Milestone Report — Optimization of Subgraph Isomorphism Algorithm

by Nelson Batista, Max Inciong, Francesca Truncale

Senior Project II

Professor Jianting Zhang

Fall 2017

Contents

1 Introduction

The goal of this project is to understand and optimize an algorithm for solving the NP-complete problem of subgraph isomorphism. An *isomorphism* between two graphs is a mapping from the vertices of one graph, say G, to another graph, say H, such that any two vertices which are adjacent in G are mapped to vertices which are adjacent in H. The subgraph isomorphism problem, in turn, is the problem of determining whether an isomorphism exists between a graph G and any of the subgraphs of a larger search graph, H. The issue that makes the subgraph isomorphism problem so difficult to solve for a particular pair of graphs is cycling through the main graph and expanding each node within the graph and checking if each of the generated subgraphs formed from each node is isomorphic to the control graph.

Fortunately, many attempts exist to solve the subgraph isomorphism problem. Among the earliest of these is the famous *Ullmann Algorithm*, proposed by Julian R. Ullmann in his 1976 paper, *An Algorithm for Subgraph Isomorphism.* He first describes a basic, naive approach to finding a mapping from the vertices of a graph to a subgraph of a larger graph. He goes on to define a “refine” procedure to dramatically reduce the number of possible mappings that must be checked. These will be covered in greater detail in the next section.

The goal of our project was to take this algorithm and improve its performance. Subgraph isomorphism is an expensive problem to solve, and finding multiple possible isomorphisms from one graph to subgraphs of another is even more expensive, so even slight improvements in the algorithm's performance is likely to lead to large performance improvements in any large-scale program which needs to use it often.

2 Ullmann Algorithm

2.1 Naive Approach

2.2 "Refine M" Procedure

2.3 Full Algorithm

3 Implementation of the Algorithm

3.1 Python Implementation and Performance

3.2 Initial C++ Implementation

3.3 Improved C++ Implementation

3.4 Optimizations through OpenMP

Process We wanted to understand the algorithm starting from pseudocode. First we gathered pseudocode to determine the algorithm for subgraph isomorphism. From there, we wanted to transcribe it into python, and then into C++ and from there parallelize it using threads. After using the python implementation, we decided to utilize something known as the Ullman algorithm for our C++ implementation. We have also since changed our plan to utilize CUDA and GPGPU to utilizing OpenMP

Graph Class The graph class that we ended up using is an unweighted directed graph. This means that when searching for isomorphisms, we have to take into consideration the direction of the nodes rather than simply the connections.within each graph. The graphs are unweighted, which means the isomorphism function does not take into account the values contained within each of the nodes.

Python Implementation

Ullman Algorithm

C++ Implementation

OpenMP OpenMP utilizes a computer's core in order to do separate work in parallel. We intend to place it where the bottlenecks are. The default number of cores for plenty of computers is four cores. We intend to place the parallelisms where there tend to be plenty of iterations. This means that theoretically, the program should run approximately 4 times faster than without the parallel processes. However, we have to take into consideration the overhead necessary to run OpenMP in the first place. It is more likely that it will take approximately 3 to 3.5 times faster given that the bottlenecks are completely parallelized. The most important place to place the OpenMP is in the refine_possible_assignments functions. This function is where the majority of the work goes, and goes through the most iterations. Ideally, we want to balance the loads equally so that each core in the computer does an equal amount of work so as to maximize the efficiency of the program. However, it should be enough to simply parallelize every iterative process, but that would not be efficient enough. Ideally we would further parallelize in nodes that have plenty of potential assignments. This way the loads are balanced and more work is done in the areas that require more work

Performance The python program was very inefficient but it was necessary for us to understand what we had to do for a C implementation. It took us between 13 and 16 seconds to determine if isomorphisms existed within our graphs and data.

4 Bibliography