Sequentializing Compiler-Based Graph Representations of Code for Machine Learning

Wenfei Tang, Fangyuan Yang, Cooper Stevens, and Samin Riasat
(Group 28)

Outline

- Introduction
- Related Work
- Method
- Evaluation
- Conclusion
- Future Work

Introduction

Graph representations of source code are good at informing on dependencies and execution. Examples include:

- Control data-flow graph (CDFG)
- ProGraML proposed representation
- etc.

Natural Language Processing (NLP) methods are well-researched, powerful tools for learning a variety of features about a piece of text. Examples include:

- DeepTune
- inst2vec
- etc.

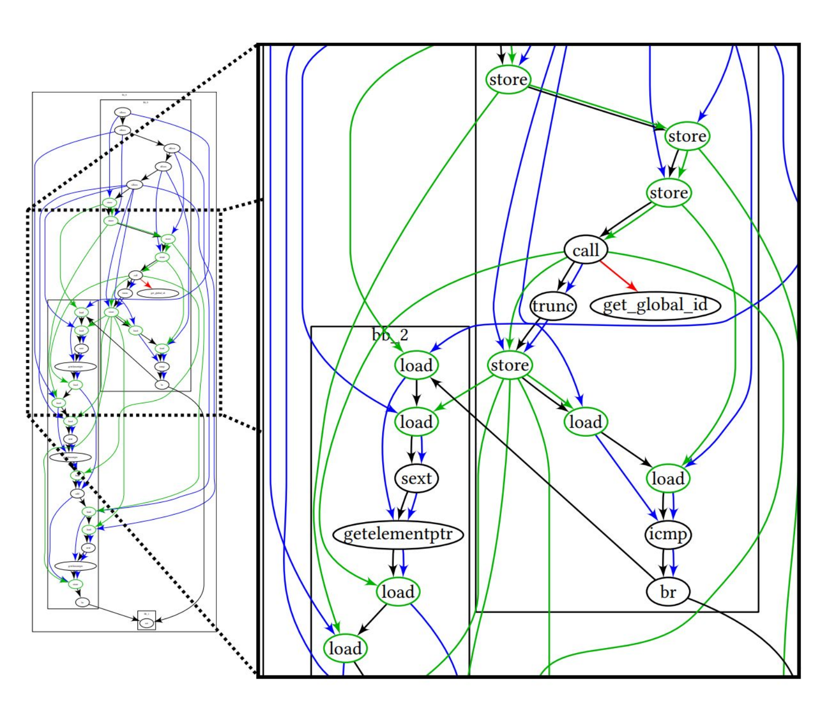
Solution Overview

Objective: We look to get the best of both worlds of the **information-rich graph representation** and the **powerful NLP methods**.

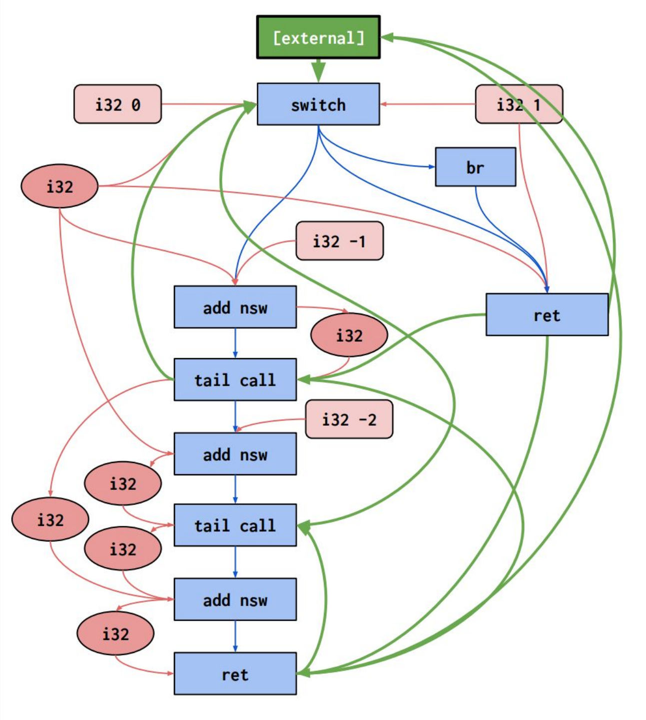
Issue: NLP methods cannot take graph representations as input, as the inputs are necessarily *sequential*.

Solution: *Sequentialize* the graph representation for use in NLP methods.

