Learning to Parallelize Source Code via OpenMP with Transformers

Re'em Harel[1, 2, 3], Yuval Pinter[1] and Gal Oren[2, 4] ✉

[1]Department of Computer Science, Ben-Gurion University of the Negev, Israel
[2]Scientific Computing Center, Nuclear Research Center – Negev, Israel
[3]Department of Physics, Nuclear Research Center – Negev, Israel
[4]Department of Computer Science, Technion – Israel Institute of Technology, Israel
reemha@bgu.ac.il, uvp@cs.bgu.ac.il, galoren@cs.technion.ac.il

***Abstract*—The world has switched to many-core and multi-core shared memory architectures. As a result, there is a growing need to utilize these architectures by introducing shared memory parallelization schemes to software applications. OpenMP is the most comprehensive API that implements such schemes, characterized by a readable interface. Nevertheless, introducing OpenMP into code, especially legacy code, is challenging due to pervasive pitfalls in management of parallel shared memory. 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 directive and even suggest specific clauses. The model can also provide an immediate, on-the-fly, advice to the developer without compiling or executing the code. We created a novel database, OPEN-OMP, specifically for this goal and train a transformer model, named PRAGFORMER, for these tasks and show that they outperform statistically-trained baselines and automatic source-to-source parallelization. Our source code and database are available at: https://github.com/pragformer/PragFormer.**

I. INTRODUCTION

The world switched from single core architecture to multi-core and many-core architecture that share the same address space (shared memory). Due to the constant need to increase the performance of applications, software developers needed to adjust their serial or distributed applications and introduce shared-memory parallelization schemes, such as OpenMP [1], to their applications. However, introducing OpenMP directives to existing applications is challenging due to pervasive pitfalls in management of parallel shared memory environments [2]. To ease the burden of this task, source-to-source (S2S) parallelization compilers were created over the years that aim to insert OpenMP directives [3], [4]. Nevertheless, it was shown in [5], [6], [4] 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.

In this work, we propose using transformers based architecture [7] to identify segments needing an OpenMP directive, and even suggest specific OpenMP clauses. Due to the negligible inference time (contrary to S2S compilers), the model can also be used as an immediate "advisor" for developers to identify locations that can benefit from an OpenMP directive.

II. CORPUS

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 OpenMP directives. Then, only OpenMP directives defined over a 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. Each entry in the database contains the loop-segment, the OpenMP directive (if exists) and an AST representation of both. The following Table I contains the statistics of the database.

Description	Amount
Total code snippets	17,013
For loops with OpenMP directives	7,630
Reduction	1,455
Private	3,403

TABLE I
STATISTICS OF THE OPENMP DIRECTIVES ON THE RAW DATABASE.

The data-validation-test sets are divided into an 80%–10%–10% ratio.

III. PROPOSED APPROACH

We propose a novel model, PRAGFORMER (Figure 1), for identifying OpenMP directives and clauses. The transformer architecture [7] has gained popularity due to its impressive ability to leverage data from vast unlabeled sequence resources in a *pre-training* phase and apply it to a large range of end tasks, a property known as *transfer learning*. At a high level, the transformer is a model which receives a sequence of vectors as its input and passes it through an *encoder* module, which outputs a sequence of context-dependent vectors. These, in turn, are fed into a *decoder* that can perform a classification task such as the one at hand. In the encoder, the main mechanism allowing this behaviour is the *self-attention* component. This component calculates a numeric score for each position in the input sequence with respect to the other ones based on their input vectors. The score then determines the level of consideration in preparing this position's output for ingestion by the next layer. In the decoder, the attention mechanism is also applied to items from the input, enabling *cross-attention* [8]. In source code, attention scores may represent how much influence each of the other variables or statements have over a given input.

We implement PRAGFORMER based on the pre-trained state-of-the-art DeepSCC [9] model, which itself is based on fine-tuning RoBERTa [10]. Similar to our task, which is identifying and advising the need for an OpenMP directive, DeepSCC was trained to classify the programming language of a source code. The self-attention mechanism is crucial in our task, as identifying the need for an OpenMP directive is hinted in long-range dependencies in the source code, to which the transformer model's self-attention mechanism is most suited.

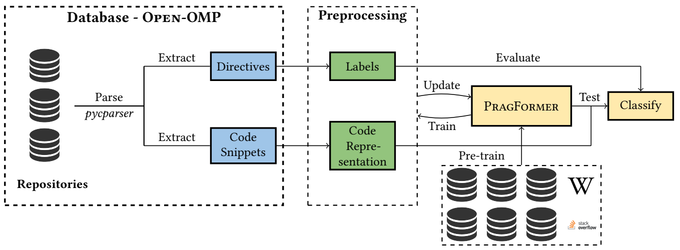

Fig. 1. Overview of the workflow for classifying OpenMP directives and clauses. PRAGFORMER is our proposed model.

Raw text (or source code) cannot serve as an input to an ML model; instead, transforming the code to a sequence of tokens from a pre-set vocabulary is needed. This transformation process, known as *tokenization*, is then followed by associating each keyed token with a numerical vector (*embedding*) which is used as input to the model, to be modified along with the rest of the model's parameters during training. Previous work [11], [12], [13], [14] suggests that representing the source code as an AST produces better results than text. However, the tokenizer and the pre-trained model (DeepSCC and RoBERTa) were trained exclusively on natural language text. Therefore, exploiting the transfer learning property with different representations other than text will likely produce poor results.

