Creating Code LLMs for HPC: It's LLMs All the Way Down

Aman Chaturvedi
achaturv@umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

Daniel Nichols (advisor)
dnicho@umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

Siddharth Singh (advisor)
ssingh37@umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

Abhinav Bhatele (advisor)
bhatele@cs.umd.edu
Department of Computer Science, University of Maryland
College Park, Maryland, USA

Abstract

Large language models (LLMs) are increasingly being used by software developers, researchers, and students to assist them in coding tasks. While newer LLMs have been improving their coding abilities with regards to serial coding tasks, they consistently perform worse when it comes to parallelism and HPC-related coding tasks. Bridging this gap and creating HPC-capable code LLMs could drastically improve the quality and quantity of code research software developers can write. The current poor performance of LLMs on HPC-related problems can be partially attributed to the lack of significant HPC data in their training, which is what we address in this poster. We present HPC Coder v2, a new LLM created by fine-tuning a previous code LLM using HPC synthetic data. We demonstrate that it is one of the most capable open-source LLMs for generating parallel code to date.

1 Synthetic Data Generation

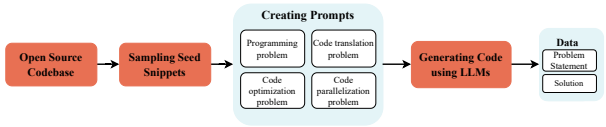

Figure 1: Synthetic data generation process

Previous work has demonstrated that fine-tuning smaller LLMs on synthetic data can improve their performance on specific tasks [5]. We focus on fine-tuning HPC Coder v2 with purely synthetic data that is artificially generated using multiple state-of-the-art LLMs. In order to collect more diverse data samples, we use open source codebases to collect seed text snippets to inspire the LLM. Our prompt template is divided into four different categories: programming, translation, optimization and parallelization problems. By using different kinds of prompt templates we intend to increase the model's competence in solving different kinds of problems and make it more creative. The LLM is then tasked with generating a problem statement and solution based on the prompt. This process is highlighted in Figure 1. An example response can be seen in Figure 2. Our final dataset, HPC-Instruct, consists of over 120k samples of data generated with Gemini-Pro, DBRX, Llama-3-70B, and Mixtral-7B. By leveraging these models, we collect a large amount of data that encapsulates HPC tasks which will help us improve HPC Coder v2's understanding of HPC coding problems.

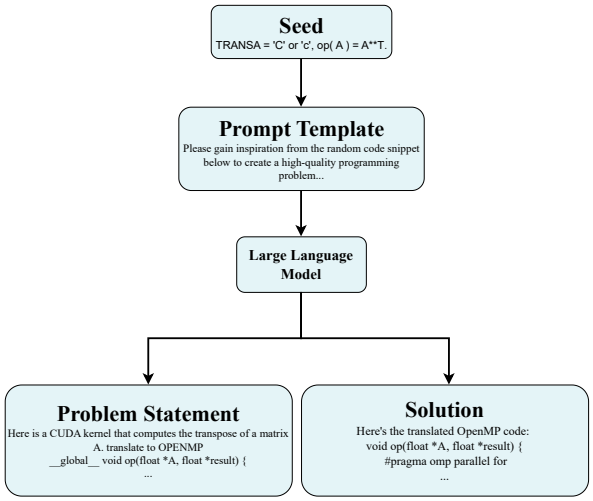

Figure 2: Example synthetic data generation output.

2 Fine-Tuning

Using HPC-Instruct we finetune Deepseek-Coder-1.3B and Deepseek-Coder-6.7B [2] with AxoNN [4] on four Perlmutter nodes with four 80GB A100 GPUs per node. The fine-tuning experiments result in four variations of each model: the base and instruct models, with and without instruction masking. Instruction masking is a technique where parts of the input data are intentionally hidden from the LLM during fine-tuning. The models are fine-tuned in bfloat16 with a batch size of 128 and sequence length of 8192 for two epochs. The total fine-tuning time was 208 node hours for all eight models Figure 4.

3 Evaluation Results

In this section we evaluate how HPC Coder v2 performs against other models in various code generation tasks.