Representation for Optimization Heuristics



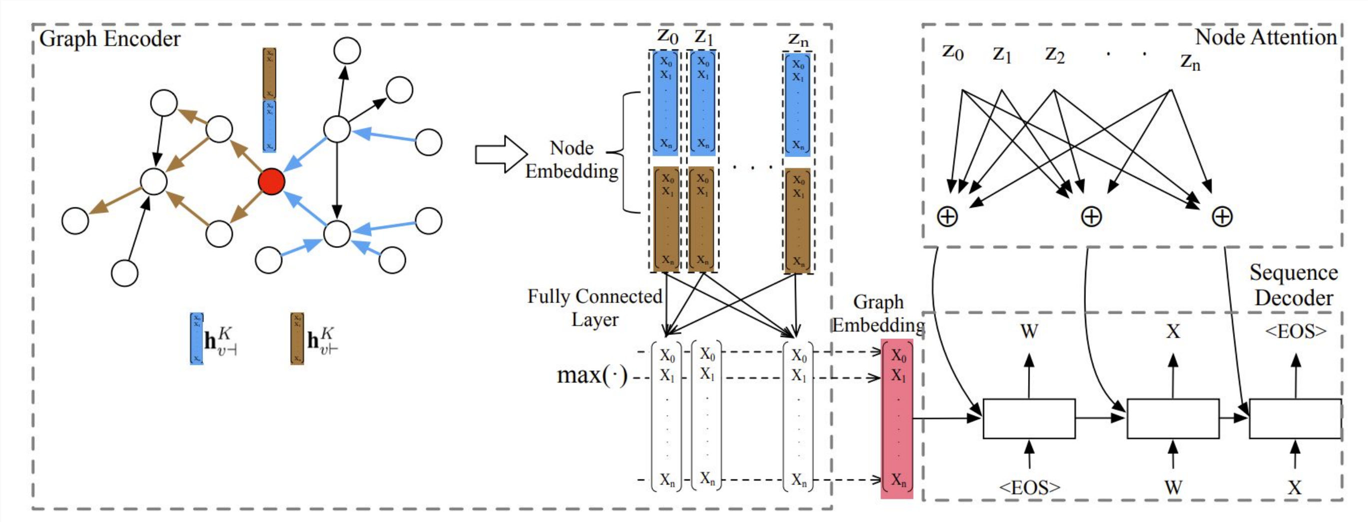
CDFG + CALL + MEM



ProGraML

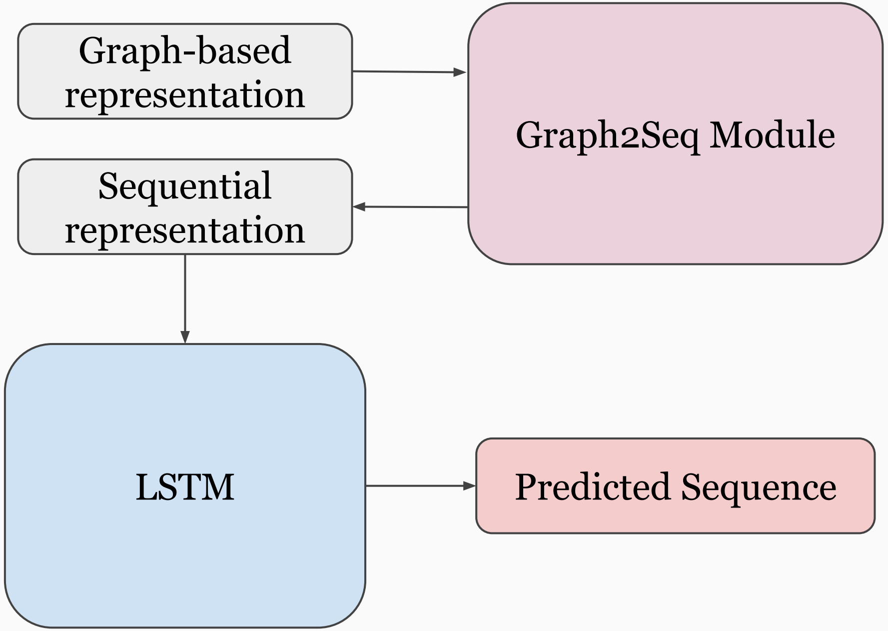
Graph-to-Sequence Model

- End-to-end graph-to-sequence model
- Maps an input graph to a sequence of vectors



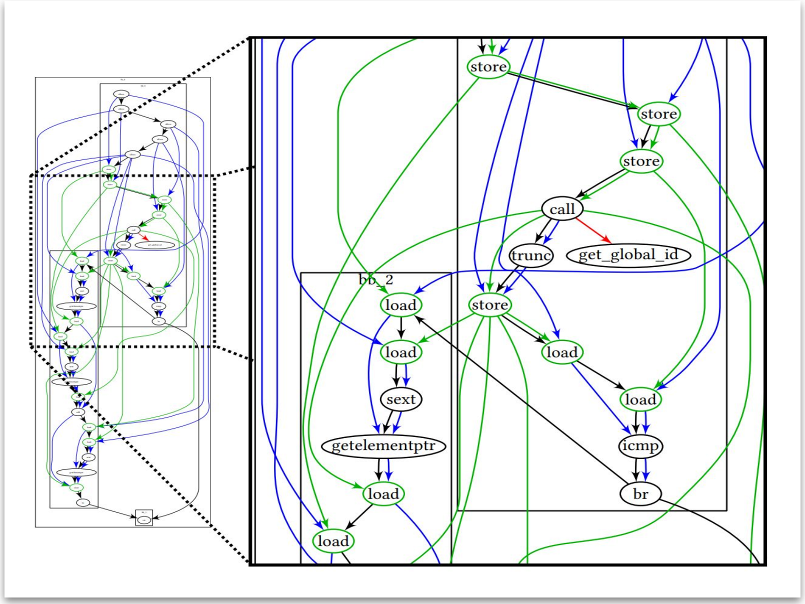
Our Work

- Build an architecture that can combine both graph representations and deep learning models for sequential representations



