POSTER: Learning to Parallelize in a Shared-Memory Environment with Transformers

Re’em Harel
reemha@bgu.ac.il
Department of Computer Science,
Ben-Gurion University of the Negev
Department of Physics, Scientific Computing Center, Nuclear Research Center – Negev, Israel

Yuval Pinter
uvp@cs.bgu.ac.il
Department of Computer Science,
Ben-Gurion University of the Negev,
Israel

Gal Oren
galoren@cs.technion.ac.il
Department of Computer Science,
Technion – Israel Institute of Technology
Scientific Computing Center, Nuclear Research Center – Negev, Israel

Abstract
In past years, the world has switched to multi and many core shared memory architectures. As a result, there is a growing need to utilize these architectures by introducing shared memory parallelization schemes, such as OpenMP, to applications. Nevertheless, introducing OpenMP work-sharing loop construct into code, especially legacy code, is challenging due to pervasive pitfalls in management of parallel shared memory. To facilitate the performance of this task, many source-to-source (S2S) compilers have been created over the years, tasked with inserting OpenMP directives into code automatically. In addition to having limited robustness to their input format, these compilers still do not achieve satisfactory coverage and precision in locating parallelizable code and generating appropriate directives. In this work, we propose leveraging recent advances in machine learning techniques, specifically in natural language processing (NLP), to suggest the need for an OpenMP work-sharing loop directive and data-sharing attributes clauses — the building blocks of concurrent programming. We train several transformer models, named PragFormer, for these tasks and show that they outperform statistically-trained baselines and automatic source-to-source (S2S) parallelization compilers in both classifying the overall need for an parallel for directive and the introduction of private and reduction clauses. In the future, our corpus can be used for additional tasks, up to generating entire OpenMP directives. The source code and database for our project can be accessed on GitHub 1 and HuggingFace 2.

Keywords: Machine learning, Concurrent computing methodologies

1 Introduction
Since the end of Dennard’s scaling [10], the world shifted towards shared memory architectures, in which multiple processors (cores) share the same memory address space. As a result, developers needed to adapt their applications to include the appropriate parallelization schemes thus, fully utilizing this architecture and increasing the performance. For example, the most comprehensive API that implements these parallelization schemes is the OpenMP API [7, 9]. The OpenMP API consists of a set of compiler directives (pragmas), library routines, and environment variables that allows a program to be executed in parallel (multi-threaded) within a shared memory environment.

Introducing OpenMP work-sharing loop construct to applications is not an easy task, even for parallel programming professionals, as it requires understanding and comprehending the dependencies between variables and other pervasive pitfalls in managing parallel shared memory [1].

To facilitate the performance of introducing parallel work-sharing loop construct (see [7] section 11.5), many S2S parallel compilers have been created over the years, tasked with inserting these constructs into code automatically. Nevertheless, it was shown in [12, 14, 15] that the S2S compilers have many pitfalls, such as producing suboptimal directives, compared to human experts; degrade performance; have limited robustness to their input; and at times, fail to insert a directive at all.

Due to the recent success of complex transformers-based architectures in the Code Language Processing (CLP) [6, 8], new useful products, or code advisors, that enable long, complex, and relevant code suggestions (either by code completion or code generation) have emerged. These advisors, such as Github’s Copilot 3 [2], use NLP-based models to improve the developers’ productivity by suggesting predicted code on-the-fly in IDEs.

Nevertheless, these advisors do not attempt to improve code performance by suggesting parallelization schemes.

1https://github.com/Scientific-Computing-Lab-NRCN/PragFormer
2https://huggingface.co/spaces/Pragformer/PragFormer-demo
3https://github.com/features/copilot
Therefore, the possibility of creating a similar code advisor based on NLP models that attempt to improve the code’s performance by suggesting OpenMP directives rises. The ability to create such an advisor is possible since OpenMP, and similar parallelization schemes, can be introduced incrementally to code (loop by loop), i.e., the directives can be introduced during development or to existing production (legacy) code.

In this paper, we present PragFormer (Figure 1), a transformer-based method for facilitating code parallelization through OpenMP, demonstrating its improved performance over S2S systems, as well as the necessity of its sophisticated model architecture. The model is tasked with identifying parallel work-sharing loop construct (for clarity, an OpenMP directive over a for-loop) and identifying data-sharing attributes clauses i.e., a private or a reduction clause.

![Figure 1. Overview of the workflow for classifying parallel work-sharing loop constructs and clauses. PragFormer is our proposed model.](image)

### 2 Database

In order to train the model, we created a database, or corpus, of code snippets, which we call Open-OMP. The github.com was queried with the phrase “OpenMP” to extract the relevant repositories, written in C, that most likely contain parallel work-sharing loop constructs. Then, only OpenMP directives defined over a for-loop segment were included in the database as positive data. While negative data examples were taken from snippets of code without OpenMP directives appearing in files where elsewhere such directives do exist, to rule out cases where code amenable to directives was not annotated due to developers unfamiliar with parallelization schemes or those who work with incompatible hardware. The following Table 1 contains the statistics of the database.

| Description | Amount | # Line | Amount |
|-------------|--------|--------|--------|
| Total code snippets | 32,099 | < 10 | 17,836 |
| OpenMP directives | 14,906 | 11–50 | 13,046 |
| reduction | 2,147 | 51–100 | 782 |
| private | 5,568 | > 100 | 435 |

(a) Statistics of the OpenMP directives on the raw database. (b) Code snippet lengths in the raw database.

#### Table 1. General statistics related to the database.

3 Results

PragFormer identifies locations (for-loops) that can benefit from a work-sharing loop construct and not the directive. Furthermore, the premise of this work is that the annotated positive labels were produced by a human expert, thus the label corresponds with the best performance achieved by the directive. Therefore, the evaluation of PragFormer is performed by measuring the number of correct predictions regarding the actual label and not by performance.

