IEICE TRANS. INF. & SYST., VOL.E86–D, NO.5 MAY 2003

PAPER *Special Issue on Reconfigurable Computing*

Data Dependent Circuit for Subgraph Isomorphism Problem*

Shuichi ICHIKAWA†, ***Regular Member and*** **Shoji YAMAMOTO**†, ***Student Member***

SUMMARY Although the subgraph isomorphism problem has various important applications, it is generally NP-complete and difficult to solve. Though a custom computing circuit can reduce the execution time substantially, it requires considerable hardware resources and is inapplicable to large problems. This paper examines the feasibility of data dependent designs, which are particularly suitable to a Field Programmable Gate Array (FPGA). The data dependent approach drastically reduces hardware requirements. For graphs of 32 vertices, the average logic scale of data dependent circuits is only 5% of the corresponding data independent circuit. The data dependent circuit is estimated to be maximally 460 times faster than the software. Even if the circuit generation time is included, a data dependent circuit is estimated to be 2.04 times faster than software for graphs of 32 vertices. The performance gain would increase for larger graphs.

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

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. For example, see Fig. 1. 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, including scene analysis in computer vision and search operation in chemical structural formula databases. However, the subgraph isomorphism problem is generally NP-complete [6] and computationally difficult to solve.

For a practical solution, several algorithms have been proposed. Ullmann [14] proposed a depth-first search algorithm with a smart pruning procedure (*refinement procedure*), which is now the most popular and frequently used algorithm for this problem. In his paper [14], Ullmann also pointed out that his refinement procedure can be implemented with asynchronous hardware.

A custom circuit for a subgraph isomorphism problem would be a promising way to solve the problem quickly. However, the Ullmann's hardware requires too many resources for practical applications. The purpose of this study is to examine the data dependent designs of Ullmann's hardware, which drastically reduces the hardware resources.

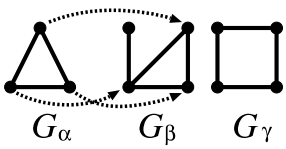


Fig. 1 Subgraph isomorphism.

If any input of the logic circuit turns out to be constant, the circuit can be reduced. In the same manner, we can shrink the logic scale of Ullmann's circuit by fixing the input graph instance. The obvious drawback of this approach is that the circuit is not reusable. It would be impractical to implement such data dependent circuits in usual ASIC technologies. Instead, a reconfigurable device technology (e.g., FPGA) is essential to our approach.

2. Custom Circuit for Subgraph Isomorphism Problems

Custom computing circuitry is a sensible choice for a cost-effective acceleration of solutions to subgraph isomorphism problems. This section briefly outlines some preceding studies and their problems. More references to related works are found in our earlier papers [9], [10].

2.1 Ullmann Circuit

As described in Sect. 1, Ullmann's refinement procedure can be implemented with a logic circuit [14]. Let us consider the Ullmann circuit in more detail.

Let p_α and p_β be the number of vertices of graph G_α and G_β, respectively. The matrices A and B are the adjacency matrices of G_α and G_β. An adjacency matrix is one of the data structures used to represent graphs [7]. The matrix M is used for temporary variables in the refinement procedure.

$$A = [a_{ij}] \qquad (1 \le i, j \le p_\alpha), \tag{1}$$

$$B = [b_{ij}] \qquad (1 \le i, j \le p_\beta), \tag{2}$$

$$M = [m_{ij}] \qquad (1 \le i \le p_\alpha,\ 1 \le j \le p_\beta). \tag{3}$$

Please note that all these matrices are implemented by

Manuscript received August 24, 2002.

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

*This work appeared in outline form as an extended abstract in the Proceedings of FPL2002, pp.1068–1071, LNCS 2438, Springer (2002).

