its features to exclude the influences from logic synthesis tool. There are still many tips and tricks in logic synthesis, which can affect much to the results. The characteristics of each design becomes clearer by evaluating gate-level designs.

The derived netlist is mapped onto OR2C technology to extract the logic scale and gate delay. The PFU count is summarized in Table 1. PFU (programmable function unit) is a basic logic component of OR2C FPGA [7]. The PFU count in the table includes both logic gates and memory cells. The operating frequency in Table 1 is based on an estimated gate delay. Thus, the routing delay is not counted here.

As seen in Table 1, PFU count varies much according to design, even if the order of resources is the same. As the largest chip of OR2C FPGA is OR2C40A, which contains 900 PFU, comb and seq_i do not fit in a single OR2C FPGA chip. PFU count is strongly related to cost, so we use PFU count as a measure of implementation cost in the following discussion.

Another point is performance. Operating frequency alone is insufficient as a performance measure. Even for the same set of input graphs, each design requires a different number of cycles, because the sequence and configuration are different in each design. Therefore, we have to count the number of cycles using hardware simulators, in addition to estimating the operating frequency.

We have to pay close attention to the nature of input data (graphs), because cycle count is strongly data-dependent. Here, we examine four sets of edge density [1] $(ed_\alpha, ed_\beta) = (0.2, 0.2)$, $(0.2, 0.4)$, $(0.4, 0.2)$, and $(0.4, 0.4)$. The average of 100 trials on randomly generated connected graphs are measured for each set of (ed_α, ed_β).

The execution time can be estimated with operating frequency and cycle counts. Figure 5 summarizes the relationship between the area and the time of each design. The slanting line in the figure is the $AT = constant$ line that corresponds to Ullmann's original design. Cost is regarded to be almost proportional to PFU count (area), and performance is defined by the reciprocal of the execution time. Therefore, the AT product is regarded as a measure of the ratio of cost to performance. Figure 5 shows that *seq_j, seq_x*, and *proposed* designs turn out to be cost-effective solutions to the subgraph isomorphism problem.

It seems a little odd that the proposed algorithm is very cost-effective. The first reason is that it is very small and fits well to OR2C FPGA. This makes the AT product better. Conversely, its ineffectiveness in pruning can make the AT product worse. In fact, a worse AT product is seen in the cases of $(ed_\alpha, ed_\beta) = (0.2, 0.2)$, $(0.4, 0.2)$, and $(0.4, 0.4)$. However, the proposed algorithm works very well in the case of $(ed_\alpha, ed_\beta) = (0.2, 0.4)$, where the average execution time is far bigger than that of other three cases. The result in Figure 5 is the sum of all four cases, so the case $(0.2, 0.4)$ dominates.

Why does the proposed algorithm do well when the average execution time is large? Notice that subgraph isomorphism would be very likely if $ed_\alpha < ed_\beta$ holds. In such cases, pruning does not work well even with a refinement procedure, because there are numerous isomorphisms in many subtrees. If the execution time does not differ as much, the proposed algorithm can be more cost-effective, because it is smaller than Ullmann's circuitry.

The lesson is that we have to choose a suitable design, considering the nature of the application and input graphs. In some cases, the proposed algorithm would

[1] Let us consider a graph with p vertices and q edges. The 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: $0 \leq ed \leq 1$.

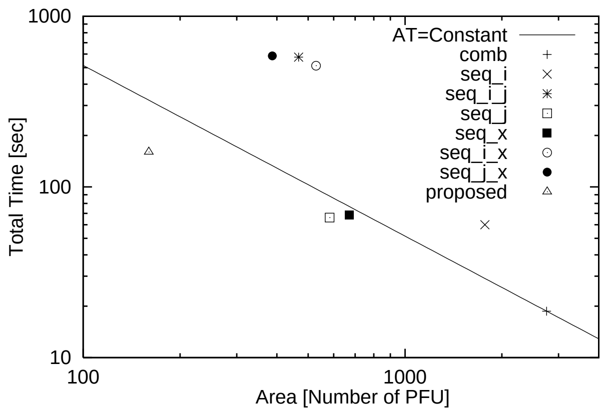

Fig. 5. Area vs. Time

be very effective. In more general cases, seq_j would be reasonable. Seq_x also seems good, but the logic scale of seq_x would be $O({p_\beta}^2)$ instead of the $O(p_\alpha p_\beta)$ of seq_j. As $p_\alpha \leq p_\beta$ holds generally and $p_\alpha \ll p_\beta$ holds in some applications, seq_j would scale better than seq_x for bigger graphs.

6 Prototype Hardware

In this section, the FPGA implementation of the proposed algorithm is outlined. We chose this design mainly because it is the smallest and simplest. The details of this prototype are described in other papers [8] [9].

