Deep Graph Learning for Program Analysis and System Optimization

Goals

Rather than relying on expert compiler writers to develop clever heuristics to optimize the code, we can utilize machine learning to optimize a compiler to make the machine run faster. We represent high level programs as weighted graphs. This enables the proposed framework to efficiently analyze the structural information flow of software programs and determine their parallelization.

Problem Formulation

Given a complex software application, the goal is to learn a mapping function that predicts which code segments would run best on a specific hardware device in heterogeneous hardware platforms.

Framework



Unified end-to-end deep graph learning framework

Dynamic Dataflow Graphs

 \Box A dynamic dataflow graph is defined as G (N, E, W)

- \Box N: a set of instructions dynamically generated at runtime
- $\Box E$: a set of dependent edges between instructions
- W: a set of edge weights representing the execution time for memory operations



Graph Representation in Graphs

- Collect the representative dynamic trace. This is augmented with the help of the control flow graph (CFG). The goal is to fully color the CFG.
- Check data, control, and memory dependencies and insert edges accordingly

Graph Example



An example of a standard two-dimensional nine point stencil calculation and its corresponding graph representation. By adopting this graph representation, we can see that some patterns are recurring due to for loops used in the code.

Unified Deep Graph Learning Framework

- Graph autoencoder: partition the dynamic dataflow graph into representative kernels which later will be used to predict the label by graph neural networks.
 (a) Compute an embedding of each node in a graph;
 - (b) Learn the latent embedding using GCN;
 - (c) Compute the pairwise distance between nodes;
 - (d) Perform spectral clustering;
- Graph neural networks: After each kernel is identified, we use a GNN to predict the correct platform to execute the kernel by updating the node vectors iteratively.

Evaluations and Results

We use gem5-gpu to simulate a varying number of outof-order CPU cores and GPUs, and we compare against the different baselines for eight applications against our PGL framework.