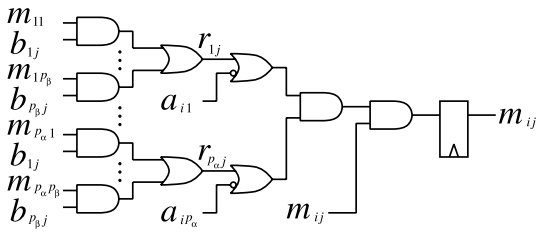


Fig. 2 Element circuit for refinement procedure.

memory modules or flipflops.

Figure 2 illustrates the element circuit to calculate m_{ij}, as proposed by Ullmann [14]. The whole circuit for the refinement procedure includes a $p_\alpha \times p_\beta$ array of this element circuit. Sharing the common subcircuits $r_{kj}\,(1 \le k \le p_\alpha)$ among $m_{ij}\,(1 \le i \le p_\alpha)$, the total hardware resource for the refinement procedure is reduced to $O(p_\alpha\, {p_\beta}^2)$, which is still very large. The Ullmann circuit also requires memories and sequencers, but they are negligible compared to the resources needed for the refinement procedure.

Experimental evaluations with a Field Programmable Gate Array (FPGA) revealed that the original Ullmann circuit requires too many logic gates, most of which are used for the refinement procedure. As far as hardware is concerned, we must drastically reduce the number of logic gates [9].

2.2 Partially Sequential Ullmann Circuit

Ichikawa et al. [9] examined the modified Ullmann designs that adopt partially sequential implementations of the refinement procedure circuitry to reduce the logic scale. Although some of the designs proved to be more cost-effective than the Ullmann's original circuit, they are still expensive.

2.3 Konishi Circuit

Ichikawa et al. [10] proposed a new algorithm that adopts a pruning condition simpler than that of Ullmann's refinement procedure. Called "Konishi's algorithm" after its inventor, it requires *much less* hardware because its pruning procedure is so much simpler than Ullmann's. Though this algorithm exhibits performance inferior to Ullmann's algorithm because of its reduced pruning ability, it is still far more cost-effective than either the original Ullmann or the partially sequential Ullmann circuits [9].

The hardware accelerator of Konishi's algorithm was implemented and evaluated on a FPGA board [10], [11]. As this circuit is very small, two equivalent units could fit into a single Lucent 2C15A FPGA, which outperformed the software implementation on an off-the-shelf microprocessor.

2.4 Data Dependent Ullmann Circuit

To solve larger problems by custom circuitry, both logic scale and matching efficiency are important. It is desirable, if possible, to reduce the hardware requirement of the Ullmann's algorithm. This paper examines data dependent implementations of the Ullmann circuit that drastically reduces the number of logic gates.

Generally, if any input of the logic circuit turns out to be constant, the circuit can be reduced. This reduction can be applied recursively. Therefore, a data dependent circuit can be smaller than a data *independent* circuit with the same function. Smaller circuits usually work at a higher frequency, and thus can be faster. A data dependent circuit is more cost-effective, because it is smaller and faster†.

A data dependent design is well suited to FPGA implementation due to its nature. On the other hand, the evident drawback is that the circuit is not reusable. Since a custom circuit must be generated for each input instance, the processing time for logic synthesis, mapping, placement and routing is also important along with the execution time itself.

For computationally difficult problems such as subgraph isomorphism problems, the execution time for larger problems increases very quickly. As we will show later, the circuit generation time could be inferior or even negligible compared to the execution time. This is one of the reasons why the authors chose a data dependent approach to computationally difficult problems.

3. Data Dependent Circuit for Graph Problems

Although there have been no studies on the data dependent approach to solving subgraph isomorphism problems, some other graph problems were examined in prior studies. This section briefly describes these studies.

Chakradhar and Agrawal [3] showed the feasibility of a VLSI solution to an Independent Set Problem (ISP). Their approach is based on the transformation of the graph to a logic circuit. Their study is relevant here because (1) ISP is an NP-complete problem, and (2) their VLSI solution is data dependent. However, their approach is not general but limited to special cases. In addition, neither implementation nor evaluation was discussed.

