Hardware Accelerator for Subgraph Isomorphism Problems

Shuichi Ichikawa, Lerdtanaseangtham Udorn, and Kouji Konishi
Dept. Knowledge-based Information Engineering, Toyohashi University of Technology
1-1 Hibari-gaoka, Tempaku, Toyohashi 441-8580, Japan
E-mail: `ichikawa@tutkie.tut.ac.jp`

Abstract

Many applications can be modeled as subgraph isomorphism problems, which are generally NP-complete. This paper presents an algorithm that is suited for hardware implementation. The prototype accelerator that operates at 16.5 MHz on a Lucent ORCA 2C15A FPGA outperforms the software implementation of Ullmann's algorithm on a 400 MHz Pentium II by 10 times in the best case.

1 Subgraph Isomorphism

Many applications, including scene analysis and chemical structural formula databases, are modeled as subgraph isomorphism problems. However, subgraph isomorphism is generally NP-complete [1] and difficult to compute within a reasonable time.

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.

As is easily seen, subgraph isomorphism can be solved by brute-force enumeration with a depth-first tree-search algorithm. Let p_α, p_β, q_α, and q_β be $|V_\alpha|$, $|V_\beta|$, $|E_\alpha|$, and $|E_\beta|$, respectively. Consider a search tree whose depth is p_α. Each stage corresponds to each vertex $v_\alpha \in V_\alpha$. At each stage of the tree, possible mappings of v_α to $\exists\, v_\beta \in V_\beta$ branch out. At each leaf, the correspondence of edges in E_α to E_β is checked. If adjacency is preserved in this mapping, subgraph isomorphism is found.

2 Algorithm

As a naive tree-search algorithm requires an impractical execution time for vast search space, some procedure is required to prune unnecessary sub-trees. Ullmann proposed a smart tree-search algorithm with a *refinement procedure* for pruning [2].

It is a necessary condition for subgraph isomorphism that adjacent vertices in G_α are mapped to adjacent vertices in G_β. The refinement procedure checks this necessary condition recursively at each node of the search tree. If the condition is not satisfied, the subgraph isomorphism is never found under this sub-tree. The refinement procedure involves some overhead, 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 variables $r_{xj}(1 \le x \le p_\alpha, 1 \le j \le p_\beta)$ are temporal.

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

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

Ullmann pointed out that m_{ij} can be updated in parallel by using parallel hardware [2], though such implementation incurs $O(p_\alpha {p_\beta}^2)$ logic gates and $O({p_\beta}^2)$ memory cells. This grows rapidly for bigger p_α and p_β. Hence, some other way of pruning is required for practical implementation.

The problem with the refinement procedure is that it checks the necessary condition on all vertices. This requires a huge amount of resources to update m_{ij} in parallel. Therefore, in this study, we propose to use only mapped vertices for pruning. More precisely, the adjacency among $v_{\alpha 1}, ..., v_{\alpha i}$ is checked at the i-th level of the search tree. The idea is that it is necessary for any subgraph of G_α to be isomorphic to a subgraph of G_β for G_α to be isomorphic to a subgraph of G_β. This algorithm requires only $O(p_\alpha \log p_\beta)$ gates and $O({p_\beta}^2)$ memory cells, though the ability of pruning decreases accordingly.

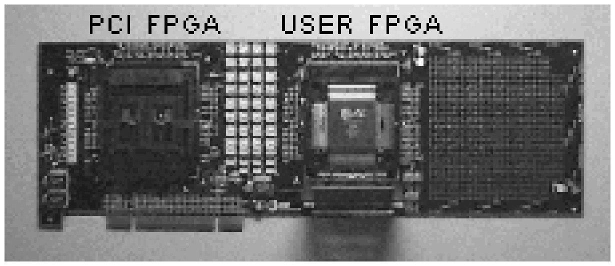

Figure 1: OPERL board

3 Implementation

The proposed algorithm was implemented on an OPERL board [3], which is a run-time reconfigurable PCI card with two Lucent OR2C FPGA chips (Figure 1). USER FPGA (OR2C15A) contains application circuitry, which can be programmed from the host. PCI FPGA (OR2C15A) contains PCI interface circuitry and a run-time reconfiguration controller for USER FPGA. The host is a personal computer with AMD K6-III (400 MHz) and FreeBSD 2.2.8R. USER FPGA can be re-programmed in 4.4 ms, and the application program can transfer data by using system call (read/write/mmap) or I/O instructions.

We implemented the unit that can handle up to $(p_\alpha, p_\beta) = (15, 15)$, which fits well the basic component of OR2C FPGA (16×4 bit SRAM). This unit operates at 16.5 MHz, which is the half of PCI clock frequency. 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. This single unit occupies 40% of OR2C15A, so the USER FPGA was designed to accommodate 2 independent units to operate in parallel. Total usage of USER FPGA is 86% with interface logic.

4 Evaluation

The execution time for subgraph isomorphism is data dependent, so we show the average of 100 trials on random generated graphs. We chose p_α, p_β, ed_α, and ed_β as the parameters here. Ed_α and ed_β are edge density, which is defined as the ratio of the number of edges of the graph to that of the perfect graph; $ed_\alpha = 2\,q_\alpha / p_\alpha(p_\alpha - 1)$.

Figure 2 shows the performance of our prototype accelerator, where $(ed_\alpha, ed_\beta) = (0.2, 0.4)$. The host computer dispatches a set of G_α and G_β to each unit, waits for "finish" signals by polling status registers, and then dispatches the next set of graphs. We use the average execution time for 100 sets as a performance measure. The performance shown in Figure 2 is normalized by that of fully optimized software implementation of Ullmann's algorithm on a 400 MHz

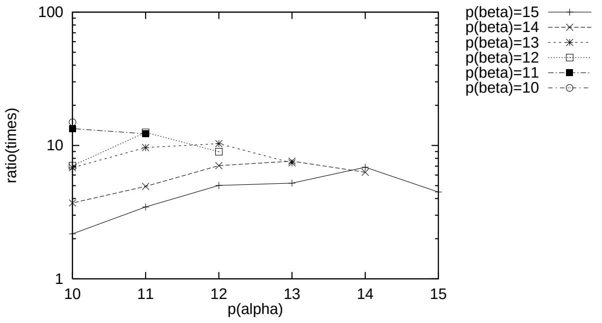

Figure 2: Performance Ratio

Pentium II. The real execution time of software was between 1 and 200 seconds for each graph set in Figure 2.

As seen, even our 16.5 MHz prototype can outperform the software implementation on commodity 400 MHz processor. The performance gain can be boosted easily by using larger FPGA or more accelerators in parallel.

5 Concluding Remarks

Currently, our prototype handles only small graphs. However, it is very easy to build hardware that can handle larger graphs, because our algorithm only requires modest resources as described in Section 2. The bottleneck is $O({p_\beta}^2)$ memory requirement, but it is not serious for the state-of-the-art LSI technology. The most serious problem is the explosion of execution time, even if we could build the larger hardware.

However, the problem can be greatly alleviated by making the accelerator more application-specific. For example, vertices are distinguishable each other in structural formula, according to their own property (e.g. carbon atom, benzene ring). Such property greatly restricts the mapping from G_α to G_β, making the search space drastically small. Consequently, we can solve a very large problem in reasonable time. FPGA is suitable for such application specific design.

Acknowledgments

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] S. Ichikawa and T. Shimada, "Reconfigurable PCI Card for Personal Computing," *Proc. 5th FPGA/PLD Design Conf. & Exhibit*, pp. 269–277, Tokyo, 1997.