AI in Programming and Compilers

Moderator: Henry Dietz[0000-0002-5878-881X]

Panelists: AB Siddique, Aniket Shivam , P. Sadayappan

University of Kentucky, Lexington KY 40506, USA
hankd@engr.uky.edu

Abstract. Especially in the past year, it seems that AI (artificial intelligence) has demonstrated the potential to augment or even obsolete a lot of the analysis and methods that mankind has spent huge amounts of effort developing. In the late 1980s, GP (genetic programming)[1] was proposed as a method to automatically create programs, but the technique has not found broad application in the programming language and compiler communities despite some impressive accomplishments in inventing human-competitive algorithms[2]. In contrast, LLM (large language model)[3] tools like ChatGPT (chat generative pre-trained transformer) seem to be able to synthesize well-structured textual responses to arbitrary queries, including queries that require the system to write program code. Even before that, neural networks and deep learning technology[4] had begun to show that trained AI is capable of providing valuable insights into, or even solving, a very broad range of complex problems. However, until recently, AI techniques have not been widely used in optimizing/parallelizing compilers and they still are not commonly used for writing programs. The panelists here are asked to share some of their insights as to where these types of AI have real potential and where perhaps they do not.

Keywords: Large Language Models, Generative Pre-trained Transformers, Neural Networks, Deep Learning, Genetic Programming, Parallel Programming, Optimizing Compilers, and Parallelizing Compilers.

1 Introduction

This panel is bringing together … to explore the application of AI techniques to the problems of writing and maintaining parallel programs and construction of better optimizing/parallelizing compilers.

The panel discussion at LCPC (and this paper) begins with a brief introduction by the moderator followed by ten-minute position presentations from each of the panelists. Each panelist independently determines the content of their position presentation and submits a section for this paper after the workshop. To ensure collection of opinions on some specific topics, the moderator prepared and distributed this "Introduction" section of the paper before the workshop, giving three prompts to be addressed by each panelist. The conclusion is written by the moderator after the workshop, summarizing the discussion that followed the position presentations.

1.1 AI Writing Parallel Programs

When the moderator entered the parallel processing field in the late 1970s, he saw a strong consensus that parallel programming would soon become a standard skill expected of all people who program computers. Well, that has not yet happened. Parallel programming is hard, and as complexity of both applications and parallel computer systems have grown much larger, parallel programming has arguably become even more difficult.

Does the ability of AI methods to write programs mean that perhaps the answer to the elusive goal of making everyone competent in producing parallel programs is to not have the humans write programs at all, but merely to have humans describe the desired behavior and have AI tools create the parallel program? Certainly, many people have been suggesting that LLMs have this potential, however, at the same time there is the understanding that LLMs have no underlying model for the meaning of the code they emit. In fact, it could be argued that as code authors, LLMs are little more than sophisticated automation for plagiarism of existing documents in their training set. However, there are other AI methods, such as GP, which are documented as being able to invent new algorithms[2], and perhaps methods combined will vastly outperform any method alone? Perhaps simply limiting the scope of applications is sufficient for LLMs to write good code[5]? Unfortunately, LLMs also are well know to be prone to "hallucination," and that makes their output difficult to trust from a correctness perspective.

Ironically, even if LLMs fail to write trustworthy code, there is a large and growing community that sees them as useful in the program lifecycle especially for testing and debugging[6][7][8][9]. This utility is largely rooted in the LLM ability to explain code and to generate test cases. It is useful to note that debugging parallel programs is generally much more difficult than debugging serial programs because program state is more complex and less observable, so more effective debugging methods are potentially very valuable.

Thus, the first prompt given to the panelists is:

> ***Perhaps AI will mean programs are written by giving a prompt in English and current programming languages will become like assembly language is treated now, or perhaps AI tools will simply help with identifying algorithms or debugging code. How do you see AI methods being used in parallel programming?***

1.2 AI in Compiler Optimization and Parallelization

Just as writing parallel code is hard for humans, it can be very difficult for humans to create highly efficient optimization and parallelization methods. As far back as 1987, the Superoptimizer[10] tried to help by using a pruned exhaustive search to find optimal sequences of machine instructions for simple operations like absolute value or integer multiplication by a constant. AI methods have the potential to much more dra-

matically focus the search space, perhaps making it possible to generate better code for much more complex program fragments.

Optimizing and parallelizing compilers generally contain a large number of "correctness preserving" optimization and parallelization transformations, some of which take multiple tuning parameters. Selection of which transformations to execute in what order and with what parameter values is a very complex problem, and the relationship between these decisions and efficiency of execution on a particular target machine is very opaque. Perhaps AI methods can learn how to make better adjustments for a particular target architecture?

The second prompt is about use of AI in compilers themselves:

> ***How do you see AI methods being used within optimizing, parallelizing, compilers? Please give a brief example or two and explain for each if AI is augmenting or replacing other compiler methods.***

1.2 Which AI

Although AI has been a field for many years, it is only relatively recently that some AI methods have become mainstream technologies for solving specific types of problems. For example, the base concepts of neural networks were around for many decades before meeting wide acceptance and common use. The moderator first heard of neural networks by the name "multi-valued logic," years later again as "fuzzy logic," more years later as "neural networks," and now most often as "deep learning" – with various significant differences. The key point is that this approach was not of great value until implementation of training methods (using parallel computers and software like TensorFlow[11]) became strong enough to deal with training large networks with huge data sets. Other AI techniques are maturing too. As they mature, we develop a better understanding of the kinds of problems each can and cannot solve.

The third and final prompt for the panelists is simply:

> ***There are a lot of different AI methods, algorithms, and tools out there. Which are the ones that every optimizing/parallelizing compiler writer should be familiar with? For each, briefly explain why.***

2 Panelist: ...

Position statement contributed by each panelist...

5 Conclusion

To be written by the moderator after the panel.

References

1 Koza, J.R., “Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,” Stanford University Computer Science Department technical report STAN-CS-90-1314, (1990), http://www.genetic-programming.com/jkpdf/tr1314.pdf

2 Koza, J.R.; Keane, M.A.; Streeter, M.J.; Mydlowec, W.; Yu, J.; & Lanza, G. (2003). Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Springer. ISBN 1-4020-7446-8

3 Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang et al. "A survey on evaluation of large language models." arXiv preprint arXiv:2307.03109 (2023).

4 Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.

5 Kashefi, Ali, and Tapan Mukerji. "Chatgpt for programming numerical methods." Journal of Machine Learning for Modeling and Computing 4, no. 2 (2023).

6 Biswas, Som. "Role of ChatGPT in Computer Programming.: ChatGPT in Computer Programming." Mesopotamian Journal of Computer Science 2023 (2023): 8-16.

7 Surameery, Nigar M. Shafiq, and Mohammed Y. Shakor. "Use chat gpt to solve programming bugs." International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290 3, no. 01 (2023): 17-22.

8 Liu, Yue, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D. Le, and David Lo. "Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues." arXiv preprint arXiv:2307.12596 (2023).

9 Tian, Haoye, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. "Is ChatGPT the Ultimate Programming Assistant--How far is it?." arXiv preprint arXiv:2304.11938 (2023).

10 Massalin, Henry. "Superoptimizer: a look at the smallest program." ACM SIGARCH Computer Architecture News 15, no. 5 (1987): 122-126.

11 Abadi, Martín. "TensorFlow: learning functions at scale." In Proceedings of the 21st ACM SIGPLAN international conference on functional programming, pp. 1-1. 2016.