Babb, Frank and Agarwal [1] introduced Dynamic Computation Structures (DCS) to solve closed semiring problems (e.g., Transitive Closure, Shortest Path). They mapped graph instances to DCS on FPGA in a

†This kind of technique is not limited to hardware. In the software area, it is well known as *software specialization* or *partial evaluation* [4].

straightforward manner, thus implementing a data dependent circuit for solving graph problems. However, their target applications were so simple that they are solvable in polynomial time. It is not realistic to build custom circuits for such simple problems since software is fast enough to handle them. Another serious drawback is that the circuit generation takes several hours.

Dandalis, Mei and Prasanna [5] presented domain-specific mapping for a simple graph problem (single-source shortest path). Though a data dependent custom circuit is powerful and versatile, long circuit generation time offsets the advantages of custom computing. They concentrated on reducing circuit generation time, and their mapping system cut it from 4 hours to 0.1 second. The speed-up ratio to software implementation was asymptotically 3.75, considering both mapping time and hardware execution time. The problem is that their application is polynomially solvable and overly simple for custom computing.

Huelsbergen [8] presented a Dynamic Graph Processor (DGP) for reconfigurable hardware, in which graph data structures are represented as logic circuits on FPGA. DGP is interesting in that vertices and edges may be dynamically inserted/deleted at low cost. His target applications were simple and easy to solve; e.g., reachability, transitive closure, shortest unit path, connected-component identification, and cycle detection. The problem with DGP is that it mainly exploits edge parallelism and does not always fit every problem.

Bingham and Serra [2] presented a data dependent circuit for Hamiltonian Cycle problem (HC), which is a well-known NP-complete problem. Their circuit was shown to be competitive with software implementation, considering the time for mapping, placement, and routing. Their design is quite straightforward. They map vertices to connected finite state machines that emulate search tree traversal. Their approach is interesting in that its framework is generally applicable to hard computation problems. On the other hand, their algorithm seems somewhat naive and unrealistic. Their paper, for example, presents no pruning technique, which is always a critical factor in performance. Their evaluation is also not thorough, and no comparison is made with other techniques.

In the present paper, a data dependent circuit for a subgraph isomorphism problem is examined. This problem is NP-complete and requires a long execution time. For such problems, a custom computing approach is suited and hopeful. We adopt Ullmann's algorithm, which is popular and frequently used in real application programs.

4. Design Approaches

In this section, a couple of design alternatives are described and examined. Quantitative evaluations of these designs are shown in Sect. 5.

4.1 Original Ullmann Circuit

First of all, the original Ullmann circuit should be evaluated as a basis for further evaluation. The source code is written in VHDL, which was taken from the previous project [9]. The details of this design are found in another paper [13]. This design is denoted as "INDEP" in this paper, because it is independent of input graph instances.

4.2 Data Dependent Designs

The fundamental advantage of a data dependent approach is that logic gates can be reduced. See the examples in Fig. 3. If any input is fixed to a constant, the logic gate is reduced, and one of the inputs propagates to the output. This process can be applied recursively.

If the input graph G_α is fixed, the elements a_{ij} of adjacency matrix A become constants. Then, we can reduce the memory or flipflops to store A and the logic gates connected to a_{ij} in each element circuit (Fig. 2). Practically speaking, we can simply replace a_{ij} in the VHDL source code with the corresponding constants. Once any a_{ij} is replaced by a constant, unnecessary logic gates are automatically reduced at the logic synthesis stage. This design approach is denoted by "ALPHA" in the following discussion.

In the same manner, "BETA" denotes a design derived by fixing the input graph G_β to a specific instance. "BOTH0" denotes the circuit that has both G_α and G_β fixed.

Please note that the circuit generated by ALPHA or BETA is reusable to some extent, because one of the input graphs is still variable. BOTH0 generates an input-specific circuit, which is much smaller than ALPHA or BETA, though it is not reusable.

