Towards Automatic Parallelization using Graph Neural Networks

Surya Kant Sahu

September 30, 2019

1 Abstract

In recent years, the number of cores in processors have seen a gradual rise, but to make use of them, programmers have to either re-write the single-threaded code or provide "hints" for the compiler to try to parallelize the code block. Automatic Parallelization aims to automate the process of analysing code and parallelizing existing program. This has been achieved upto some extent (Fortran Compilers), but due to the complex code understanding required, there are no efficient solutions.

We first prove that this problem is NP-Hard, then this paper introduces problem formulation where a Learning agent learns to optimize a greedy algorithm, trained by fitted Q-learning. We exploit the fact that programs can be represented as Directed Acyclic Graphs (DAGs), the problem is formulated as Graph-to-Graph problem, where a learning agent learns to generate several graphs optimized for p processors given a single sequential program as a DAG.

We built a toy dataset of generated DAGs, and we compare models trained on theoretical reward based on Work-Span law, and actual speedup of Python programs generated by DAGs, and show that these models can scale to an order of magnitude higher than the training set.

2 Introduction

Suppose a computer program is represented as a graph $G = \{V, E, W\}$, where V is the set of vertices representing some atomic computation, E is the set of ordered pairs (v_1, v_2) representing the dependency of v_2 on v_1, and W is defined as $W = \{w(v) | \forall v \in V\}$, where $w(v)$ is the time required for computation v to complete. Let $F_t(\hat{G}, p)$ be the execution time of $\hat{G}$ in p processors. We formulate this problem as Node Classification i.e. the model has to predict the right class $\hat{p}$ for each node in the graph G, where p is the number of available processors, and p is the assigned processor for the vertex v, to minimize the running time for the graph G. A Graph G is split into multiple subgraphs $g_i \forall i : 0 \leq i < p$ where g_i is the graph to be executed by processor p_i. The finishing times of all graphs are measured, then a reward is calculated which is a function of the finishing time and derived from Work-Span law, the model is optimized to maximize the expected reward.

3 Methodology

Due to lack of actual DAGs of programs to test our method, we synthesized a Dataset of randomly generated DAGs. These are randomly generated using a popular library NetworkX in Python, and the work for each node is also randomly assigned.