3.1 Generation of Serial and Parallel Code

We use ParEval [3] and HumanEval [1] to evaluate the models on serial and parallel code generation. The results between masked and unmasked models do not show a consistent pattern. Fine-tuning the base models results in the best parallel code performance, but the

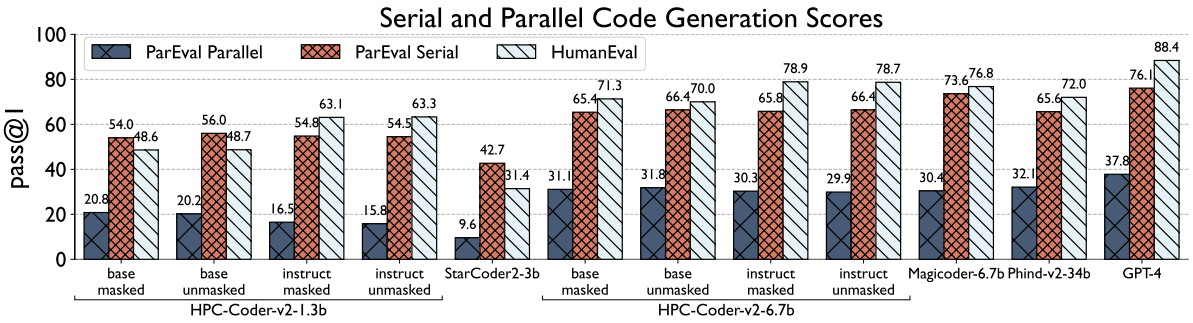

Figure 3: ParEval and HumanEval results

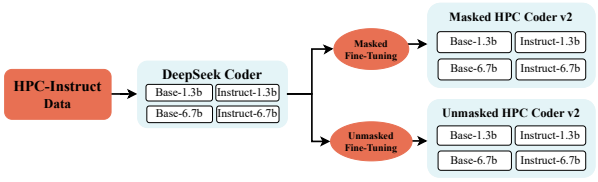

Figure 4: Fine-tuning process

worst HumanEval performance. One explanation for this could be that DeepSeek instruct's fine-tuning dataset is contaminated with HumanEval prompts. HPC Coder v2 6.7B is the best open-source LLM under 30B parameters at parallel code generation Figure 3.

3.2 HumanEval Performance for Different Languages

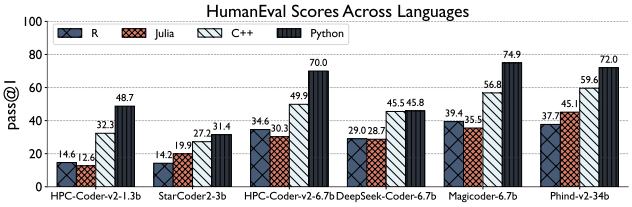

Figure 5: HumanEval performance for different languages

The unmasked base models are chosen as the final HPC Coder v2 models to be evaluated for different languages. Figure 5 shows that the 1.3B model performs better than StarCoder2 3B model on all languages except Julia. We can also see that the 6.7B model performs better than DeepSeek Coder 7B on all languages and is comparable to Phind v2 34B for Python and R. We also evaluate the models on inference and throughput in Figure 6. The HPC Coder models are faster and use less memory than other models while having comparable or better parallel code generation.

4 Conclusion

In this poster we presented the HPC Coder v2 models which are trained on purely HPC-related coding tasks. HPC Coder v2 6.7B is

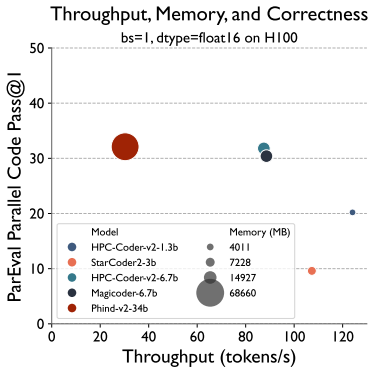

Figure 6: Throughput

evaluated to be the best performing open-source model for parallel code generation under 30B parameters. In future work we will generate more synthetic data for HPC and fine-tune bigger models to improve the model's performance. We would also like to evaluate HPC Coder v2 on other benchmarks. We believe that HPC Coder v2 can act as a stepping stone in how HPC developers and software engineers can efficiently write, maintain and scale HPC code and help them improve productivity and efficiency of parallel programs.

References

[1] Mark Chen and et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:arXiv:2107.03374

[2] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. arXiv:2401.14196 [cs.SE]

[3] Daniel Nichols, Joshua H. Davis, Zhaojun Xie, Arjun Rajaram, and Abhinav Bhatele. 2024. Can Large Language Models Write Parallel Code?. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24)*. Association for Computing Machinery, New York, NY, USA.

[4] Siddharth Singh and Abhinav Bhatele. 2022. AxoNN: An asynchronous, message-driven parallel framework for extreme-scale deep learning. In *Proceedings of the IEEE International Parallel & Distributed Processing Symposium (IPDPS '22)*. IEEE Computer Society.

[5] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source Code Is All You Need. *arXiv preprint arXiv:2312.02120* (2023).