4.3 Improvements on Circuit Generation

It seems both easy and simple to commit logic reduction to logic synthesis system, since such reduction is an essential feature of logic synthesis. However, no synthesis tool can answer every need. It is a serious issue that logic synthesis consumes too much computational resource and execution time, in particular for large problems.

What we need here is a simple and problem-specific optimization. This could be realized by a simple preprocessing, which might improve the situations. In this study, we propose two simple preprocessing techniques.

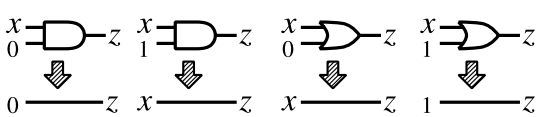


Fig. 3 Reduction of logic gates.

Table 1 Evaluation environment.

Item	Note
VHDL Source Code Generation	Duron 800 MHz, Memory 512 MB, Windows2000 SP2 Written in C, compiled with gcc 2.95.3-4 on Cygwin
Synthesis	Duron 800 MHz, Memory 512 MB, Windows2000 SP2 Synopsys FPGA Compiler II (2000.11 FC3.5.2) (used with options – Optimize: Area, Effort: High)
Mapping	Pentium II 450 MHz, Memory 512 MB, WindowsNT 4.0 SP6a Lucent ORCA Foundry 9.4a (for OR2CxxA architecture)
Software Implementation	Pentium III 600 MHz, Memory 256 MB, Red Hat Linux 6.2 Written in C, compiled with gcc/egcs-1.1.2-30

Since we already have to generate the VHDL source code of a data dependent circuit according to input graphs, it is easy enough to add some reduction features at this stage. Such preprocessing actually does no harm because the generated VHDL source code is optimized again by the logic synthesis tool.

The first thing we need to do is to reduce the AND and OR gates that are directly connected to a_{ij} or b_{ij}. The outputs of the corresponding gates are replaced with constants by the VHDL source generator, as shown in Fig. 3. We added this feature to BOTH0, creating a new method denoted by "BOTH1" in the following discussion. BOTH1 does not handle recursive reduction, which is left for the logic synthesis stage.

We can further reduce gates by exploiting the nature of Ullmann's algorithm [14]. As you can readily see in Fig. 2, the value of m_{ij} can change only from 1 to 0 during the refinement procedure. Thus, if the initial value of m_{ij} is 0, this m_{ij} remains permanently 0. As the initial values of m_{ij} can be easily determined from the input graph instances (G_α and G_β), we can thus reduce the AND and OR gates connected to such m_{ij}, together with the corresponding element circuit and the flipflop to store m_{ij}. We added this feature to BOTH1, and denote this improved method as "BOTH2."

5. Evaluation

This section describes the evaluation results of the design approaches described in Sect. 4. Each of the results shown in this section is the average of 50 pairs of G_α and G_β, which are randomly generated trees. Trees were chosen for inputs because they are the sparsest connected graphs. To reduce simulation parameters, $p_\alpha = p_\beta$ is also assumed in this study. It would be interesting to investigate the cases of dense graphs, disconnected graphs, and the graphs of $p_\alpha < p_\beta$. However, since they are beyond the scope of this feasibility study, we leave them to future studies.

The evaluation environment is summarized in Table 1. First, each input graph pair is processed by a VHDL source-code generator. At this stage, the reduction techniques are applied as described in Sect. 4. Second, the derived VHDL code is passed to the logic synthesis system to generate a netlist. Then, it is mapped onto FPGA. We did not perform the placement and routing (P&R) in this feasibility study.

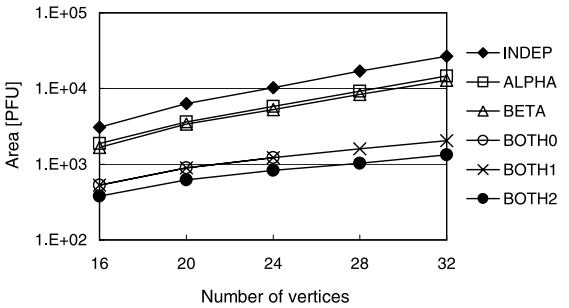


Fig. 4 Logic scale.

