Program Synthesis meets Large Language Models — A portable and performant way to program HPC applications

Principal Investigator: Harshitha Menon; gopalakrishn1
PI Organization: Computation/CASC
Mission Focus/Core Competency: High Performance Computing, Simulation & Data Science

Abstract

As we reach the limits of Moore's law and the end of Dennard scaling, increased emphasis is being given to alternative system architectures and computing paradigms to drive future performance improvements. As specialized and diverse hardware becomes increasingly available, it is important to ensure that HPC applications can run correctly and efficiently on them. Given the complexity of scientific codes, manually rewriting them to support emerging hardware is a challenging task. Large Language Models (LLMs) have shown great potential in a variety of software related tasks, from code completion to competitive programming, and can be used to assist in software portability. However, there are challenges associated with using it in HPC. To this end, we propose to leverage the power of LLMs to enhance developer productivity and software portability by developing a framework to use LMs fine-tuned for HPC tasks and providing methods to check for correctness. Our project will allow LLNL scientific applications and portable abstraction libraries like RAJA to leverage LLMs to assist in code portability for current and emerging hardware.

1 Description

High-performance computing (HPC) is essential for accelerating scientific discoveries as it enables scientists to perform complex simulations that would otherwise be impossible. To keep up with the computational demands of modern scientific research, HPC relies on the latest hardware technologies. With the end of Dennard scaling and slowdown of Moore's Law, we will see more specialized and heterogeneous hardware being used HPC [1]. With hardware heterogeneity becoming more prevalent in HPC, it has become important to ensure that scientific software runs correctly and efficiently across a range of platforms. In the past, HPC applications have had to support new architecture features, from SIMD vector registers, massively parallel systems, to multi-core architectures. While these features have provided significant performance benefits, they often required developers to rewrite their code to utilize the new hardware efficiently. As new architectures often have different hardware features than its predecessor, rewriting them would require developers to have a deep understanding of the hardware and how to program it effectively to achieve the best performance gains. Given that the software complexity of scientific software and libraries has increased significantly, rewriting them for new architectures is a challenging task.

To keep up with the architecture advances, simply relying on compiler technologies would be inadequate. We anticipate that software engineers will have to rely on advanced techniques, such as program synthesis, to port softwares on emerging architectures. Program synthesis involves automatically generating programs from high-level language specification and is expected to reduce software development overhead. The emergence of large language models (LLMs) such as GPT-3, OpenAI Codex, and BigCode [2], has enabled generation of code from natural language descriptions of programmer intent, which presents

a new avenue for program synthesis. LLMs have shown incredible potential in a variety of software-related tasks ranging from competitive programming, code completion, bug detection, and code optimization, among others. There are early indications that program synthesis using LLMs are about to bring a fundamental transformation in the way software will be developed. The U.S. Department of Energy (DOE) recognizes this opportunity, as evident from various reports such as the DOE's report on extreme heterogeneity [1] and the AI for Science report [3], where the need for AI-driven methods to create scientific software has been identified as one of the primary research directions.

Although the progress in the field of LLMs brings optimism, it is not without challenges. On the one hand, these models have the potential to boost developer productivity and portability of code by providing AI-automated solutions. On the other hand, dataset collection, specification of programmer's intent and architectural characteristics in the HPC realm remain open problems. In addition, care should be taken as these models lack an understanding of program semantics, which means that there are no guarantees regarding the correctness of the generated code.

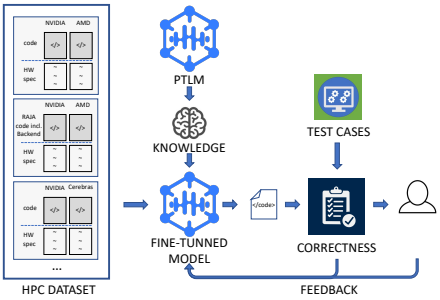

Figure 1: Overview of our approach

The goal of this proposal is to enable HPC programs to leverage the power of LLMs to enhance developer productivity and software portability by focusing on developing LLMs fine-tuned to HPC applications and providing tools to ensure correctness. Furthermore, our work can assist in producing backends of portable abstraction libraries, such as RAJA and Kokkos, for future architectures. We have identified three main objectives for this work. The first is to collect dataset pertaining to HPC; the second is to develop models fine-tuned on HPC dataset; and the third is to develop program verification tools to ensure the correctness of the generated code. Figure 1 illustrates the overall workflow involving the three goals of this project. Using a disciplined approach to applying LLMs in scientific applications has the potential to assist in portability of software and improve programmer's productivity, which translates to monetary savings in the deployment of new HPC systems.