For evaluation, the precision, recall, and F1 score [11] is calculated on the test set as the performance measurements. The results of PragFormer are compared to a novel S2S compiler ComPar [13] and a statistical trained Bag-of-Words (BoW) [16] model with a logistic regression classifier. The first rows of Table 2 present the performance of PragFormer, BoW and ComPar on the directive classification task. According to all measurements, the best results are achieved by PragFormer.

The second and third rows in Table 2 present the results for identifying the need for a private and/or a reduction data-sharing attributes clauses. PragFormer produces excellent recall and precision for both clauses, demonstrating a good balance between finding many true cases without allowing many false predictions to sift through.

In order to test the generality of PragFormer, we apply it to two existing dedicated OpenMP benchmarks that do not appear in Open-OMP: PolyBench [4] and the Standard Performance Evaluation Corporation (SPEC-OMP) [5]. Table 2 presents the results of PragFormer and ComPar on PolyBench and SPEC-OMP. The results of PragFormer are comparable to, and even slightly better than, the ones over the Open-OMP test set.

| Test | Model | P    | R    | F1   | Acc  |
|------|-------|------|------|------|------|
| Directive | BoW + Logistic | 0.71 | 0.71 | 0.71 | 0.71 |
|       | ComPar  | 0.51 | 0.56 | 0.36 | 0.5 |
|       | PragFormer | 0.86 | 0.85 | 0.86 | 0.85 |
|       | BoW + Logistic | 0.79 | 0.78 | 0.78 | 0.79 |
|       | ComPar  | 0.56 | 0.51 | 0.40 | 0.56 |
|       | PragFormer | 0.89 | 0.87 | 0.87 | 0.87 |
|       | BoW + Logistic | 0.78 | 0.78 | 0.77 | 0.78 |
|       | ComPar  | 0.92 | 0.52 | 0.76 | 0.79 |
| Poly | PragFormer | 0.95 | 0.95 | 0.95 | 0.95 |
|       | ComPar  | 0.43 | 0.43 | 0.43 | 0.43 |
| SPEC | PragFormer | 0.83 | 0.84 | 0.82 | 0.83 |
|       | ComPar  | 0.76 | 0.75 | 0.74 | 0.75 |

#### Table 2. Comparison between PragFormer and the competing systems on the task of identifying the need for an OpenMP directive (top three rows); the private clause identification; the reduction clause identification; the need for an OpenMP directive in the PolyBench and SPEC-OMP suite.
Acknowledgement

This research was supported by the Israeli Council for Higher Education (CHE) via the Data Science Research Center, Ben-Gurion University of the Negev, Israel, the Lynn and William Frankel Center for Computer Science, and Intel Corporation (oneAPI Center of Excellence program). Computational support was provided by the NegevHPC project [3].

References

[1] [n. d.]. Automatic Parallelism and Data Dependency. https://web.archive.org/web/20140714111836/http://blitzprog.org/posts/automatic-parallelism-and-data-dependency. [Online].
[2] [n. d.]. GitHub Copilot. https://github.com/features/copilot. [Online].
[3] [n. d.]. NegevHPC Project. https://www.negevhpc.com. [Online].
[4] [n. d.]. PolyBench Benchmarks. https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/. [Online].
[5] [n. d.]. SPEC-HPG. https://www.spec.org/order.html. [Online].
[6] Moritz Beller, Hongyu Li, Vivek Nair, Vijayaraghavan Murali, Imad Ahmad, Jurgen Cito, Drew Carlson, Ari Aye, and Wes Dyer. 2022. Learning to Learn to Predict Performance Regressions in Production at Meta. arXiv preprint arXiv:2208.04351 (2022).
[7] OpenMP Architecture Review Board. 2021. OpenMP Application Programming Interface Version 5.2. Independently published.
[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).
[9] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. IEEE computational science and engineering 5, 1 (1998), 46–55.
[10] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. 1974. Design of ion-implanted MOSFET’s with very small physical dimensions. IEEE Journal of Solid-State Circuits 9, 5 (1974), 256–268.
[11] Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In European conference on information retrieval. Springer, 345–359.
[12] Re’em Harel, Idan Mosseri, Harel Levin, Lee-or Alon, Matan Rusanovsky, and Gal Oren. 2020. Source-to-source parallelization compilers for scientific shared-memory multi-core and accelerated multiprocessing: analysis, pitfalls, enhancement and potential. International Journal of Parallel Programming 48, 1 (2020), 1–31.
[13] Idan Mosseri, Lee-or Alon, Re’em Harel, and Gal Oren. 2020. ComPar: Optimized Multi-compiler for Automatic OpenMP S2S Parallelization. In OpenMP: Portable Multi-Level Parallelism on Modern Systems - 16th International Workshop on OpenMP, IWOMP 2020, Austin, TX, USA, September 22-24, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12295), Kent F. Milfeld, Bronis R. de Supinski, Lars Koesterke, and Jannis Klinkenberg (Eds.). Springer, 247–262. https://doi.org/10.1007/978-3-030-58164-2_16
[14] S Prema, R Jehadeesan, and BK Panigrahi. 2017. Identifying pitfalls in automatic parallelization of NAS parallel benchmarks. In Parallel Computing Technologies (PARCOMPTECH), 2017 National Conference on. IEEE, 1–6.
[15] S Prema, Rupesh Nasre, R Jehadeesan, and BK Panigrahi. 2019. A study on popular auto-parallelization frameworks. Concurrency and Computation: Practice and Experience 31, 17 (2019), e5168.
[16] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics 1, 1 (2010), 43–52.