We decided to leave P&R for future studies for the following reasons. (1) The execution time of P&R is strongly dependent on implementation constraints such as required operational frequency and I/O-pin limitations. Since it is difficult to discuss P&R without a loss of generality, we decided to keep the focus on the general feasibility of this approach. (2) P&R is very time consuming. As described above, we examined 50 graph sets for each set of parameters. Performing thousands of P&R trials were simply not feasible.

5.1 Logic Scale

Figure 4 displays the average logic scale derived from the mapping reports of 50 input graph pairs for each design approach. The logic scale is measured by the number of PFUs (programmable function units) of Lucent OR2C FPGA [12].

ALPHA and BETA require almost the same amount of resources, which is about half of INDEP. The logic scale of BOTH0 is only 17–12% of that of INDEP for $16 \leq p_\alpha, p_\beta \leq 24$. For $p_\alpha, p_\beta \geq 28$, the results of BOTH0 are unavailable since it takes too much time for the synthesis tool to finish, as shown in Fig. 6. BOTH1 can generate circuits much quicker than BOTH0, while keeping the quality of the circuit practically equal to BOTH0. The circuit of BOTH2 is even smaller and faster than BOTH1. For graphs of 32 vertices, BOTH2 requires only 5% of the resources needed by INDEP.

The reduction in resources is drastic in fully data dependent designs (BOTH0, BOTH1, BOTH2) compared to partially data dependent designs (ALPHA,

Table 2 Detailed logic scale of INDEP (PFU).

	Number of vertices				
	16	20	24	28	32
R. P.	2562	5601	9216	15680	24575
Memory	256	502	722	982	1280
Etc.	253	250	272	286	961
Total	3071	6353	10210	16948	26816

Table 3 Detailed logic scale of BOTH2 (PFU).

	Number of vertices				
	16	20	24	28	32
R. P.	45.14	72.68	104.80	143.82	188.34
Memory	91.46	212.32	300.34	405.74	528.14
Etc.	243.84	338.96	422.56	480.00	614.50
Total	380.44	623.96	827.70	1029.56	1330.98

Table 4 Maximum operational frequency (MHz).

Design	Number of vertices				
	16	20	24	28	32
INDEP	27.8	27.9	n/a	n/a	n/a
ALPHA	31.9	31.2	n/a	n/a	n/a
BETA	32.2	32.3	n/a	n/a	n/a
BOTH0	37.0	35.4	35.3	n/a	n/a
BOTH1	37.0	35.4	35.3	31.7	31.7
BOTH2	44.0	39.5	36.7	36.4	35.7

BETA). Therefore, we no longer include ALPHA and BETA in the following discussion.

Tables 2 and 3 show the details of INDEP and BOTH2 designs, respectively. In these tables, "R. P." stands for the circuit for the refinement procedure. The item "Memory" refers to the memory resource for adjacency matrices and other temporal storage. All other circuits are included under "Etc." As seen in Table 2, the resource for the refinement procedure is dominant in the INDEP design, whereas it occupies only 12–14% of resources in the BOTH2 design. This fact clearly shows the advantages of BOTH2 over INDEP, particularly for larger graphs.

5.2 Operational Frequency

Average operational frequencies are listed in Table 4. These results are derived from the reports of technology mapping software. The operational frequencies of INDEP, ALPHA, and BETA for more than 24 vertices are unavailable because they require so many PFUs that the mapping software aborts them. The results of BOTH0 for 28 and 32 vertices are also unavailable, because the synthesis phase takes too much time to finish as seen in Fig. 6.

