Izmir Institute of Technology
Computer Engineering Department
CENG513 Final Exam Spring 2024
Question 3

Student Name: Gökay Gülsoy Student No: 270201072

June 11, 2024

Question 3

Previously compilers run on computers which were underpowered. Over the few decades, computer technology both in terms of hardware and software stack improved considerably. Improvements in the hardware stack are faster CPUs with multicores, GPUs & accelerators, more memory and storage, increased computation power with cloud computing capabilities. Improvements in the software stack are better and faster algorithms, Integer linear programming techniques, SMT solvers, Theorem provers and advancements in Deep learning. Multicore architectures may allow compiler optimizations which are usually very time consuming due to required transformation state space search much faster by utilizing more than one CPU core. GPUs and accelerators can parallelize the algorithms used in different parts of compiler construction to speed up overall construction process. Having machines with more memory and storage can allow complex algorithms which have high space complexity to run easily and process more data at hand. Along with advancements in hardware infrastructure faster or less space consuming algorithms which were impractical to implement previously, can be practically implemented to speed up different phases in compiler construction. Three key points for proposed next generation compiler technology are building compilers as a service, automating compiler construction and using machine learning in compiler optimizations [1].

Currently compilers are still build with ancient technology which means that they are command line tools running on developers workstation with a single CPU thread and flow consists of sequential execution of passes. By building compilers as a service, compilers can have access to tremendous processing power along with huge memory and storage capabilities by utilizing specialized accelerators. Compiler infrastructure can be built with modern system building methods and frameworks. It can give ability to learn from everyone and quickly improve infrastructure over time. According to Mendis's model of compiler optimization, series of optimization passes are applied to input IR to find out possible fastest and correct output IR. This procedure requires searching transformation space and finding optimization strategy with minimum cost. As the transformation space can be very large, transformation space search can be automated and optimized by utilizing machine learning. Today transformations are manually created, but this procedure is usually very time consuming and hard to maintain due to thousands of contributors and millions of lines of code. Out of over 3,600 x86 instructions, compilers can generate only less than 1000 [2]. Automated compiler construction tools are being developed but they are falling behind the advancements done by computer architects on processor microarchitectures and ISAs. Problem with naively using ML systems is that they do not guarantee correctness which is the most critical task that should be guaranteed by compiler. In order to guarantee correctness, more accurate cost models should be created, but accurate modeling of a processor manually is hard because of high complexity of today's ISAs. Researchers propose to build a neural network models so that for new architectures instead spending a lot of human hours for rewriting auto-training can be utilized.

I have chosen papers "VeGEN: A Vectorizer Generator for SIMD and Beyond" and "Vemal: Learnt Vectorization". Paper named "VeGen: A Vectorizer Generator for SIMD and Beyond" is about a parallelism type called "Lane Level Parallelism" (LLP) which captures the model of parallelism implemented by both SIMD and non-SIMD vector instructions. Researchers present a vectorizer generator called "VeGEN" which automatically generates a vectorization pass and it is using target-independent vectorization algorithm. This design enables systematically targeting non-SIMD instructions which until now requires ad hoc coordination between different compiler phases. Overall, Researchers described a framework that jointly performs vectorization and vector instruction selection while maintaining the modularity of traditional target independent vectorizers for SIMD instructions. Paper named "Vemal: Learnt Vectorization" is about learning end-to-end compiler optimization policies with gated graph neural networks which surpass the performance of hand-crafted compiler heuristics by imitating an optimal solution to Instruction packing problem of SLP (superword level parallelism) vectorization [3]. If we compare these two papers both of them are in essence related to utilizing architectural support provided by computer architects at a compiler level to achieve better results and they emphasize that even though there are many architectural improvements over few decades compiler technology is falling behind them and not truly utilizing most of these features, so compiler technology should be adapted to today's modern architectures and microprocessors.

I want to ask speaker the question that why compiler technology is falling behind the developments compared to other fields of computer science, is it related to complexity of the filed because it requires knowledge and experience in many other subbranches of computer science such as data structures and algorithms, formal languages and automata theory, and computer architecture.

References

[1] Oct. 2022. [Online]. Available: `https://www.youtube.com/watch?v=abodcRZMWgg`.

[2] Y. Chen, C. Mendis, M. Carbin, and S. Amarasinghe, "Vegen: A vectorizer generator for simd and beyond," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, Virtual, USA: Association for Computing Machinery, 2021, pp. 902–914, ISBN: 9781450383172. DOI: `10.1145/3445814.3446692`. [Online]. Available: `https://doi.org/10.1145/3445814.3446692`.

[3] S. Amarasinghe, "Compiler 2.0: Using machine learning to modernize compiler technology," in *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES '20, London, United Kingdom: Association for Computing Machinery, 2020, pp. 1–2, ISBN: 9781450370943. DOI: `10.1145/3372799.3397167`. [Online]. Available: `https://doi.org/10.1145/3372799.3397167`.