Technische Universität München

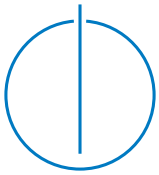
Department of Informatics

and

Chair of Electronic Design Automation

Inter Disciplinary Project

Author:	Gurusiddesha Chandrasekhara
Supervisor:	Prof. Dr.-Ing. Ulf Schlichtmann
Advisor:	Yong Hu
Submission Date:	31.10.15



Contents

1 Introduction

1.1 Motivation

Network on Chip (NoC) is a communication sbusystem on an integrated circuit typically between intellectual property(IP) cores in a System on Chip(SoC). In the modern NoCs there are numerous possible topologies and similarly in the integration of IPs for SoCs there can a lot of possible architectures. In order to find the optimal solution, some algorithms model the system as a graph and then optimize it iteratively. Graph algorithms play an important role in the these optimizations in perticular graph transformation or graph rewriting techniques.

Graph transformation concerns the technique of creating a new graph out of an original graph algorithmically. The basic idea is that the, design is represented as graph and the transformations are carried out based on transformation rules. Such rules consist of an original graph, a subgraph which is to be matched on to the original graph and a replacing graph, which will replace the matched subgraph. Formally, a graph rewriting system usually consists of a set of graph rewrite rules of the form $L->R$, with L being called the pattern graph (or left-hand side, LHS) and R being called the replacement graph (or right hand-side of the rule, RHS). A graph rewrite rule is applied on the host graph by seraching for an occurrence of the pattern graph and by replacing the found occurrence by an instance of the replacement graph.

As we apply these series of rules (often interchangebly), the number of resultant graphs inturn designs increase significantly. Usually the rule applying process goes in layers and represent the resultant graphs as nodes of a tree. Sometimes there can be lot duplicates in the resultant graphs as we apply rules interchangebly. Therefore, we need to identify the potential duplicates inorder to avoid unnecessary designs. There should be a logic to identify repeated node(graph in a tree) and merge this resultant node with the existing node. This can be termed as `node merging` problem.

1.2 Tasks

As explained before, to do node merging, we need to identify duplicates, we need to compare graphs. Now this problem became a exact graph matching or graph

isomorphism problem and this calls for a fast, efficient adn reliable graph isomorphism algorithm. This is important since, each time we apply a rule we get a new graph, which has to be compared with all the previously obtained graphs. For example, we apply a rule and get graph n, which has to be comapred with $n-1$ times. For rules n the number of comparison given by,

$$n(n-1)/2 \tag{1.1}$$

In addition to node merging, a little work was carried out on creating a Graphical editor for changing and viewing NoCs. The NoC can be modelled using Eclipse Modelling Framework(EMF).

Chapter 2 discusses the details of node-merging such as logic, graph isomorphism problem and algorithm, results. Chapter 3 discusses the introduction and possibility of using GMF in the current project. Chapter 4 gives conclusion and possible future work.

2 Node Merging

2.1 Node Merging Implementation

General process of node merging follows application of rule, and comparing with the existing tree nodes. And taking decision based on the result. Which is illustrated by the code below and comments will make it clear

```
 for(E rule: rules){
 V newNode = ruleExecutor.execute(node, rule)
 if(newNode != null) {
 /* For all the older nodes in the tree check if it has any match*/
 for(V oldNode:tree.getVertices()){
        if(nodeComparator.compare(oldNode, newNode)){
            /* If yes then just add an edge to the existing node */
            Integer edgeId = new Integer(tree.getEdgeCount());
          edgeIdToRuleMap.put(edgeId, rule);
         tree.addEdge(edgeId, node, oldNode, EdgeType.DIRECTED);
            flag = true;
            break;
        }
    }
    /* If this is a uniques graph then create a new node and add it to the tree */
    if(!flag){
    Integer edgeId = new Integer(tree.getEdgeCount());
     edgeIdToRuleMap.put(edgeId, rule);
     tree.addVertex(newNode);
    tree.addEdge(edgeId, node, newNode, EdgeType.DIRECTED);
        }
    }
}
```

2.2 Graph Matching

Two graphs are said to be isomorphic if they have the vertices connected in the same way. Formaly, two graphs G and H with graph vertices $V_n = \{1, 2, ..., n\}$ are said to be isomorphic if there is a permutation p of V_n such that $\{u, v\}$ is in the set of graph edges $E(G)$ iff $\{p(u), p(v)\}$ is in the set of graph edges $E(H)$.

Exact graph matching or graph isomorphism problem is the computational problem of determining whether two finite graphs are isomorphic. While the graph isomorphism is a problem in itself, its importance is very high in multiple fileds such as electronic design automation, Chemistry and molecular biology etc.

2.2.1 Complexity

Besides its practical importance, the graph isomorphism problem is a curiosity in computational complexity theory as it one of a very small number of problems belonging to NP neither known to be solvable in polynomial time nor NP-complete. At the same time, isomorphism for many special classes of graphs can be solved in polynomial time, and in practice graph isomorphism can often be solved efficiently. This is a special case of the subgraph isomorphism problem, which is known to be NP-complete. However Laszlo Babai has claimed that the Graph Isomorphism can be solved in quasipollynomial time. Quantities which are exponential in some power of a logarithm are called “quasipolynomial.”

3 Graphical Modelling Framework

4 Conclusion

List of Figures

List of Tables