IV. RESULTS

For evaluation, the precision, recall, and F1 score [15] is calculated on the test set as the performance measurements. The results of PRAGFORMER are compared to a novel S2S compiler COMPAR [16] and a statistical trained Bag-of-Words (BoW) [17] model with a logistic regression classifier. The first rows of Table II 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 II present the results for identifying the need for a *private* clause and the need for a *reduction*. 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 [18] and the Standard Performance Evaluation Corporation (SPEC) [19]. Table II presents the results of PRAGFORMER and COMPAR on PolyBench and SPEC-OMP. However, COMPAR failed to parse 287 snippets from the SPEC-OMP benchmark mainly due to unrecognized keywords, such as *register*. Thus, we exclude them from COMPAR's results. 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	**PRAGFORMER**	**0.80**	**0.81**	**0.80**	**0.80**
	BoW + Logistic	0.73	0.74	0.73	0.74
	COMPAR	0.51	0.56	0.36	0.5
private	**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
reduction	**PRAGFORMER**	**0.89**	**0.87**	**0.87**	**0.87**
	BoW + Logistic	0.78	0.78	0.77	0.78
	COMPAR	0.92	0.52	0.46	0.79
Poly	**PRAGFORMER**	**0.93**	**0.93**	**0.93**	**0.93**
	COMPAR	0.43	0.43	0.43	0.43
SPEC	**PRAGFORMER**	**0.81**	**0.80**	**0.80**	**0.80**
	COMPAR	0.76	0.75	0.74	0.75

TABLE II
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 SUITE; AND THE NEED FOR AN OPENMP DIRECTIVE IN THE SPEC-OMP SUITE.

V. CONCLUSION & FUTURE WORK

In this work, we presented a novel data-driven model, named PRAGFORMER, that aims to classify the need of an OpenMP directive and OpenMP clauses. PRAGFORMER manages to perform fundamental tasks on the roadmap to parallelization better than two competitors—a top-shelf deterministic S2S compiler and a lightweight text-aware ML model.

In most cases, the context of the loop itself is enough to generate an OpenMP directive. However, to push performance further, it is effective to include context from previous and next lines of code. Thus, we will expand the database containing these extra lines of code. Moreover, additional code snippets from C++, and Fortran will be added to the database. In addition, to improve the effectiveness of PRAGFORMER, we plan to explore additional code representations. Most importantly, PRAGFORMER provides a crucial step towards a more robust framework based on NLP techniques that will eventually be able to insert the full OpenMP directives automatically with high fidelity.

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 [20].

REFERENCES

[1] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[2] "Automatic Parallelism and Data Dependency," https://web.archive.org/web/20140714111836/http://blitzprog.org/posts/automatic-parallelism-and-data-dependency, [Online].

[3] H. Yao, H. Deng, and C. Zou, "A survey of loop parallelization: Models, approaches, and recent developments," *International Journal of Grid and Distributed Computing*, vol. 9, no. 11, pp. 143–156, 2016.

[4] S. Prema, R. Nasre, R. Jehadeesan, and B. Panigrahi, "A study on popular auto-parallelization frameworks," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, p. e5168, 2019.

[5] R. Harel, I. Mosseri, H. Levin, L.-o. Alon, M. Rusanovsky, and G. Oren, "Source-to-source parallelization compilers for scientific shared-memory multi-core and accelerated multiprocessing: analysis, pitfalls, enhancement and potential," *International Journal of Parallel Programming*, vol. 48, no. 1, pp. 1–31, 2020.

[6] S. Prema, R. Jehadeesan, and B. Panigrahi, "Identifying pitfalls in automatic parallelization of nas parallel benchmarks," in *Parallel Computing Technologies (PARCOMPTECH), 2017 National Conference on*. IEEE, 2017, pp. 1–6.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[9] G. Yang, Y. Zhou, C. Yu, and X. Chen, "Deepscc: Source code classification based on fine-tuned roberta," *CoRR*, vol. abs/2110.00914, 2021. [Online]. Available: https://arxiv.org/abs/2110.00914

[10] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[11] V. Jayasundara, N. D. Q. Bui, L. Jiang, and D. Lo, "Treecaps: Tree-structured capsule networks for program source code processing," *arXiv preprint arXiv:1910.12306*, 2019.

[12] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 200–20 010.

[13] K. Li, Z. Liu, T. He, H. Huang, F. Peng, D. Povey, and S. Khudanpur, "An empirical study of transformer-based neural language model adaptation," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7934–7938.

[14] S. Haque, A. LeClair, L. Wu, and C. McMillan, "Improved automatic summarization of subroutines via attention to file context," *CoRR*, vol. abs/2004.04881, 2020. [Online]. Available: https://arxiv.org/abs/2004.04881

[15] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European conference on information retrieval*. Springer, 2005, pp. 345–359.

[16] I. Mosseri, L. Alon, R. Harel, and G. Oren, "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*, ser. Lecture Notes in Computer Science, K. F. Milfeld, B. R. de Supinski, L. Koesterke, and J. Klinkenberg, Eds., vol. 12295. Springer, 2020, pp. 247–262. [Online]. Available: https://doi.org/10.1007/978-3-030-58144-2_16

[17] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1, pp. 43–52, 2010.

[18] "PolyBench Benchmarks," https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/, [Online].

[19] "SPEC-HPG," https://www.spec.org/order.html, [Online].

[20] "NegevHPC Project," https://www.negevhpc.com, [Online].