Graph Representation

- Generated based on the enriched LLVM-based control-and dataflow graph
- Four parts:
  - Nodes
  - Edges
  - Adjacency lists
  - Node attributes



CDFG + CALL + MEM

Evaluation: Intro to CPU/GPU Task

- Fix some heterogeneous hardware system (AMD Tahiti 7970 GPU)
- Input
  - The source code of a kernel
- Objective
  - Pick CPU or GPU to execute the kernel on
  - Consider the decision to be correct if we chose the hardware that would result in a faster execution

```
__kernel void Add(__global const int* x,
                  __global const int* y,
                  __global int* z, const int d) {
 const int id = get_global_id(0);
 if (id < d)
   z[id] = x[id] + y[id];
}
```

Example kernel

Model Parameters

- Parameters include:
  - Embedding dimension
  - Training batch size
  - Training epochs
  - Layer size
- Embedding dimension and layer size are proportional to model complexity.
- Training batch size and training epochs determine the model's exposure to the training data.

Evaluation Results

- Representations: **CDFG**, ProGraML
- Embedding dimension
  - 10 good enough for compiler-based representation
- Converges fast with decent performance
  - We trained only 100 epochs for 75% accuracy
  - DeepTune: 1000 epochs for 79% accuracy

TABLE I
ACCURACY OF GNN-BASED MODELS FOR DIFFERENT CHOICES OF HYPER-PARAMETERS

Model	Node embedding dimension	Train batch size	Sample layer size	Number of training epochs	Accuracy on test set
CDFG	10	32	4	100	0.75
CDFG	10	16	4	100	0.74
CDFG	50	32	4	100	0.71
CDFG	10	32	8	32	0.66
CDFG	150	32	4	100	0.70
CDFG	100	32	8	100	0.72
CDFG	100	32	16	100	0.70
PROGRAML	10	32	1	100	0.65
PROGRAML	100	4	4	100	0.67

TABLE II
COMPARISON OF GNN-BASED MODELS WITH LSTM-BASED MODELS [3]

	Model	Accuracy
GNN	CDFG	0.75
	PROGRAML	0.67
LSTM	DEEPTUNE [5]	0.79
	Barchi et al. [1]	0.76

Conclusion

- The proposed serialization of compiler-based graph representations seems reasonable in that it provides reasonably good performance on the CPU/GPU task
- However, we have also seen that this representation was unable to outperform state-of-the-art models for the same task (e.g. DeepTune)
- We recognize that these results are premature to consider this investigation fully closed
    - These results were produced under relatively restricted conditions (Google Colab) whereas the state-of-the-art results were likely to have had access to more advanced hardware

Future Work

- Therefore, further investigation into the utility of our proposed representation may be worth while in future work, as the results have proven to be promising thus far
- One might look to research what improvements can be made to these models by introducing additional model complexity (which would require more advanced hardware than we had access to)
  - Sequentialize such that input is compatible with advanced models like DeepTune
- One might also look into other methods of serializing graph representations
  - For example, one may look into ways of encoding edge information into a serial representation of a ProGraML graph

References

[1] F. Barchi, G. Urgese, E. Macii, and A. Acquaviva, "Code mapping in heterogeneous platforms using deep learning and llvm-ir," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3317789

[2] A. Brauckmann, A. Goens, S. Ertel, and J. Castrillon, "Compiler-Based Graph Representations for Deep Learning Models of Code," in *Proceedings of the 29th International Conference on Compiler Construction*, 2020, pp. 201–211.

[3] A. Brauckmann, A. Goens, and J. Castrillon, "Compy-learn: A toolbox for exploring machine learning representations for compilers," in *2020 Forum for Specification and Design Languages (FDL)*, 2020, pp. 1–4.

[4] C. Cummins, Z. Fisches, T. Ben-Nun, T. Hoefler, M. O'Boyle, and H. Leather, "ProGraML: A Graph-based Program Representation for Data Flow Analysis and Compiler Optimizations," in *Thirty-eighth International Conference on Machine Learning (ICML)*, 2021.

[5] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, "End-to-End Deep Learning of Optimization Heuristics," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 219–232.

[6] K. Xu, L. Wu, Z. Wang, Y. Feng, M. Witbrock, and V. Sheinin, "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks," *arXiv preprint arXiv:1804.00823*, 2018.

Thanks for listening!

Do you have any questions?