Shiva Badruswamy
Graduate CS Student
Stanford University
Stanford, CA

Machine learning and compiler optimizations: using inter-procedural analysis to select optimizations

OVERVIEW

Current machine learning models for compiler optimization select the best optimization strategies for functions based on isolated per function analysis. In this approach, the constructed models are not aware of any relationships with other functions around it (callers or callees) which can be helpful to decide the best optimization strategies for each function. In this project, we want to explore the SCC (Strongly Connected Components) call graph to add inter-procedural features in constructing machine learning-based models to find the best optimization strategies per function. Moreover, we want to explore the case that it is helpful to group strongly related functions together and optimize them as a group, instead of per function.

TENTATIVE TASKS

1. Explore how machine learning can be used to develop heuristics that can be employed to balance analysis passes and transformations toward optimal compile-time, code-size, and compute/memory performance.
2. Use ML to identify patterns and code features that can be checked for using usual techniques.
3. Use ML to identify shortcomings in existing heuristics such as transformation cut-off values or pass schedules. For discovered shortcomings, identify inputs and propose alternative heuristics to improve performance for those inputs.
4. Figure out how to get and layout data for analysis by ML models to get the best possible loss rates. The usual ML model setup tasks and decisions such as identifying the right model, width and depth of model, cost of data labeling efforts versus using unsupervised models, training/test/val data splits and sufficiency of data, hyper parameters to be tuned, proper activation and loss functions, standardizations and normalization layers needed, batch size estimation, need for GPU clusters to accelerate training etc.
5. Explore what needs to be built into the LLVM libraries to enable online inference whenever an optimizing compiler is run for a ML pass. Specifically, lay out how analysis and transformations will yield for a ML pass.

TENTATIVE GOALS

1. Improved heuristics for existing (inter-procedural) passes, e.g. to weight inlining versus function cloning based on code features.
2. Machine learning models to select the best optimizations using code features and inter-procedural analysis. This model can be used for functions in isolation or groups of functions, e.g., CGSCCs.
3. Time permitting project extension: Extend the SCC call graph to functions called on GPUs with potential opportunities for building optimized kernels that work across both CPUs and GPUs to optimally balance compile-time, code-size, and compute/memory performance.

NOTES

1. After discussions with LLVM mentors for this project, I have learnt that this topic is very much open ended. So, this document might undergo significant changes on describing goals and tasks as more clarity is gained and more data is collected and explored.
2. Extensions to GPU functions depend on how much LLVM compiler libraries can be utilized for supporting compilation to GPU assembly. For instance, Nvidia's compiler nvcc uses gcc to compile to CPU assembly and PTX to compile to GPU assembly in a combined fashion. Can LLVM compiler be used as well alongside PTX is a question that is unclear for now.