Thrust 1: HPC Dataset for Program Synthesis

While large version control repositories, such as GitHub, provide a wealth of data for machine learning models, HPC codes account for only a small fraction of the dataset. Furthermore, the PTLMs aren't trained for specific tasks such as porting software efficiently from one hardware to another. The first thrust will involve collecting datasets for HPC relevant tasks that will be used to fine-tune PTLMs. For code portability, the data collection process would involve gathering instances of how a code was ported from one architecture to another. We can obtain such datasets from resources like RajaPerf, Codesign proxy apps that have different implementations for different architectures. We will develop an architectural model, similar to [4], that will be used for specifying hardware characteristics. Our portability-oriented dataset will consist of code examples and hardware specifications

showcasing how the code was ported.

Thrust 2: Training models for HPC codes

Similar to how the machine learning community employs fine-tuning and transfer learning to perform well on tasks unseen, we will use pre-trained language model (PTLM) as the starting point and retrain it on HPC codes. In our recent work [5], we demonstrated that PTLMs can be fine-tuned to model HPC and scientific code to outperform general LLMs in HPC-specific tasks, such as autocomplete HPC functions, add OpenMP pragmas, and model performance of source code changes. We plan to use a model from the *BigCode project*, where Co-I Guha is part of the leadership team. BigCode is training PTLMs on permissively licensed, open source code on GitHub. Its preliminary 1B parameter models are SOTA for their size [2]. The objective of this thrust is to train a model, fine-tuned on data that includes code for both source and target hardware, in addition to their hardware specifications, to learn a mapping between them.

Thrust 3: Correctness and Verification

Although these LLMs are good, they are not perfect, and at times they may produce code that looks reasonable but with subtle bugs. In order to build trust in such a system, we need to build tools to check for correctness. We will verify the results of the model in several different ways. We will use test cases and generate inputs that can unearth subtle bugs in code. We will tap into years of research from the Programming Language (PL) community and work of Co-I Guha to develop formal methods to verify the generated code and create new test cases. We will identify and characterize the most common bugs that appear in the generated code and will check for a limited number of issues, such as incorrect branching and segmentation faults, among others. Finally, we will employ mechanisms to obtain feedback from users and verification tools and use them for Reinforcement Learning to further improve the model.

2 Expected Outcomes

Expected outcomes of this project are the following: (1) Corpus of HPC code edits and hardware specifications; (2) A framework to specify hardware features; (3) Model trained on portability-specific dataset for HPC that learns code mapping for architectures; (4) Verification tool to check for bugs in the generated code and generate new test cases; (5) New technique that uses the feedback for reinforcement learning to further improve the model.

Risks and Mitigation. Machine learning models are known to require large amounts of data and compute resources to learn in order to make accurate predictions. Mitigating this is one of the main tasks of the project. We will use PTLMs, which have been trained on vast amounts of publicly available code repositories, and fine-tune it with additional data specific to the problem domain. This will enable us to train faster while using lesser data.

Relevance to Mission Focus Area and Core Competency. Our project directly supports one of the LLNL core competencies: high-performance computing. Future deployment of HPC systems for NNSA stockpile stewardship simulations will benefit from our work.

Key Team Members

The commitments from the team members are as follows: H. Menon at 40%, G. Geor-

gakoudis at 25%, K. Parasyris at 20%, Postdoc at 75%, Prof. A. Guha + Ph.D Student (subcontract - Northeastern University)

Advisors: B. Kailkhura at 5%, D. Beckingsale at 5%, I. Laguna at 5%, T. Gamblin.

Estimate of Proposal Budget. FY23: $743K (1.0 FTE, 1 Postdoc, subcontract); FY24: $798K (1.0 FTE, 1 Postdoc, subcontract); FY25: $833K (1.0 FTE, 1 Postdoc, subcontract).

References

[1] JS Vetter, R Brightwell, M Gokhale, P McCormick, R Ross, J Shalf, K Antypas, D Donofrio, A Dubey, T Humble, et al. Extreme heterogeneity 2018: Doe ascr basic research needs workshop on extreme heterogeneity (2018). *DOI: https://doi.org/10.2172/1473756*.

[2] The BigCode Team. SantaCoder: Don't reach for the stars!, 2023. `10.48550/arXiv.2301.03988`.

[3] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. Ai for science: Report on the department of energy (doe) town halls on artificial intelligence (ai) for science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[4] Li-Wen Chang, Izzat El Hajj, Christopher Rodrigues, Juan Gómez-Luna, and Wen-mei Hwu. Efficient kernel synthesis for performance portable programming. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[5] Daniel Nichols, Anirudh Marathe, Harshitha Menon, Todd Gamblin, and Abhinav Bhatele. Modeling parallel programs using large language models. *Under Submission*, 2023.