A data dependent approach reduces the depth of combinatorial logic by removing unnecessary gates. Thus, a smaller circuit naturally operates at higher frequency as seen in Table 4. One must recall that, while reducing logic gates, we did not change the sequence control of the Ullmann circuit. This means that the cycle count required to finish the algorithm is identical to the original INDEP design, i.e., the performance of circuitry is proportional to its operational frequency. For example, we can estimate that the BOTH2 design is about 1.42 times faster than the INDEP design in a case of 20 vertices.

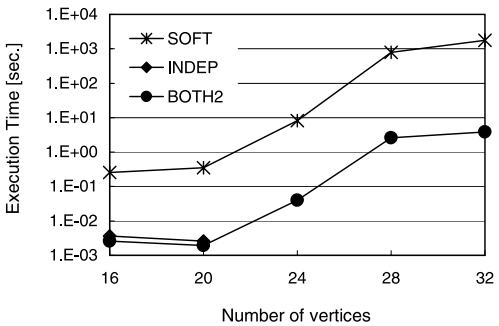


Fig. 5 Execution time of subgraph isomorphism detection.

As the wiring delay is not counted in Table 4, the real operational frequency after P&R might be much lower. Nevertheless, a smaller circuit would not lose its advantages over a larger circuit because the latter is a much more difficult target for P&R. Taking all of this into consideration, the performance gain of BOTH2 over INDEP could be greater than the estimation derived from mapping reports.

5.3 Execution Time

As a basis for evaluation, the software implementation of Ullmann's algorithm was evaluated by a Pentium III processor with the same data sets. The execution environment is detailed in Table 1. The results are designated as "SOFT" in Fig. 5 and Table 5.

Figure 5 shows the execution times of SOFT, INDEP, and BOTH2. The execution times of INDEP and BOTH2 were estimated from (1) the operational frequency derived from mapping reports and (2) the cycle count derived from the cycle-level simulator.

INDEP is not possible for large graphs. Even for small graphs, INDEP is slightly slower than BOTH2 since it operates at a lower frequency. In contrast, BOTH2 is feasible even for large graphs, and is much faster than software.

Table 5 summarizes the acceleration ratio of BOTH2, that is, the ratio of the execution time of SOFT to the execution time of BOTH2. As readily seen, the performance gain grows from 99–460 for 16–32 vertices. This gain emerges from the parallelism in the circuit for refinement procedure. Thus, this gain would increase for graphs with more vertices because there is a $p_\alpha\, p_\beta$ parallelism in the refinement procedure circuit.

Table 5 Execution time and acceleration ratio.

Design	Number of vertices				
	16	20	24	28	32
SOFT [sec]	2.532e–01	3.514e–01	8.111e+00	7.903e+02	1.764e+03
BOTH2 [sec]	2.564e–03	1.921e–03	4.033e–02	2.573e+00	3.838e+00
SOFT/BOTH2	98.8	182.9	201.1	307.2	459.6

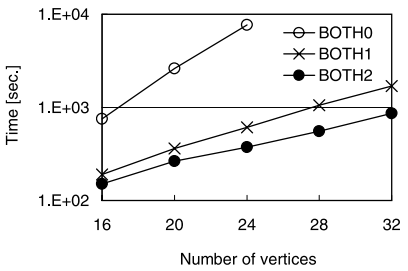


Fig. 6 Circuit generation time.

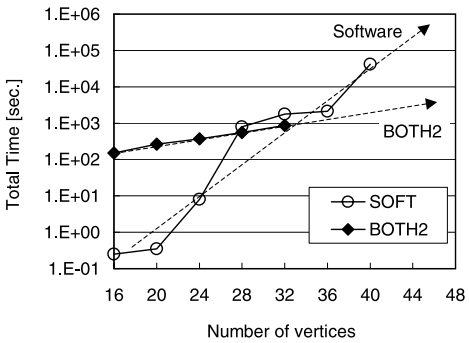


Fig. 7 Overall time for SOFT and BOTH2.

Table 6 Detailed generation time of BOTH2 (sec).