The prototype was implemented on an OPERL board [10], which is a run-time reconfigurable PCI card with two Lucent OR2C FPGAs [7] (see Figure 6). USER FPGA (OR2C15A) contains an application circuit that is programmed and accessed from the host computer via PCI bus. Another is PCI FPGA (OR2C15A), which contains PCI interface circuitry and a run-time reconfiguration controller for USER FPGA. The host computer is a personal computer with AMD K6-III (400 MHz) and FreeBSD 2.2.8R. USER FPGA can be programmed in less than 5 ms. The application program can transfer data using system call (read/write/mmap) or I/O instructions.

The key to utilizing SRAM-based FPGA is the extensive use of mapping RAM in logic. Subgraph isomorphism problems naturally fit this scheme, as it is implemented by a tree-search with adjacency matrices. We implemented a unit that can handle up to $(p_\alpha, p_\beta) = (15, 15)$, which fits well to the basic component of OR2C FPGA (16×4 bit SRAM). This unit operates at 16.5 MHz, which is half of the PCI clock. The unit could have been pipelined for 33 MHz operation to derive twice the performance, but we chose to make things simple for this prototype.

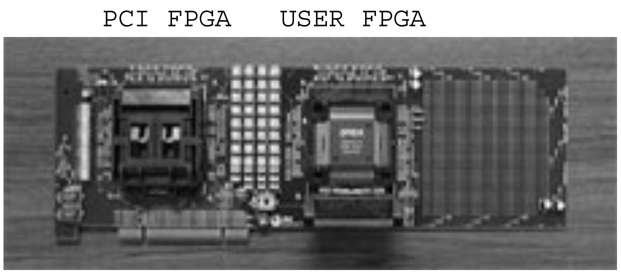

Fig. 6. OPERL board

Even this simple prototype outperforms an off-the-shelf microprocessor. An OR2C15A chip that contains two units of our proposed algorithm shows about 20 times better performance than the software implementation of Ullmann's algorithm on a 400 MHz Pentium II processor. For the performance details of this prototype, see another paper [8].

The performance can be boosted easily by (1) pipelining hardware, (2) implementing several units that work in parallel, and (3) using a larger FPGA chip. For example, the largest chip of OR2C FPGA (OR2C40A) contains 900 PFUs. Using OR2C40A, four independent units of the proposed algorithm can be implemented on one chip. In this case, the performance gain would scale up four times of a single unit.

7 Conclusion

Though the prototype hardware can handle only small graphs, it is easy to implement an accelerator for larger graphs. For example, seq_j requires only an $O(p_\alpha p_\beta)$ logic gates. This is not so much for a VLSI implementation. The real problem is the explosion of execution time. Remember that this problem is NP-complete. Pruning alleviates the problem, but never solves it.

The key is to design an application specific circuit. In this paper, we treated algorithms and designs for general subgraph isomorphism problems. However, as mentioned at the end of Section 5, the algorithm and architecture should be customized for the application in order to maximize performance, considering the nature of the data.

Take, for example, the case of a chemical structural database. In such an application, the data structure is not a general graph. Each vertex has attributes to represent its class: H, C, benzene ring, etc. The edges can also have attributes: single bond, double bond, etc. Such attributes help to make execution time shorter, if such information is used for pruning.

It is not always cost-effective to design such special circuitry, but the evolution of FPGA and logic synthesis will make it feasible in the near future.

Acknowledgment

This work was partially supported by the Hori Information Science Promotion Foundation and the Ministry of Education, Science, Sports and Culture of Japan.

References

1. M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.
2. J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, Vol. 23, No. 1, pp. 31–42, 1976.
3. M. J. Swain and P. R. Cooper. Parallel hardware for constraint satisfaction. In *Seventh National Conference on Artificial Intelligence (AAAI '88)*, pp. 2:682–686. Morgan Kaufmann, 1988.
4. 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.
5. 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.
6. J. Gu, W. Wang, and T. C. Henderson. A parallel architecture for discrete relaxation algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No. 6, pp. 816–831, Nov. 1987.
7. Lucent Technologies Inc. *ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series FPGAs Data Sheet*, 1996.
8. S. Ichikawa, L. Udorn, and K. Konishi. An FPGA-based implementation of subgraph isomorphism algorithm. *IPSJ Transactions on High Performance Computing Systems*, 2000 (to appear, in Japanese).
9. S. Ichikawa, L. Udorn, and K. Konishi. Hardware accelerator for subgraph isomorphism problems. In *Proc. IEEE Symp. FPGAs for Custom Computing Machines (FCCM '00).* IEEE Computer Society, 2000 (to appear as extended abstract).
10. S. Ichikawa and T. Shimada. Reconfigurable PCI card for personal computing. In *Proceedings of the 5th FPGA/PLD Design Conference & Exhibit*, pp. 269–277, Tokyo, 1997. Chugai (in Japanese).

This article was processed using the LaTeX macro package with LLNCS style.