CS294-112 Project Proposal: Using Deep Reinforcement Learning for Compiler Optimization Selection and Ordering

Objective: The impact of compiler optimizations is order-dependent. To produce good results, the compiler needs knowledge of the program features, the underlying systems and the run-time profiles. In this project, we will explore the benefits of applying deep reinforcement learning algorithms to achieve a better ordering of LLVM compiler optimization passes given a specific program. This NP-hard challenge is generally referred to as the phase-ordering challenge of the compiler [1]. The LLVM compiler could be used to compile multiple coding languages, but as a case study, the optimized compiler will be used in compiling different High Level Synthesis (HLS) programs.

Related Work: As the search space of phase-ordering challenge is too large for brute-force search, many works have been proposed to explore the space by using machine learning. Huang *et al.* tried to overcome this challenge for HLS applications by using modified greedy algorithms [2]. In [3] both independent and Markov models were applied to automatically target an optimized search space for iterative methods to improve the search results. In [4], genetic algorithms were used to tune heuristic priority functions for three compiler optimization passes. In [5] the challenge was formulated as a Markov process and supervised learning was used to predict the next optimization, based on the current program state.

Technical Outline: In the deep reinforcement learning setting, we define different optimization passes as the actions to take, the negative clock cycle count of the program as the reward and the features of a transformed program as the states. The HLS toolchain can generate cycle count with fairly low latency (typically less than 15 seconds). This justifies why it is chosen as the environment simulator in the deep reinforcement learning setting. The main challenges of this project include formulating the states and properly representing the actions. The states will be extracted from the program as 32 predefined features, *e.g.,* the number of memory accesses, loops and instructions. To extract these features, an appropriate framework will be built. There will be 38 different actions (passes), which will be considered in each step (compilation). Some of these passes might not be binary. For example, when using loop-unrolling an unrolling factor must be set. This works will make the following contributions:

- Build a framework that will extract important LLVM features from a specific HLS program.
- Use these features in different deep reinforcement learning techniques to overcome the phase ordering challenge.
- Tune these deep reinforcement learning techniques to better fit these programs and further minimize their run-time.

References

[1] S.-A.-A. Touati and D. Barthou, “On the decidability of phase ordering problem in optimizing compilation,” in *Proceedings of the 3rd conference on Computing frontiers*, ACM, 2006, pp. 147–156.

[2] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, S. Brown, and J. Anderson, “The effect of compiler optimizations on high-level synthesis for fpgas,” in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, IEEE, 2013, pp. 89–96.

[3] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. O’Boyle, J. Thomson, M. Toussaint, and C. K. Williams, “Using machine learning to focus iterative optimization,” in *Proceedings of the International Symposium on Code Generation and Optimization*, IEEE Computer Society, 2006, pp. 295–305.

[4] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O’Reilly, “Meta optimization: Improving compiler heuristics with machine learning,” in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, ser. PLDI ’03, 2003.

[5] S. Kulkarni and J. Cavazos, “Mitigating the compiler optimization phase-ordering problem using machine learning,” in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA ’12, 2012.