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 end of Dennard scaling, increased attention is being paid to emerging computing paradigms and alternative system architectures to drive future performance improvements. With the emergence of more specialized and heterogeneous hardware, we need to ensure that HPC applications can run correctly and efficiently on these platforms. Given the complexity of scientific codes, rewriting them to support them on emerging hardware is a challenging task, and relying only on compiler techniques will be inadequate. 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 it; generating HPC specific dataset, training on HPC related tasks, and ensuring correctness of the generated program. To this end, we propose to leverage the power of LLMs to enhance developer productivity and software portability by developing a systematic 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 researchers and scientists to perform complex simulations and data analysis that would otherwise be impossible or impractical. For this, it relies on the latest hardware technologies for computational power to keep up with the demands of modern scientific research. However, recent hardware trends have made it harder to achieve this level of performance. With the end of Dennard scaling and slowdown of Moore's Law, which had powered the performance improvements via increased transistor count with constant power density, designers are looking at alternative approaches to scaling, such as more specialized hardware. Over the next several years, we will see more specialized and heterogeneous hardware in HPC [].

With hardware heterogeneity becoming more prevalent in HPC, it has become important to ensure that scientific software can run correctly and efficiently across a wide range of platforms. In the past, HPC applications have had to support various 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. Furthermore, with the emergence of heterogeneous systems, such as CPUs with FPGAs and AI accelerators, developers must be able to write code that can run efficiently on a variety of different hardware systems. As new architectures often have different hardware features than its predecessor, such as amount of concurrency, different memory hierarchies, or different instruction sets, 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 and they now contain millions of lines of code, rewriting them for new architectures is a challenging undertaking.

To keep up with the architecture advances, simply relying on compiler technologies would be inadequate for exploiting the full potential of the future heterogeneous hardware. 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. It can greatly enhance programmer productivity and software performance. The emergence of large language models (LLMs) such as GPT-3, Open AI's Codex, Polycoder, and AlphaCode, has enabled the generation of code solely based on natural language specifications of programmer intent, which presents a new avenue for program synthesis. Large language models (LLMs) have shown incredible potential in a variety of software-related tasks ranging from competitive programming, code completion, bug detection, code optimization, document generation, and code search. 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 [] and the AI for Science report [], 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 productivity and portability of code by providing AI-automated solutions. On the other hand, dataset selection, 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.

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 LMs fine-tuned to HPC applications and providing methods and tools to ensure correctness and enable verification. Furthermore, this approach can assist portable abstraction libraries like Raja and Kokkos in porting them onto future architectures. We have identified three main objectives for this work. The first is to collect dataset pertaining to HPC to tune LMs; the second is to develop tools to capture programmer's intent; and the third is to develop program verification tools that can 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 procurement and administration of new HPC systems.

Thrust 1: HPC Dataset for Program Synthesis

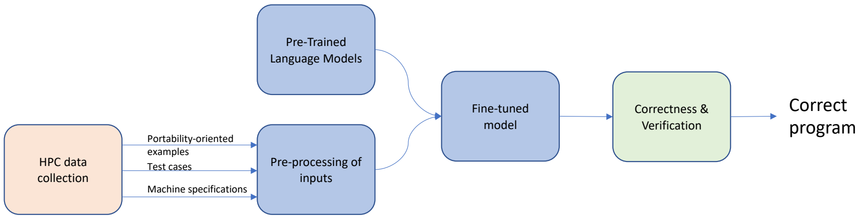

Figure 1: Overview of our approach

Similar to how the machine learning community employs fine-tuning and transfer learning on smaller datasets to perform well on tasks unseen, we will use pre-trained language models (PTLM), which have been trained on vast amounts of publicly available code repositories from platforms like GitHub. We will then fine-tune it with additional data specific to the problem domain or task at hand. 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. 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. The goal is to create a portability-oriented dataset and use it to perform fine-tuning or transfer learning.

Thrust 2: Tools to capture programmer's intension and hardware characteristics

One of the major challenges in program synthesis is capturing the user's intent to the level of detail required by the model to create high-quality code without burdening the user. We will need multi-modal specifications, involving code, examples, and test cases, to describe complex functionality. The idea is that while providing specifications using any of these modalities on their own can result in gaps, a combination of them would address shortcomings. Furthermore, we will develop language support to provide specification of architecture characterization to address portability. The HPC dataset consisting of the portability examples along with the test cases and machine specifications will be used to fine-tune a model that can assist in the portability task.

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. Moreover, pre-trained language models do not understand the syntax or semantics of the code, which means that they do not necessarily produce correct code. 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 work from the Programming Language (PL) community to develop formal methods to verify the generated code and generate test cases. Finally, we will employ mechanisms to obtain feedback from user interactions to improve the system.

2 Expected Outcomes

The project milestones, along with the responsible team member (in parentheses), are outlined below.

Year 1	
Year 2	
Year 3	

Expected outcomes of this project are the following: (1)

Risks and Mitigation. Machine learning models are known to require large amounts of data to learn in order to make accurate predictions. Furthermore, training large models is time-consuming and require significant compute resources. 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 supports directly one of the LLNL core competencies: high-performance computing. Future procurement of HPC systems for NNSA stockpile stewardship simulations will benefit from our findings.

Key Team Members
The commitments from the team members are as follows: H. Menon at 40%, G. Georgakoudis 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 Request. FY23: $743K (1.0 FTE, 1 Postdoc, subcontract, travel); FY24: $798K (1.0 FTE, 1 Postdoc, subcontract, travel); FY25: $833K (1.0 FTE, 1 Postdoc, subcontract, travel).

References