	Number of vertices				
	16	20	24	28	32
VHDL Gen.	0.56	0.62	0.67	0.74	0.83
Synthesis	98.28	148.56	211.71	314.38	503.27
Mapping	51.68	114.68	161.60	240.34	357.42
Total	150.52	263.86	373.98	555.46	861.52

5.4 Circuit Generation Time

As a data dependent circuit is generated for each input graph, the circuit generation time is very important. Figure 6 displays the advantages of preprocessing.

The circuit generation time of BOTH0 is too large and increases very quickly for larger graphs. However, BOTH1 incorporates a simple preprocessing step, which consequently alleviates the burden of logic synthesis and drastically reduces the time for synthesis. After all, the circuit generation time of BOTH1 is only 7.9% of BOTH0 for 24 vertices. The preprocessing of BOTH2 also reduces both the gates and the synthesis time. BOTH2 only requires 4.9% of the time needed by BOTH0 for circuit generation (24 vertices).

Table 6 shows the details of the circuit generation time of BOTH2. The synthesis times and the mapping times are almost comparable in BOTH2, while the time for VHDL source-code generation (incl. preprocessing) is negligible. Considering that the synthesis was a major bottleneck in BOTH0, the improvement of BOTH2 in circuit generation time is remarkable. Now that the synthesis time is not a severe bottleneck in BOTH2, we would have to focus on how to accelerate mapping (and P&R) for further improvement.

6. Conclusion

Figure 7 illustrates the total time for SOFT and BOTH2. Software execution time (SOFT) increases very quickly; e.g., the time for 32 vertices is 6900 times longer than for 16 vertices. On the other hand, the total time of BOTH2 accumulates very slowly; e.g., the time for 32 vertices is only 5.7 times longer than for 16 vertices. According to our evaluation, BOTH2 is already 2.04 times faster than SOFT in the case of 32 vertices. The advantage of BOTH2 will be greater for larger graphs, as shown in Fig. 7.

Though P&R was not performed in this study, we can show that it does not materially affect our conclusion. The important thing is that the execution time is not a bottleneck in BOTH2. Let us examine the case of 32 vertices as an example. From Tables 5 and 6, we can clearly see that the circuit generation time (861.5 sec.) is far greater than the execution time (3.84 sec.) itself. Even if the circuit were to operate 10 times slower than expected, the total time is hardly affected. This means that we can perform P&R under a very loose constraint, which consequently keeps P&R time at a modest level. Even if placement and routing are taken into consideration, the data dependent approach continues to look promising, especially for large graphs.

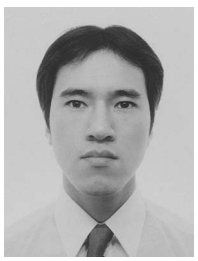
Acknowledgments

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. 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] 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] 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.
[4] C. Consel and O. Danvy, "Tutorial notes on partial evaluation," Proc. 20th ACM Symp. on Principles of Programming Language, pp.493–501, ACM, 1993.
[5] 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.
[6] M.R. Garey and D.S. Johnson, Computers and Intractability, Freeman, 1979.
[7] J. Gross and J. Yellen, Graph Theory and Its Applications, CRC Press, 1998.
[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, "Evaluation of accelerator designs for subgraph isomorphism problem," Proc. 10th Int'l Conf. Field-Programmable Logic and Applications (FPL2000), LNCS1896, pp.729–738, Springer, 2000.
[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, 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.
[12] Lucent Technologies Inc., ORCA OR2CxxA (5.0 V) and OR2TxxA (3.3 V) Series FPGAs Data Sheet, 1996.
[13] H. Saito, A study on hardware implementation of Ullmann's algorithm, Master's Thesis, Dept. Knowledge-based Information Engineering, Toyohashi University of Technology, 2000. .
[14] J.R. Ullmann, "An algorithm for subgraph isomorphism," J. ACM, vol.23, no.1, pp.31–42, 1976.

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 machinery. He is a member of ACM, IEEE, and IPSJ.



Shoji Yamamoto received his B.E. degree in 2002 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.